Using Multi-Feature and Classifier Ensembles to Improve Malware Detection

Yi-Bin Lu[1], Shu-Chang Din[2*], Chao-Fu Zheng[1], and Bai-Jian Gao[1]

[1]Department of Computer Science, Chung Cheng Institute of Technology, National Defense University
[2]Graduate School of National Defense Science, Chung Cheng Institute of Technology, National Defense University

ABSTRACT

With the rapid growth of internet application, malware has become one of the major threats to information security. Traditionally, anti-virus products use signature matching to detect malware, but the drawback is that they can not detect new and unknown malware. Recent studies showed that the use of machine learning can successfully detect new and unknown malware, but the limitation of this technique is its high false rate. The performance of machine learning is influenced by two main factors: (1) the features used to represent the instances; and (2) the algorithm used to generate classifier. In this paper, we improved the accuracy of machine learning from these two factors. On the one hand we combined features extracted from both content-based and behavior-based analyses to represent the instances; on the other hand, we used classifier ensembles to replace individual classifier. Based on our methodology, a hybrid-classifier was implemented to classify unknown executables as either malicious or benign. Experimental results show that the methods proposed in this paper can improve the accuracy of malware detection effectively.

Keywords: malware, machine learning, features, classifier ensembles

運用多重特徵及集成分類法以改善惡意程式的偵測

陸儀斌[1]　丁淑章[2*]　鄭朝福[1]　高百健[1]

[1]國防大學理工學院資訊科學系(所)
[2]國防大學理工學院國防科學研究所

摘　　要

網際網路的蓬勃發展使得惡意程式對資訊安全造成嚴重的威脅。傳統上是以病毒碼比對的方式偵測惡意程式，其缺點是無法偵測出新型及未知的惡意程式。近年來許多研究顯示運用機器學習可有效偵測出未知的惡意程式，其缺點是錯誤率過高。本文針對特徵及演算法這兩項影響機器學習準確性的關鍵因素著手改進。在特徵方面，結合內容及行為特徵來表示樣本；在演算法方面，運用集成分類法取代單一分類法。基於上述改進方法，本文提出一個混合型分類法用來區分未知程式屬於惡意或正常類別。實驗結果顯示，本文所提出的方法可有效提高惡意程式偵測的準確性。

關鍵字：惡意程式，機器學習，特徵，集成分類法

文稿收件日期 99.1.24; 文稿修正後接受日期 99.9.1; *通訊作者
Manuscript received January 24, 2010; revised September 1, 2010; *Corresponding author

I. INTRODUCTION

Traditionally, anti-virus products use signature matching to detect malware. They create a unique pattern tag for each malware so that its future executables can be correctly classified. The drawback is that they can not detect new and unknown malware. Recent studies [1-7] showed that the use of machine learning can successfully detect new and unknown malware, but the limitation of this technique is its high false rate. Therefore, how to improve the accuracy of machine learning becomes the critical issue in using machine learning for malware detection in practical applications. The performance of machine learning is influenced by two main factors: (1) the features used to represent the instances; and (2) the algorithm used to generate classifier. In this paper, we plan to improve the accuracy of machine learning from these two factors.

First, for the features used to represent the instances, Schultz et al. [1] proposed three kinds of features: Dynamic Link Library (DLL) / Application Programming Interface (API) function calls used, strings, and byte sequences (i.e. Binary N-grams). For each executable, the authors constructed binary feature vectors based on the presence or absence of each to represent the executable. Subsequent researchers [8-12] chose one of them as feature of executables in using machine learning to detect malware. Recently, Masud et al. [13, 14] proposed a combination of three different kinds of features to achieve high accuracy in detecting malicious executables. They are binary N-grams, assembly instruction sequences, and DLL/API function calls. Besides choosing the features proposed by Schultz et al. [1], some researchers also tried to modify Schultz's features for malware detection. In [15], Reddy et al. proposed using variable length N-grams to replace fixed length N-grams. They believed that fixed length N-grams cannot capture meaningful sequences of different lengths. In [16], Ye et al. proposed "interpretable strings" to replace simple printable strings. Ye et al. suggested that printable string, like "3d%3dtgyhjij", should not be extracted as features, since it can't be interpreted. In [17], Shafig et al. extracted 189 features based on the structural information of PE files. Besides the 73 DLLs referred to in executables, they extracted 116 statically computable features such as number of sections, size of the stack and the heap, etc. The aforementioned features are all classified as content-based features. Content-based features primarily target at the structural information of an executable. The advantage is that such information can be extracted from the content of an executable without really executing it. But the shortcoming is that they often fail to reveal system interaction information. For example, using 'KERNEL32.dll /CopyFile' DLL/API to copy an .exe file to a system directory is likely to be malicious, while copy a .txt file to user's directory is thought to be benign.

