Veto-based Malware Detection

Raja Khurram Shahzad, Niklas Lavesson

Seventh International Conference on Availability, Reliability and Security

2012 Prague

2012 Seventh International Conference on Availability, Reliability and Security

Veto-based Malware Detection

Raja Khurram Shahzad
School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
Email: rks@bth.se

Niklas Lavesson
School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
Email: niklas.lavesson@bth.se

***Abstract*—Malicious software (malware) represents a threat to the security and privacy of computer users. Traditional signature-based and heuristic-based methods are unsuccessful in detecting some forms of malware. This paper presents a malware detection approach based on supervised learning. The main contributions of the paper are an ensemble learning algorithm, two pre-processing techniques, and an empirical evaluation of the proposed algorithm. Sequences of operational codes are extracted as features from malware and benign files. These sequences are used to produce three different data sets with different configurations. A set of learning algorithms is evaluated on the data sets and the predictions are combined by the ensemble algorithm. The predicted output is decided on the basis of veto voting. The experimental results show that the approach can accurately detect both novel and known malware instances with higher recall in comparison to majority voting.**

***Keywords*-Malware; scareware; detection; veto voting; feature extraction; classification; majority voting; ensembles.**

I. Introduction

Malicious software (malware) is a common computer threat and is usually addressed through static and dynamic detection techniques. Anti-malware tools are only able to detect known instances and the success rate is circa 30 % [1]. In an effort to extend both the static and dynamic approaches, some researchers apply machine learning (ML) algorithms, which show promising results both in detecting known and new malware. To increase the detection accuracy, the output of multiple algorithms is combined (based on different parameters) and a decision is taken on the basis of majority voting. The decision taken by the majority voting can be wrong if majority of classifiers classify malware instances as benign. As a consequence, a ML-based detection model is proposed in which the inductive biases of multiple algorithms are combined and the final prediction is given on the basis of veto voting. The experimental results show that the proposed veto-based classification can be used to accurately detect both novel and known instances of malware, and performs better than majority voting at least for the studied problems.

A. Aim and Scope

The aim is to investigate a malware detection approach that combines the output of a set of classifiers and provides a classification on the basis of veto voting. A malware detection application is developed, which implements both veto voting and majority voting. The results from veto voting are compared with the results from majority voting. To achieve better predictive results, quality of information derived from the data in pre-processing is very important. Therefore, two pre-processing techniques are also proposed and investigated.

B. Contribution

The research contributions in this paper are as follow. First a malware detection model is proposed and implemented which combines the inductive biases of different algorithms and uses veto voting for the prediction. Second, two pre-processing techniques are proposed to extract features from executables. These pre-processing techniques can be used to extract both hexa-decimal based features or assembly instruction based features of different sizes ranging from small to large sizes.

C. Outline

The remainder of the article is organized as follows: Section II discusses the background, terminology and related work. Section III presents the veto-based classification by discussing its architecture. Section IV discusses the pre-processing techniques. Section V describes the experimental procedure. In Section VI, the results are discussed and analyzed. Finally Section VII concludes the work and describes future directions.

II. Background

One of the challenges faced by computer users is to keep the data and communication away from unwanted parties who exploit vulnerabilities present in the operating system (OS) or third party software to jeopardize communication and access data. A popular way to exploit vulnerabilities remotely is by using a malware [2]. Traditionally, malware detection is conducted either by using static analysis, i.e., by matching specific patterns called signatures or on the basis of a rule set, or dynamic analysis, i.e., changes occurring in the system due to the execution of a specific file. The main

978-0-7695-4775-6/12 $26.00 © 2012 IEEE
DOI 10.1109/ARES.2012.85

deficiency of these techniques is that they fail to detect zero day attacks.

The use of ML has been investigated in fields such as natural language processing, medical diagnosis, and malware detection. ML can be divided into two broad categories. In supervised learning, algorithms are used to generate a classifier or a regression function using labeled data. To achieve better predictive performance a finite set of classifiers can be combined together as an ensemble. The prediction of most ensembles is based on majority voting. If the data is incompletely labeled, unsupervised learning is used. To achieve better predictive performance in unsupervised learning, deep learning can be used. In deep learning, algorithms learn from multiple levels of representations to find complex patterns in the data. Any algorithm used in supervised or unsupervised learning has its own inductive bias. Inductive bias of learning algorithms refers to the set of assumptions that a learning algorithm uses for predicting the output of unseen inputs [3]. In other words, it is a set of assumptions that is used by learning algorithm to prefer one hypothesis over the other in the search space, in order to find a suitable hypothesis which can give better predictions for the problem in question. These factors affect classifier performance [4]. In the case of malware classification, an algorithm may be more accurate in classifying viruses than adware. Due to these reasons, detection results may vary from one learning algorithm to another learning algorithm. Therefore, it may be appropriate to join the results of classifiers trained at different representations to achieve improved accuracy in predicting the class of unseen instances.