In order to resolve the shortcoming of content-based features, recent studies have proposed using behavior-based features for detecting malware. Liu [18] listed 35 host behaviors (such as create files, set auto run etc.) to detect malicious executables. Liu et al. [19] selected 8 behaviors shared by many malware (such as registry modification etc.) as behavioral features. Hu et al. [6] proposed a malicious detecting method using 35 run-time behavioral features, such as hiding windows, hooking keyboard etc. The idea is to provide an executable with the environment where it can be executed realistically, so that the executable can then be classified as either malicious or not according to what it does. The limitation of using behavior-based features for malware detection is that the decision model can work well only if all behaviors can be clearly observed. It may not observe random or environment specific behaviors. Moreover, modern malware will often try and evade detection by hiding its malicious behaviors.

In this paper, we plan to combine content-based and behavior-based features for the purpose of complementary results. As most executables nowadays target at Microsoft Windows platform and exploit the DLL/API to access system resources such as files, the registry, processes, and networking information etc. We chose DLL/API function calls as content-based feature of an executable. The reason behind the use of DLL/API is that the presence or absence of an informative API is related to the presence or absence of a certain action in the code of an executable. Another

reason is that the domain of DLL/API is limited compared with binary N-grams and strings. For example, there are only thousands of DLL/APIs, but the domain of N-grams is 2^32 (4,294,927,296) for N=4. Such a huge domain of features will produce problems in memory storage and execution time. For behavior-based features, 12 primary behaviors commonly seen in malicious executables were chosen through group discussion. Section III provides detailed descriptions of these behaviors.

Secondly, for the algorithm used to generate classifier, several machine learning algorithms have been investigated for the malware detection. They are Support Vector Machine (SVM) [4,20,21,22], Decision Tree (DT) [2,5], Artificial Neural Networks (ANN) [24-25], Naïve Bayes (NB) [1,5], and k Nearest Neighbors (kNNs) [3,23] etc. Experimental results of above literature show that the classifiers generated by these popular algorithms perform passably. To improve the performance, the solution is to combine different classifiers just like the old saying: two heads is better than one. This is particularly true when the heads are diverse. Combing the predictions of different classifiers to achieve a better performance than any of the base classifiers involved in the combination is an approach used in various fields. Recent studies have proved that individual base learning methods can be replaced by ensemble learning methods to improve the accuracy of malware detection. Kolter [3] tested an ensemble learning method called Boosted to boost SVM, Naïve Bayes and decision tree, the experimental results showed that Boosted decision tree outperforms individual base learning methods. Menahem et al. [26] proposed a multi-inducer ensemble method that combines five different base learning methods (Decision Tree, Naïve Bayes, kNNs, VFI and OneR) to improve malware detection. Besides the aforementioned ensemble learning methods, popular ensemble learning methods such as bagging, voting, stacking, and grading all have been successfully used in several applications [27-30]. In this paper, the performance of these popular methods was compared using dataset. Meanwhile, we propose a new ensemble learning method called SVM-AR, it combines SVM and association rules [31] based on hierarchical taxonomies. The motivation for using the hierarchical taxonomies is to classify unknown executables in two steps for improving the accuracy of malware detection. In the first step, the executables were roughly divided into malicious and benign group by a hyper-plane determined by SVM algorithm. After that the false predictions were filtered out by applying association rules induced from the behavioral patterns of executables. One can think of the hyper-plane as a global optimal cut, while the association rules are local optimal filters. Experimental results show that SVM-AR outperforms all popular ensemble learning methods.

The rest of this paper is organized as follows: Section II briefly describes a number of popular classification algorithms examined in this study. Section III describes our machine learning method for improving malware detection. The experimental results are presented and discussed in section IV. Section V is the conclusions.

II. RELATED CLASSIFICATION ALGORITHMS

In this section, a number of classification algorithms examined in this research are briefly described as follows:

2.1 Individual Learning Algorithms

NB: Naïve Bayes [1] is a probabilistic method. Given an unknown testing instance, it uses Eq.(1) to compute posterior probability of each class and then the class with the highest value is its prediction.

$$C = \arg\max_{C_i} P(C_i)\prod_j P(F_j \mid C_i) \qquad (1)$$

Where $P(C_i)$ is the probability of class i, $P(F_j|C_i)$ is the conditional probability of each feature value given the class i.

SVM: SVM [32] is based on the structural risk minimization principle from the statistical learning theory. It is particularly suitable for solving binary classification problems. SVM can identify an optimal separating hyper-plane between two classes in a high dimension feature space. The optimal hyper-plane is subject to the constraint as:

$$min\ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i \qquad (2)$$
$$s.t.\ y_i(w\cdot x_i + b) \geq 1-\xi_i \quad ,\xi_i \geq 0, \quad i=1,...,l$$

Where w is normal to the hyper-plane, ξ_i is error for the i-th instance. C is a parameter to represent the tradeoff between maximizing the margin and minimizing the training error. The concept of SVM is shown in Fig.1.

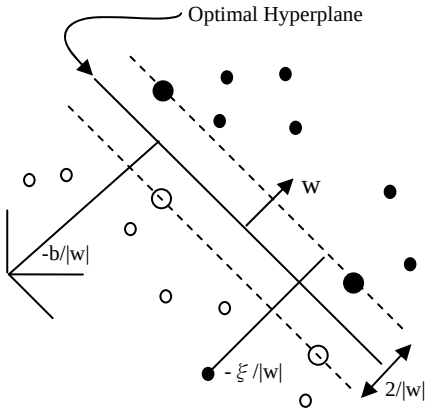

Fig.1. SVM optimal hyperplane.

kNNs: kNNs [33] is the simplest learning method. To classify an unknown testing instance, it finds the k-nearest neighbors from training instances and then returns the plurality vote of their class labels of these k-nearest neighbors as its prediction.

DT: A decision tree [34] is a rooted tree with internal and leaf nodes. The internal nodes correspond to features and leaf nodes correspond to class labels. It builds a tree by selecting the feature that best divides the training instances into proper classes in learning. For testing, it uses the features and their values of an instance to traverse the tree from the root to a leaf, and then outputs the class label of the leaf as its prediction.

OneR: OneR [35] is a very simple classification model; it uses the minimum-error feature (attribute) for prediction.

2.2 Ensemble Learning Algorithms

In this paper we examined several well-known ensemble algorithms used in various fields. They are briefly described as follows:

Bagging: The bagging method [27] produces a number of imaginary training sets first. Each imaginary training set is generated by randomly drawing N instances, where N is the size of the original training set. It is possible that some instances are drawn multiple times while other are left out. After that, each homogeneous individual classifier is generated with a different imaginary training set. The weight of each individual classifier is equal, so the final prediction is determined according to plurality vote.

Boosting: The boosting method [27] produces a set of weighted homogeneous classifiers by iteratively learning a classifier from a weighted dataset, evaluating it, and then reweighting each instance in the dataset based on whether the prediction for the instance is correct or not. In every iteratively learning, it pays more attention to those instances with false predictions in the last iteration. For testing, it uses the set of homogeneous classifiers and their weights to predict the class with the highest weight.

Voting: It is the most intuitional method to combine heterogonous classifiers. Each base-level classifier casts a vote for its prediction, and then combines classifiers according to the plurality vote.

Stacking: The Stacking method [28] uses two levels to combine heterogonous classifiers. In level-0, each heterogonous base-classifier predicts the probabilities for every class. And then it combines the different predictions of each base-classifier and class value into one vector for every training instance to form the meta-training set. This meta-training set is used to generate a meta-classifier in level-1. For testing, the base classifiers are first conducted for the probabilities of every class on the test instance. These form a meta-sample for the meta-classifier which predicts the final class.

Grading: The Grading method [29] learns a meta-level classifier for each heterogonous base-level classifier. The meta-level classifier is used to predict whether the base-level classifier is to be trusted. Only the base-level classifiers that are predicted to be trustworthy are selected and then combined with their predictions to make final decision by the simple plurality vote.

III. METHODS

The framework of malware detection by

using machine learning is shown in Fig.2. In general, the working process can be split into two phases: training and testing. The goal of training phase is to generate classifiers for detection, and then evaluate the classifiers during testing phase. Detailed procedures for each functional block in Fig.2 are described in the following sections.