A. Terminology

1) Data set: A piece of information from a particular object such as binary file, image, in a specific format is called its feature. The format of the feature is called feature representation and helps in performing computations task by using that particular feature. Each instance present in the original data is represented by its respective features for the ML process-able data sets. For the task of malware detection, different features can be used to represent the binary files such as files may be converted into hexa-decimal code [5] [6] or assembly instructions [7] to prepare the data sets. Features may also be extracted from the binary files such as printable strings or system calls [5] to prepare the data set.

2) Feature Selection: To improve the accuracy of ML algorithms, complexity of data sets is reduced. For this purpose, the most common approach is to apply a feature selection algorithm which measures the quality of each feature and prioritize features accordingly [4]. Only features that can provide valuable information for classification are kept and the rest are discarded. In malware detection, a large feature set is produced from the binary data set. This feature set contains many unimportant features which can degrade the performance of ML algorithm. Therefore, it is necessary to obtain a subset of valuable features by applying feature selection.

3) Classification: In ML, classification is divided into two stages, i.e., training and testing. Learning algorithms are applied on the training set to produce a model (commonly called classifier) or a regression function [3]. This stage is called training. During the testing stage, generated classifier is used to predict the class of unseen instances.

4) Ensemble Learners or Ensembles: Ensemble learners are capable of combining multiple models for improved accuracy [4]. The different models for the ensemble may be generated from the same base algorithm on different subsets of data or from different algorithms on the same data set. Ensembles perform better than a single model due to the diversity of base models.

B. Related Work

A significant amount of research for classification tasks has applied techniques ranging from statistical methods to machine learning like supervised learning and deep learning (DL). The use of DL has been investigated to learn multiple levels of representation in order to model complex relationships in the data to classify patterns and objects [8]. DL has also been used to extract features and represent them at different layers with different representations or abstraction to be used for vision, face recognition, and hand written digit recognition. Similarly, the layered architecture has also been used for detecting the malicious behavior [1]. In some cases the decision from the single model or even from multiple models may not be enough to have final predication especially when the cost of misclassifying one class is higher than misclassifying the other class. As a solution, a few researchers have used veto voting for automated identification of a disease pulmonary embolism [9] and for authorship identification [10]. The concept of veto voting is not investigated for malware detection. For malware detection researchers use ensembles which relies on majority voting. Therefore, it is worth to investigate the veto voting for malware detection.

III. Veto-based Classification

In certain situations, the decision from more than one expert may be required. In such cases, a committee of experts is formed as it is expected that a committee always performs better than a single expert. Normally committee uses majority voting for combining the decisions of the experts to reach a final conclusion. In some cases, the committee may give the right to veto the decision of the committee to any member. In ML, multiple algorithms can be used to generate multiple classifiers (experts) for single classification task. Every classifier has its own inductive bias which affects the predictive performance. Research results indicate that ensembles perform better than single classifier in fields of text categorization [11] and data classification,

etc. Several rules such as majority voting, i.e., bagging [4], weighted combination where weights represent effectiveness of member classifiers such as boosting [4], dynamic classifier selection [12], [13] and veto voting [9], [10] can be used for combining the decisions and having a final prediction. Veto voting is used to give importance to a single expert (classifier) who predicts against the majority.

In malware detection, ignoring the right prediction about a malicious file from single expert may cost higher in terms of security violations. The security violations may cause serious data loss, privacy or financial damages to the user. Therefore a veto voting based classification model is more applicable than a majority voting based model. A model is proposed which combines the inductive biases of different classifiers and final decision is given on the basis of veto voting. For brevity, the veto voting based classification system is referred as veto classifier in later sections.