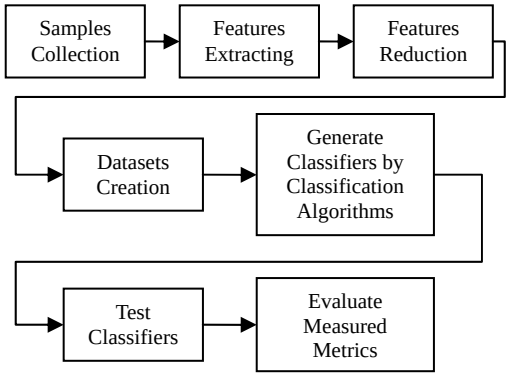

Fig.2. Malware detection by machine learning.

3.1 Samples Collection

A few years ago, the purpose of malware was to cause major damage. The malware writer wanted to cause the most damages and gather the most media attention. In the last few years, the motivation of malware writers has shifted to profit-making [36]. As a result, malware is becoming more insidious. It will carry out its malicious activities without the user's knowledge. In this paper, the malicious samples were collected according to current trends in malware. The focus is on those malware with insidious activities (such as keyloggers, password thieves, network sniffers, backdoors, stealth Trojans, spyware and rookits etc.), and not the ones with destructive behaviors (such as virus).

Unlike intrusion detection systems (IDS) that all have a common database set (DARPA datasets [37]), no standard data sets are available to researches of malware detection. As a result, the researchers had to collect benign and malicious samples to establish the datasets manually. The benign samples were gathered from the Web site Softking (http://www.softking.com.tw). Those samples consisted of different categories of programs, such as graphic software, system tool, multimedia software, office tool and internet application etc. The malware samples from different sources were then obtained. They included the website AvpClub (http://www.avpclub.ddns.info/discuz /index.php), attachments of infected e-mails, and laboratories of related anti-virus organizations. A total of 1,200 samples were collected, including 400 malware and 800 benign programs. After running through five popular anti-virus software products (Kaspersky, Symantec, Trend Micro, Spyware Detector and F-secure online) for verification, each sample was labeled either as malicious or benign.

3.2 Features Extracting

In our study, the content-based and behavior-based features were proposed and combined in order to take advantage of each feature. They are described below:

3.2.1 Content-based features

In comparison with N-grams and PE features, using strings as features offers the worst accuracy [17], although Ye et al. [16] proposed that using "interpretable string" to replace simple printable strings can improve accuracy. In [12], Lai et al. used four methods (frequency, glossary filter, support threshold, and frequency item sets) to select 100 most important strings as features for malicious detection. Their experimental results showed that the glossary method is slightly better than frequency method, but inferior to support threshold and frequency item sets. Moreover, no matter which method is used for selecting important strings, most of the selected strings are interpretable strings. So the system effectiveness of Ye et al. is possibly due to their use of Support Vector Machine ensemble bagging as classification algorithm, and not because of the selected interpretable strings as features. In [17], N-grams is better than strings, but N-grams approach incurs huge space problem and large overhead in pre-processing for features selection. In this paper, PE information was chosen as content-based features. Unlike Shafig et al. [17] who selected DLLs and computable fields from PE format as features of an executable, DLLs and API function calls were chosen in this study as

features. In addition to DLLs, API function calls were also identified because the researchers believed that the functions of a specific DLL are too wide to capture the exact intend and goal of an executable. The API is a function call for implementing a specific function. It is better than DLL in capturing the malicious purposes of an executable.

The Windows DLL/API function calls were identified as content-based features of an executable in this study. The reason behind the use of DLL/API is that the presence or absence of an informative API is related to the presence or absence of a certain action in the code of an executable. To extract such information from the import section of the file's PE format, a program called GetFileAPIs was developed to parse the DLLs/APIs function calls used by an executable. The results were compared to a popular freeware called PEView [38] to ascertain the features extracted by this program are correct. The purpose of developing this program as a replacement of the tool PEView is to save processing time. This program can analyze all executables by batch mode, while PEView can only analyze executable one by one. After all collected executables were analyzed, 2682 distinct DLL/API function calls were identified. A feature vector was then generated for each sample. The feature vector comprised of 2682 Boolean values to represent whether or not an executable used a specific DLL/API.

3.2.2 Behavior-based features

For behavior-based features, 12 primary behaviors commonly seen in malicious executables from the knowledge of relevant experts were identified through group discussion. To dynamically capture the behavior features of an executable, VMWare [39], a virtual machine installed with Microsoft Windows XP operation system, was used as the test environment to run the executable. The advantage of using VMWare is that a clean snapshot of VMWare can be easily restored every time it needs to run a new executable. Each sample was executed in VMWare and investigated by some suitable sharewares to determine whether or not there is a specific existing malicious behavior. The behaviors and tools used to observe these behaviors are listed in Table 1.

After all the samples were executed and traced, we found that some of them could not be executed properly. This may be due to incompatibility between different versions of operating system. In the end, a total of 980 samples (benign: 696, malicious: 284) was executed. Each sample was represented by a 12-Dimension feature vector.

Table 1. Behaviors and tools used to observe them

ID	Behavior Description	Tools
F1	Packed	PEiD [40]
F2	Auto-Start	AutoRuns [41]
F3	DLL Injection	ProcessExplorer [43]
F4	Modify system files	FileMon [43]
F5	Create execution file(s)	FileMon
F6	Hiding files	RootkitRevealer [44] IceSword [45]
F7	Hiding registry entries	RootkitRevealer IceSword
F8	Hiding processes	ProcessExplorer IceSword
F9	Hiding services	IceSword
F10	Open port	FPort [46]
F11	Reinstallation after removal	ProcessExplorer
F12	Hook	IceSword

3.3 Features Reduction

Since the number of extracted DLL/API features is fairly large, it is impractical to use all of them for training because most of them belong to noisy, redundant or irrelevant data. We need to choose a small, relevant and useful feature set from the entire large set. Many criteria have been proposed [3,13,14,21,23] to select the best features. In our work, we choose information gain as the selection criterion. The information gain of a feature F on a dataset S is given by Eq.(3):

$$IG(S,F) = Entropy\ (S) - \sum_{v \in Val(F)} \frac{|S_v|}{|S|} Entropy\ (S_v) \quad (3)$$

Where $Val(F)$ is the set of all possible values for feature F, and S_v is the subset of S for which feature F has value v. In our case, each feature has only two possible values, either 0 or 1, representing the feature existed in the executable or not. $Entropy(S)$ is computed using Eq.(4):

$$Entropy\ (S) = -\sum_{i=1}^{k} P_i \times Log_2 P_i \quad (4)$$

Where P_i is the proportion of instances belonging to class i in S, k is the total number of classes in S.

In our pilot studies, we ranked each DLL/API by its information gain, and then selected the top 100, 200, 500,…, 2682 DLLs/APIs to evaluate the performance of Naïve Bayes, Decision Tree, kNNs and SVMs. The results are shown in Fig.3.

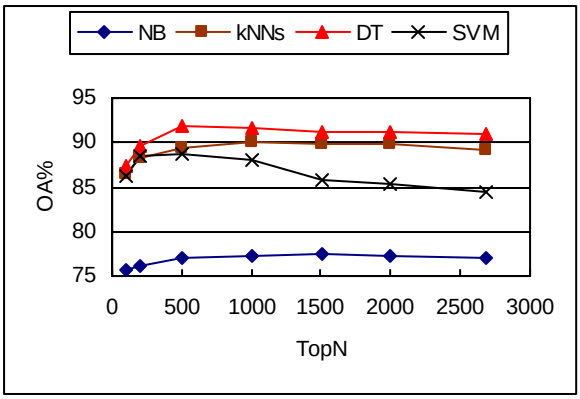

Fig.3. Accuracy vs. size of DLLs/APIs.

It can be seen that the best results were produced by selecting top 500 DLLs/APIs. A curve based on the information gain and rank of each DLL/API is shown in Fig.4.

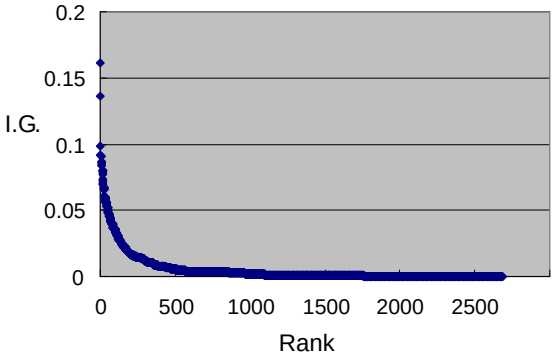

Fig.4. Information gain and rank of each DLL/API.

It also shows that only top 500 DLLs/APIs were needed to generate a significant information gain. Based on the above perspectives, the top 500 DLLs/APIs were selected as content-based features of each sample. It was unnecessary to reduce behavior-based features, as the total number of such features was small.