A. Voting Rules

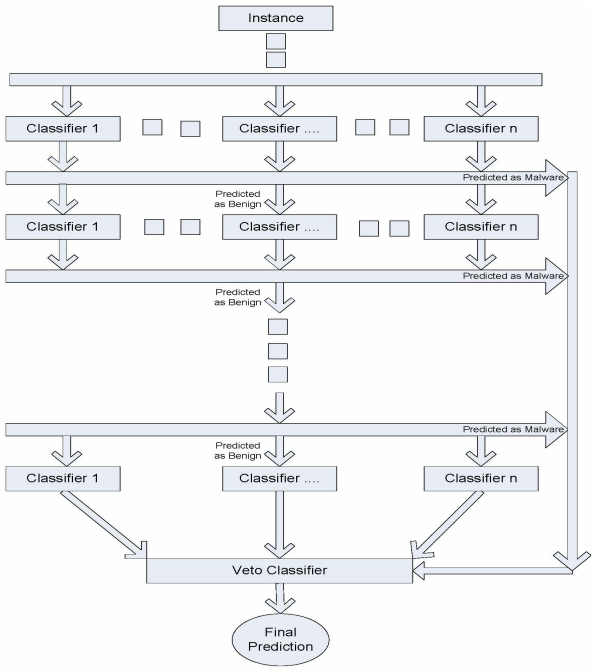
The main idea of veto based classification is to combine the multiple predictions to achieve a final decision. Formally, a veto based classification system consists of candidate classifiers (C) and vote set V. The set of candidate classifiers C and the set of votes (V) is a finite set (C, V) with predetermined fixed number of maximum classifiers and votes. The vote from each classifier is considered on the basis of rules given in Table I. Some rules mentioned in Table I are also recommended for the general voting.

Table I
VOTING RULES

Rule	Explanation
Anonymity	All votes in the vote set (V, a finite set with predetermined number of votes) are treated equally and the outcome of the classifier committee remains same with any permutation of the votes.
Neutrality	All candidates in the classifier set (C, a finite set with predetermined number of classifiers) are considered equally without any extra weighting.
Independence	Candidate classifiers are independent of each other and the result of the voting system remains same with any combination of classifiers with votes, i.e., $(C_1,V) \cup (C_2,V) \subseteq (C,V)$ where C_1 and C_2 are different combinations of classifiers.
Completeness	All votes from the classifiers are considered and counted only once.
Veto	Any vote indicating an instance as malware, alone can determine the outcome of the classification task regardless of the count of other votes.
Consistency	The result of the voting remains same even if the base classifiers are split into two disjoint sets and each set vote separately. The votes from each subset make a single vote set. Formally this can be mentioned as $(C,V_1) \cap (C,V_2) \subseteq (C,V)$ where V_1 and V_2 are the partitions of votes.
Usability	Voting scheme can be used for other similar problems.
Verifiability	Any user of the voting scheme can verify the outcome of voting by counting the votes manually also.

Ideally veto based voting performs better than single classifier because if any classifier in the committee predicts the class of an instance as malware, it veto all the other predictions from the other classifiers. For neutrality, the results from all the classifiers are combined without any additional weighting or filtering. It is also possible to use weighting mechanism [14] for the votes such as by assigning more weight to the vote of a classifier who outperformed all other classifiers in terms of accuracy during the training stage.

Figure 1. *N*-layer Implementation



B. Architectures

The model can be implemented in two possible ways.

1) N-Layers implementation: The model can be implemented in *n*-layers with any permutation of classifiers, see Figure 1. Each layer can be customized with multiple *n*-gram sizes, multiple feature representations, multiple feature selection algorithms and learning algorithms. It is recommended that different classifier shall be used while maintaining their neutrality as much as possible to increase the effectiveness [15]. From the lower layer, the instances that are declared as benign are given to upper layer for reclassification. If at any layer, an instance is classified as malware, it is not fed into the next layer. The classification results from all the layers are given to the veto classifier. The malware prediction at any layer for any instance is considered as veto for that particular instance.

2) Parallel Implementation: Instead of using layers, all the possible permutations of classifiers can be implemented

in a distributed or parallel manner. Each learning algorithm is trained over the whole data set for generating the classifiers. Instead of testing only positive instances by some classifiers, each classifier works independent of results from other classifiers. All the votes from the classifiers are collected at a central point where the veto classifier outputs the final prediction.

IV. Pre-processing

The choice of representative features affects the predictive performance of a classifier. Consequently two *n*-gram extraction techniques are proposed. Most of the research studies for malware detection indicate the usage of either hexadecimal-based *n*-gram data set or opcode-based *n*-gram data set for an experiment. The proposed extraction techniques can be used to extract both representations. For this particular study opcode *n*-gram extraction is performed, therefore the proposed techniques are explained in the context of opcode *n*-grams only.