3.4 Datasets Creation

To avoid the population number affecting the experimental results, four datasets were constructed (A, B, C, D) for the experiments. The datasets differed in the number of samples and malware percentage. Each dataset has three versions. The first version contains both content-based and behavior-based features, which means that each sample is represented to a 512-Dimension feature vector. The second and third versions contain only content-based or behavior-based features. Table 2 shows the properties of the datasets used in the experiments.

Table 2. Datasets in our experiments

Dataset		Features	#Samples (#Ben/#Mal)	Mal%
A	A1	500 DLLs/APIs + 12 behaviors	980 (696 / 284)	30%
	A2	500 DLLs/APIs		
	A3	12 behaviors		
B	B1	500 DLLs/APIs + 12 behaviors	819 (696 / 123)	15%
	B2	500 DLLs/APIs		
	B3	12 behaviors		
C	C1	500 DLLs/APIs + 12 behaviors	568 (284 / 284)	50%
	C2	500 DLLs/APIs		
	C3	12 behaviors		
D	D1	500 DLLs/APIs + 12 behaviors	405 (121 / 284)	70%
	D2	500 DLLs/APIs		
	D3	12 behaviors		

3.5 SVM-AR

A new method, SVM-AR, was proposed in this paper. It combines SVM and association rules based on hierarchical taxonomies. The reason that SVM was selected as fundamental classifier of SVM-AR is explained below: According to the review of the related papers on the topic of malware detection using machine learning, it was found that Support Vector Machine, Decision Tree, Naïve Bayes, and k Nearest Neighbors are the most common classification algorithms used by researchers. This study then tested the overall accuracy of each algorithm in malware detection using the collected datasets. The results showed that NB is the worst classification algorithm. As a result, NB was deleted from the shortlist for its poor accuracy. Empirical studies showed that the accuracy improvement achieved using multi-classifiers is related to the diversity of classifiers. The diversity between DT and association rules didn't meet this basic requirement, so DT was also deleted from the

shortlist. In the end, SVM was selected because it is superior to kNNs in algorithm efficiency, and it is most frequently used in related papers. Table 3 is a summary of the above descriptions. Apart from the reasons mentioned above, SVM is effective in resolving binary classification problems just like the one in this paper.

Table 3. A comparison sheet of popular classification algorithms

Algorithms / Criteria	SVM	DT	kNNs	NB
Classification accuracy	Good	Good	Good	Bad
Diversity to association rules	High	Low	High	High
Frequency used in the related researches	High	Medium ~ High	Medium	Low
Efficiency of algorithm	Good	Best	Bad	Good

After the description of SVM method in the preceding section, The Association Rules and how to combine these two methods to improve the accuracy for classification will be described in this section.

Let $I = \{I_1, I_2, \ldots, I_n\}$ be a set of items. The form of association rule is $X \rightarrow Y$ $[s, c]$. An association rule is an implication $X => Y$, where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \Phi$. s is the support rate of the rule, it represents the percent of the records in the dataset contain $X \cup Y$. c is the confidence of the rule, it represents the percent of records in dataset which contain X also contain Y, i.e.

$$s = \frac{Count\ (X \cup Y)}{|DB|} \tag{5}$$

$$c = \frac{Count\ (X \cup Y)}{Count\ (X)} \tag{6}$$

Where $Count(X \cup Y)$ returns the records in the dataset where $X \cup Y$ holds. $|DB|$ is the number of records in the dataset.

In this paper, we let X be a situation of behavior features, Y be a label of class. A minimum support and minimum confidence was given as thresholds to ensure that only those rules that appear with a high enough support and confidence, as specified, are retained. The rules generated look like this:

(F2 = 1, F5 = 1, F11 = 1) → Class = Malicious [5%, 100%]

(F1 = 0, F2 = 0, F6 = 0, F7 = 0, F10 = 0) → Class = Benign [1.6%, 100%]

By referring to the ID of each behavior feature in Table 1, one can determine the meaning of the malicious rule. It means that if an unknown executable has the following behaviors (auto-start, create execution file(s), and reinstallation after removal), then it is malicious. The support and confidence of this rule are 5% and 100% respectively. The meaning of the benign rule is explained as follows: An unknown executable is un-packed, without auto-run setting, without hiding files and registries, and without opening any port, it is benign. The support and confidence of this rule are 1.6% and 100% respectively.