Traditionally *n*-gram extraction is performed by using a specific size window with a fixed step; step size is equal to window size. The fixed size window traverse the data file to extract specific size *n*-grams. The generated output contains adjacent *n*-grams. To explain this process, assume that a disassembled binary file contains the following given data. A pair of characters represents an opcode (or a hexadecimal code). The task is to create *bi*-grams, i.e., *n*-gram of size 2 from this data file.

aa bb cc dd ee ff gg hh ii jj kk ll mm nn oo pp

The generated *bi*-grams from this file are "aabb ccdd eeff gghh iijj kkll mmnn oopp" and so on. The fixed size window is unable to extract some *n-grams* such as "bbcc" or "ddee". If the file size is large and data is redundant then there is a probability to have missing combinations, but still missing *n*-grams can not be produced in the appropriate frequency and can have less importance for classification task.

A. *Overlapping* n-*gram*

To address the problems of missing *n*-grams, the use of configurable sliding window to produce overlapping *n*-grams is proposed. The window can be configured with two parameters, i.e., size and step. The size parameter defines the size of a *n*-gram to be extracted and step parameter defines the number of opcodes to be skipped before extracting the next *n*-gram. It is expected that all possible combinations can be extracted by this extraction technique. Referring the example in section IV, if the window is configured as following, size = 2, i.e., two adjacent opcodes are extracted to create a *n*-gram and step = 1, i.e, after *n*-gram extraction window skips one opcode (first opcode) to move forward. This setting generates "aabb *bbcc* ccdd *ddee* eeff *ffgg* gghh *hhii* iijj *jjkk*" and so on.

B. *Non-adjacent Opcode Extraction*

Either by using traditional *n*-gram extraction or overlapping *n*-gram extraction, extracted *n*-grams can provide the information only about the dependencies of the adjacent opcodes. It is of worth to look at the information provided by non-adjacent opcodes. Non-adjacent opcodes have dependencies such as they can be function header and tail. Some changes are proposed in the overlapping *n*-gram extraction technique to explore the information both from non-adjacent opcode and non-adjacent adjacent opcode. The size parameter is changed to the start-end size parameter. The start-end size parameter defines the number of adjacent opcodes to be extracted for start and end of a *n*-gram. The step size parameter defines the number of opcodes to be skipped for extracting a new *n*-gram. A new parameter is introduced, i.e., gap size which specifies the gap between start and end opcode or number of opcodes to be skipped between start and end opcode of a *n*-gram. The example mentioned in the section IV can be used to explain this procedure. If windows is configured as following for extracting non-adjacent *bi*-grams, start-end size = 1, i.e., one opcode for the start and one opcode for the end of a *n*-gram are extracted, step = 1 and gap = 1, i.e, one opcode between the start opcode and the end opcode of a *n*-gram is skipped. This configuration produces the *bi*-grams which contains non-adjacent opcodes. The generated output is "aacc bbdd ccee ddff eegg" and so on. To have non-adjacent adjacent opcodes in a *n*-gram, the configuration can be changed as follow: start-end = 2, i.e., two adjacent opcodes for the start and the end of a *n*-gram are extracted; step size and gap size are kept 1. The generated output is "aabbddee bbcceeff ccddgghh" and so on. If the value of gap size and step size parameters is changed from 1 to 2, the generated output is "aabbeeff ccddgghh eeffiijj" and so on.

V. Experiment

The aim of the experiment is to evaluate the proposed veto voting based malware detection model and impact of the proposed data pre-processing techniques. The model can be used to detect either a specific family of malware or different types of malware; however in this experiment a specific family of malware is used. The experimental data set contains Windows-based executable files. Windows is a common OS for novice users and contains different vulnerabilities[1] which can be exploited by the malware. When a binary file in the data set is disassembled, different file features such as assembly language instructions and printable strings, are produced in text format which are further processed to extract the assembly directives i.e. opcode. Opcodes are further processed to prepare the *bi*-gram data sets using different strategies. Different text categorization techniques can be applied to the output generated in the

[1]http://technet.microsoft.com/en-us/security/bulletin/

previous step to find discriminating features of benign and malware. Term Frequency-Inverse Document Frequency (tf-idf) is used to extract the significant features from the data sets. The extracted features are used to create Attribute-Relation File Format (ARFF)[2] files. ARFF file is a structured ASCII text file that includes a set of data instances, each described by a set of features [4]. ARFF files are used as input to the proposed model which uses Waikato Environment for Knowledge Analysis (Weka) application programming interface (API) [16] for applying the learning algorithms to build and analyze classifiers. A pre-experiment is performed for the selection of learning algorithms. The main experiment is divided into two sub-experiments. In first experiment, the inductive biases of the different classifiers built on the same data set are combined. Second experiment combines the inductive biases of different classifiers built on different data sets. In both experiments, the results from all the classifiers are given to the veto classifier for final prediction.

A. Feature Representation

Opcode is used for generating *bi*-grams as feature. It is concluded in the previous studies that opcode *n*-grams are better choice for malware detection in comparison to other features such as printable strings, systems calls or byte code (hexadecimal) *n*-grams [7]. The opcode *n*-grams are capable of providing information about program flow, structure and function that can not be deduced from other representations.

B. Data Set Creation

For this particular study, scareware (rouge) software is selected as malware representation. The reason for this choice is, there is a subtle difference between scareware and benign. In case of traditional malware, presence of malicious payload discriminate a malware from the benign. However in scareware, no specific malicious payload is present that can be used to discriminate a scareware from the benign. Absence of malicious payload may deceive human expert for the classification of a particular software as scareware.

Scareware are scam software that usually masquerade as an anti-virus software and resembles the benign software in functionality. Scareware generates false alarm about the presence of malware in the user's machine. The false alarms are used to scare the users into disclosing their credit card information for buying protection[3]. No public data set e.g. virus, Trojan, and worm data sets, is available for the scareware detection experiments. Therefore, a data set with 500 files is created; out of which 250 files are scareware and 250 files are representing benign. The benign files are default applications of Windows OS such as notepad, paint and applications available online for download at download.com. All the benign files are scanned with commercial security software (anti-malware) to minimize the probability of malware presence in a benign file. Scareware files are obtained from the malware database of Lavasoft[4].

C. Pre-Processing

The disassembled file is a regular text file which contains three fields, i.e., the memory address of the instruction, the byte-based location in the file and the instruction itself (combination of opcode and operands). The next step is to extract only opcode from the files and discard irrelevant information, i.e., operands. The extracted opcodes are saved in the original order. After opcode extraction from the disassembled files, three different procedures are used to tokenize the data to produce *bi*-grams for three different data sets. Each row in a data set represents a *bi*-gram, i.e., concatenation of two opcodes. Hereafter these three data sets are referred as *bi*-gram data set, overlap data set and sliding window data set respectively to indicate the procedure used in creating that particular data set. The *bi*-gram size has yielded the best performance in a previous study [17] and possible combinations of opcodes to generate a *bi*-gram are limited, depending upon the number of reserve words in the assembly language.

For the *bi*-gram data set, a fixed size window traverse each input file from top to bottom. In every step, a *n*-gram consisting of two opcodes is extracted and recorded in another file having similar file name but different extension. The purpose of keeping similar name is to keep track of benign files and scareware files, so each file can be represented at same position in all three data sets and finally in the ARFF file. For overlap data set procedure mentioned in the section IV-A is followed with the configuration, i.e, size = 1 and step = 1. For the sliding window data set, start-end size and step parameters are kept one. To obtain the non-adjacent opcode *bi*-grams, each file is processed in four consecutive passes with a gap size ranging from 1-4. Due to the changing gap size, the first generated *bi*-grams are having a gap of one opcode between the start opcode and the end opcode, in the second pass there is a gap of two opcodes and so on. The process of generating sliding window data set is slower than generating the *bi*-gram data set and overlap data set; however the computational cost and memory requirements are lower than creating large size *n*-grams.

D. Feature Selection

Many real world problems are complex. To apply learning algorithms, the dimensionality of complex problems is reduced by choosing a subset of significant features from the given set of (raw) features. The selected subset of features plays significance role in the increase/decrease of either classification and/or computational performance. Significant

[2]http://www.cs.waikato.ac.nz/ml/weka/arff.html

[3]http://news.bbc.co.uk/2/hi/8313678.stm

[4]http://lavasoft.com

feature selection is done by using a feature selection algorithm, removing features that are deemed unlikely to help in the classification process. In the field of text classification tf-idf shows promising results for the valuable features selection. In this experiment tf-idf is applied on data sets to limit the number of features to top 1000 features per data set. The tf-idf is a statistical measure of importance of a *bi*-gram in the whole data set [18]. The tf is the number of times a *bi*-gram occurs in a file, df is the number of files in a class that contain a particular *bi*-gram. The idf of a *bi*-gram is obtained by dividing the total number of files (N) by the df and then taking the logarithm.