Using Apriori [47] algorithm, one can obtain all the association rules with certain support and confidence values. The generated rules can be divided into two groups: Class=Benign for benign association rules, Class=Malicious for malicious association rules.

Fig.5 shows how to integrate SVM and Association Rules into SVM-AR. Empirical studies show that the accuracy improvement achieved using multi-classifiers is related to the diversity of classifiers. SVM and Association Rules work in different manners and exploit different sets of features. By integrating them, the possibility of achieving better performance will be high. According to the hierarchical taxonomies in Fig.5, the concept of SVM-AR begins with the use of SVM to roughly divide executables into two areas, namely malicious and benign, by a global optimal hyper-plane. The false rate of SVM resulted from the false classification causes some of the executables to be in the wrong area. One can reduce this false rate by filtering out these false predictions with local optimal method. Here, SVM-AR uses the related association rules to filter out the suspicious executables. It can filter out false negative executables by using malicious association rules, and false positive executables by using benign association rules. Those association rules are induced from common behavior patterns of malicious/benign executables.

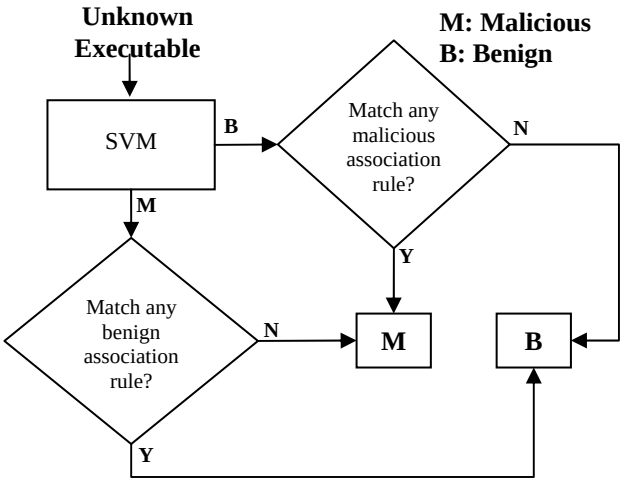

Fig.5. The architecture of SVM-AR.

There are two questions needed to be answered: First, can one guarantee that SVM-AR will be better than other algorithms? The answer is that the basic assumption of SVM-AR is not to reveal any executables that meet the benign rules within malicious boundary determined by Support Vector Machine. In the same way, the executables within benign boundary should not meet any malicious rules. The best case is that one can filter out all misclassified samples by some association rules. The general case may be that some misclassified samples can not be filtered out as they may not fulfill any of the association rules. However, the overall accuracy will still be improved because certain false positive and false negative samples have been filtered out by the association rules. Second, is it possible that some correct classified samples are also filtered out by the association rules; hence the overall accuracy will be reduced and not improved? The answer is that this can only happen in two situations. The first situation is where a benign executable is classified correctly by Support Vector Machine but fulfill a specific malicious rule. This situation should not happen as a benign executable usually does not have a strong pattern of malicious behaviors. Another situation is where a malicious executable is classified correctly by Support Vector Machine but fulfills a benign rule. This situation is likely to take place because the malicious behaviors may go by undetected under our eyes (this is the instinctive drawback for behavior analysis). However, the probability should be very low, as some malicious behaviors may slip through, but to fail in observing all malicious behaviors of a malicious executable will be near impossible. As long as one can observe one or two malicious behaviors of an unknown executable, this executable will not meet any benign rule. We believe that SVM-AR should improve classification accuracy based on the above analysis. To prove this inference, additional experiments were conducted in this study and they yielded the expected results as shown in section IV.

3.6 Measured Metrics

For evaluation purpose, we used the Overall Accuracy (*OA*), Detection Rate (*DR*), and False Positive Rate (*FPR*) as measured metrics. *OA* is the percentage of true classifications over all test samples. *DR* is the percentage of total malicious programs which were labeled as malicious. *FPR* is the percentage of benign programs which were labeled as malicious. The equations are shown as follows:

$$OA = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \quad (7)$$

$$DR = \frac{TP}{TP + FN} \times 100\% \quad (8)$$

$$FPR = \frac{FP}{FP + TN} \times 100\% \quad (9)$$

Where,

TP (True Positives): the number of malicious samples classified as malicious.

FP (False Positives): the number of benign samples classified as malicious.

FN (False Negatives): the number of malicious samples classified as benign.