E. Pre-Experiment for Algorithm Selection

A number of studies has addressed similar problem with different learning algorithms, however none of the authors is conclusive on the choice of algorithms either for malware detection or according to the produced data set. In a number of studies Ripper (JRip) [19], C4.5 Decision Tree (J48) [20], *k*-nearest neighbor (IB*k*) [21] Naive Bayes [4] and SMO [22] outperformed other algorithms. Based on previous research, a pre-experiment is performed to evaluate all these algorithms on all the three data sets. The top three algorithms, i.e., JRip, J48 and IB*k* are considered as candidates, to combine their inductive biases for final prediction in the proposed model.

F. Performance Evaluation Criteria

Each learning algorithm is evaluated by performing cross-validation tests. Confusion matrices are generated by using the responses from the classifiers. The following four estimates define the elements of a confusion matrix: True Positive (TP) represents the correctly identified scareware programs. False Positive (FP) represents the incorrectly classified benign programs. True Negative (TN) represents the correctly identified benign programs and False Negative (FN) represents the incorrectly identified scareware programs. The performance of each classifier is evaluated using Recall (R) which is the ratio of scareware programs correctly predicted from the total number of scareware programs, Precision (P), ratio of scareware instances correctly identified from the total number of programs identified as scareware. F-Measure (F1) is the harmonic mean of precision and recall and is the last evaluation measure.

VI. Results and Discussion

In the first experiment, one data set is used to build classifiers from three different algorithms. In the second experiment, three data representations are used and one classifier is trained from each representation. Majority voting is compared to the veto voting (see Table II). In first experiment, the predictions from three classifiers are collected and given to the veto classifier and majority voting. The predictions from all the classifiers including both voting strategies on each data set are shown in Table II. In second experiment, three algorithms, i.e., JRip for *n*-gram data set, JRip for overlap data set and J48 for sliding window data set are selected on the basis of the recall in the first experiment. These algorithms are used to built three classifiers and the predictions about each instance from these classifiers is given to the veto classifier and majority voting for the final prediction. The results of this experiment (see Table II) indicate that recall of the veto classifier is better than recall values in the first experiment. Majority voting shows the similar behavior for the precision.

Table II
Experimental Results

Data Set[a]	Algorithm	TP	TN	FP	FN	R[b]	P[b]	F1[b]
Experiment with one data set and three algorithms								
***n*-gram**	JRip	243	184	66	07	0.972	0.786	0.869
	J48	226	225	25	24	0.904	0.900	0.902
	IB*k*	224	225	25	24	0.896	0.899	0.897
	Veto[c]	243	203	47	07	0.972	0.837	0.900
	Majority[c]	223	233	17	26	0.895	0.929	0.912
Overlap	JRip	238	197	53	12	0.952	0.817	0.879
	J48	232	234	16	18	0.928	0.935	0.931
	IB*k*	224	224	26	26	0.896	0.896	0.896
	Veto	246	208	42	04	0.984	0.854	0.914
	Majority	230	240	10	20	0.920	0.958	0.938
S. Window	JRip	209	215	35	41	0.836	0.856	0.846
	J48	215	205	45	35	0.860	0.826	0.843
	IB*k*	164	237	13	86	0.656	0.926	0.768
	Veto	242	139	111	08	0.968	0.685	0.802
	Majority	220	204	46	30	0.880	0.827	0.852
Experiment with three data sets and one algorithm on each data set[d]								
n-gram	JRip	243	184	66	07	0.972	0.786	0.869
Overlap	JRip	238	197	53	12	0.952	0.817	0.879
S. Window	J48	215	205	45	35	0.860	0.826	0.843
	Veto	248	195	55	02	0.992	0.818	0.896
	Majority	223	247	03	26	0.895	0.986	0.938

[a] The full names of data sets are *n*-gram data set, overlap data set and sliding window data set.

[b] R is Recall, P is Precision, and F1 is F-Measure.

[c] Veto is Veto Classifier and Majority is Majority voting.

[d] Veto Classifier and Majority voting both are applied on all the three data sets.

The experimental results indicate that combining the inductive biases of multiple algorithms trained on different representations predicts better for the malware detection than combining the inductive biases of multiple algorithms trained on the same data set. The experimental results also show that veto classifier has better recall than majority voting, i.e., veto classifier reduces the number of misclassified scareware. Recall is the main measure as the objective of the veto approach is to reduce the probability of malware misclassification while tolerating a percentage of false positives or decrease in precision. If the system is tuned to predict all the applications as malware, it will give high rate of false positives which is unacceptable from user's point of view. Users require correct malware prediction as well as correct prediction for the benign application at the same time. Therefore precision is also considered as complimentary measure with the recall. The veto classifier shows a higher

tendency for the correct detection of scareware while the majority voting shows an inclination towards the detection of benign applications. Therefore the precision rate is higher for the majority voting. There are few instances which are misclassified by both voting schemes. Most of these instances are benign but declared as scareware by both veto classifier and majority voting. However the number of such instances is ignorable. The final evaluation measure is F-measure; one can argue that arithmetic mean can also be used. Arithmetic mean is an unsuitable measure as with 100 % R or P, arithmetic mean is always 50 %. This is not the case with harmonic mean as harmonic mean is always less than or equal to the arithmetic mean. If there is a difference between value of R and P such that value of R is significantly smaller than P, the harmonic mean tends strongly towards the recall.

Bi-grams are used as feature in the experiment because they are computationally inexpensive to produce. Generally such short combinations may not indicate an important function or set of instructions in the files and are difficult to analyze. However, *bi*-grams in sliding window data set can provide important information for the scareware analysis due to combination of non-adjacent opcodes. Scareware resembles the benign applications such as displaying popup windows or alert messages, and showing the dialog boxes. Therefore it is difficult for human experts to predict about the scareware by analyzing the functionality of an application only. The propose model helps human expert by automating the process of analyzing and predicting the scareware (malware). JRip and J48 algorithm are considered expensive algorithm in terms of time consumed to train and generate the model. However it is easy to analyze the rules and trees generated to discriminate the scareware and benign.

The decisions of the different classifiers are combined to produce better results and such combination shall not be considered as a substitute of a good classifier [23]. The proposed veto classifier follows the same principle. Veto classifier is neither a substitute of a good classifier nor replacing the majority voting. In the domain of decision theory, it has been suggested that different voting strategies shall be adopted for different tasks according to the problem in question. We argue that veto classifier is a better option for malware detection task as this approach addresses the problems of majority voting. There are different problems related with the majority voting such as majority voting may ignore the right decision of the minority votes. While ignoring the decision from minority votes, the total number of majority votes may have an ignorable difference in comparison with the total number of minority votes. Another problem of the majority voting is the choice of number of candidate classifiers. If the number of selected classifiers is an odd then a simple majority can be obtained but if the number of selected classifiers is an even, then a situation may arise where equal numbers of votes are given to both the benign and malware classes. In the domain of ML, different variations of majority voting has been suggested such as restricted majority voting, enhanced majority voting, and ranked majority voting to address the problems of majority voting [24]; such problems are avoided with the proposed veto classifier.

The results of veto classifier depend upon a suitable permutation of the algorithms. Some permutations may achieve 100 % recall by just predicting all the applications as malware. Some permutation can achieve 100 % precision, if all the instances are predicted as benign applications. Before permutation, classifiers selection is a critical and complex task. For a small number of classifiers an optimal combination can be found exhaustively but as the number of classifiers increases, the complexity of selection is increased due to their different inductive biases, search limits and practical applicability. The classifier selection process can be improved by a static selection or dynamic selection method [25]. One possible direction, for improving the decision strategy of veto classifier is dividing the voting process into two stages and allocate weights to the vote of each classifier on the basis of predetermined criteria. All the classifiers vote for the decision. After voting, one of the two alternatives can be used. First alternative is, weights can be readjusted according to the performance of each classifier and two top weighted classifiers (if they have different predictions) are compared to obtain the final decision. If the highest weighted classifier predicts an instance as malware, instance is considered as malware. The other alternative can be to sum all the weights. If the total weight of votes predicting the class as malware is higher than a specific (predetermined) threshold, final class of instance is predicted as malware.

VII. Conclusion and Future Work

A veto-based classification was proposed that was able to predict about malware better than majority voting. A series of experiments with *n*-gram data sets, generated from different strategies, were performed. A recent threat, i.e., scareware was used as malware representation. The results indicated that the proposed model reduced the number of false negatives (malware detected as legitimate application). For the future work the aim is to improve the proposed model in three different directions, i.e., improvement in the selection of classifiers for the optimal results, parameter tuning of the selected classifiers and improve the decision strategy. The proposed model will also be tested for detection of different types of malware.

References

[1] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. Mitchell, "A layered architecture for detecting malicious behaviors," in *Recent Advances in Intrusion Detection*, 2008, pp. 78–97.

[2] W. Stallings, *Network Security Essentials: Applications and Standards*, 4th ed. Prentice Hall, 2011.

[3] T. M. Mitchell, *Machine Learning*, 1st ed. McGraw-Hill, Inc., 1997.

[4] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Morgan Kaufmann Inc., 2011.

[5] M. Schultz, E. Eskin, F. Zadok, and S. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings of the Symposium on Security and Privacy*, 2001, pp. 38–49.

[6] R. K. Shahzad, S. I. Haider, and N. Lavesson, "Detection of spyware by mining executable files," in *Proceedings of the 5th International Conference on Availability, Reliability, and Security*. IEEE Computer Society, 2010, pp. 295–302.

[7] R. K. Shahzad and N. Lavesson, "Detecting scareware by mining variable length instruction sequences," in *Proceedings of the 10th Annual Information Security South Africa Conference*, 2011, pp. 1–8.

[8] A. Gepperth, "Object detection and feature base learning with sparse convolutional neural networks." in *Proceedings of the 2nd international conference on Artificial Neural Networks in Pattern Recognition*, 2006, pp. 221–232.

[9] D. Tikk, Z. T. Kardkovács, and F. P. Szidarovszky, "Voting with a parameterized veto strategy: Solving the KDD cup 2006 problem by means of a classifier committee," *ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) Explorations Newsletter*, vol. 8, no. 2, pp. 53–62, 2006.

[10] R. Kern, C. Seifert, M. Zechner, and M. Granitzer, "Vote/veto meta-classifier for authorship identification - notebook for pan at clef 2011," in *Proceedings of the Conference on Multilingual and Multimodal Information Access Evaluation*, 2011.

[11] S. Weiss, C. Apte, F. Damerau, D. Johnson, F. Oles, T. Goetz, and T. Hampp, "Maximizing text-mining performance," *IEEE Intelligent Systems and their Applications*, vol. 14, no. 4, pp. 63–69, 1999.

[12] G. Giacinto and F. Roli, "Methods for dynamic classifier selection," in *Proceedings of International Conference on Image Analysis and Processing*, 1999, pp. 659–664.

[13] A. Santana, R. G. F. Soares, A. M. P. Canuto, and M. C. P. d. Souto, "A dynamic classifier selection method to build ensembles using accuracy and diversity," in *9th Brazilian Symposium on Neural Networks*, 2006, pp. 36–41.

[14] H. Moulin, "Voting with proportional veto power," *Econometrica*, vol. 50, no. 1, pp. 145–62, 1982.

[15] L. I. Kuncheva, "Diversity in multiple classifier systems," *Information Fusion*, vol. 6, no. 1, pp. 3–4, 2005.

[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) Explorations Newsletter*, vol. 11, pp. 10–18, 2009.

[17] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici, "Unknown malcode detection using OPCODE representation," in *Proceedings of the 1st European Conference on Intelligence and Security Informatics*, 2008, pp. 204–215.

[18] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," in *Information Processing and Managment*, 1988, pp. 513–523.

[19] W. Cohen, "Fast effective rule induction," in *Proceedings of 12th International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 1995, pp. 115–23.

[20] J. R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[21] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[22] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," *Microsoft Research Technical Report MST-TR-98-14*, 1998.

[23] J. Franke, L. Lam, R. Legault, C. Nadal, and C. Y. Suen, "Experiments with the cenparmi data base combining different classification approaches," in *Proceedings of 3rd International Workshop Frontiers Handwriting Recognition*, 1993, pp. 305–311.

[24] A. Rahman, H. Alam, and M. Fairhurst, "Multiple classifier combination for character recognition: Revisiting the majority voting system and its variations," in *Proceedings of the 5th International Workshop on Document Analysis Systems*, 2002, pp. 167–178.

[25] D. Ruta and B. Gabrys, "Classifier selection for majority voting," *Information Fusion*, vol. 6, no. 1, pp. 63–81, 2005.