TN (True Negatives): the number of benign samples classified as benign.

IV. RESULTS AND DISCUSSION

In this section we begin by describing the experimental environment. Weka [48] and LIBSVM [49] were used in this study to perform experiments with different learning algorithms. All experiments were executed on a PC that runs on Windows XP operating system (SP2), with a Intel Pentium4 CPU and 512M of RAM.

In order to evaluate the generalized accuracy, the 10-fold cross-validation procedure was repeated 3 times for all experiments. The obtained experimental results are presented and discussed in the following sections.

4.1 Features Effect

Table 4 shows the overall accuracy for applying each individual algorithm on four datasets.

Table 4. Overall accuracy for using different features

Dataset		Naïve Bayes	SVM	kNNS	DT	OneR
A	A1	74.56%	92.88%	93.49%	92.63%	88.64%
	A2	73.63%	87.99%	90.97%	91.53%	86.94%
	A3	87.59%	90.72%	90.23%	91.53%	88.64%
B	B1	77.55%	93.67%	92.14%	92.07%	83.37%
	B2	76.97%	88.64%	89.49%	91.80%	76.94%
	B3	85.10%	86.60%	85.78%	87.04%	83.37%
C	C1	78.99%	88.61%	89.55%	88.32%	75.35%
	C2	78.52%	83.10%	85.03%	85.68%	75.35%
	C3	77.82%	81.21%	79.57%	82.04%	72.71%
D	D1	81.59%	85.60%	91.93%	89.63%	73.44%
	D2	79.52%	84.20%	83.70%	86.25%	73.44%
	D3	71.44%	75.96%	77.96%	78.02%	69.31%

It can be seen that the combination of both content-based and behavior-based features can improve the accuracy for SVM, KNNs, and Decision Tree in each dataset, but not work well for Naïve Bayes and OneR. OneR uses the minimum-error feature for prediction; this feature can be either content-based or behavior-based. Therefore, it just retains its original accuracy. Naïve Bayes uses Eq.(1) to compute posterior probability of each class, thus the combined features can not guarantee accuracy improvement. For SVM, kNNs and Decision Tree, the combination of content-based and behavior-based features will give more information to each sample. Therefore, applying these algorithms can generate more appropriate models for classification. In successive experiments, datasets that contain both features were used to represent executables.

4.2 Ensemble vs. Individual Learning

Table 5 shows the experimental results of overall accuracy of all ten algorithms based on 4 datasets.

Table 5. Rank for each algorithm

Algorithm	Datasets				Avg.	Rank
	A1	B1	C1	D1		
NB	74.56%	77.55%	78.99%	81.59%	78.17%	**10**
SVM	92.88%	93.67%	88.61%	85.60%	90.10%	**8**
kNNs	93.49%	92.14%	89.55%	91.93%	91.78%	**5**
DT	92.63%	92.07%	88.32%	89.63%	90.66%	**6**
OneR	88.64%	83.37%	75.35%	73.44%	80.20%	**9**
Voting	94.22%	93.98%	88.32%	86.10%	90.66%	**6**
Bagging DT	94.42%	92.93%	91.08%	91.35%	92.45%	**4**
Boosting DT	95.15%	94.80%	92.60%	93.33%	93.97%	**1**
Stacking	94.87%	94.42%	90.49%	91.27%	92.76%	**3**
Grading	94.59%	94.29%	91.84%	91.85%	93.14%	**2**

In our work, bagging and boosting were performed on decision tree for 10 times. The ensemble algorithms, including voting, stacking and grading, used NB, SVM, kNNs, DT and OneR as their base learning algorithms. The average accuracy of each algorithm was calculated and ranked based on 4 datasets. It can be seen that the degree of accuracy of ensemble learning algorithms was higher than the individual algorithms.

4.3 The Performance of SVM-AR

The performance of SVM-AR is illustrated in Figs.6-8. We tested on four datasets (A1, B1, C1, and D1) to avoid the data affecting the performance. It can be seen that SVM-AR outperforms nearly all popular ensemble learning algorithms at overall accuracy, detection rate and false positive rate in each case. Fig.7 and Fig.8 show that the malware's percentage in the dataset affects the detection and false positive rate. Except SVM-AR, most of the algorithms didn't perform well on false positive rate in the datasets with higher percentage of malware. The reason is that most of the algorithms tend to misclassify the instances of the less represented class, leading

to high false positive rate. However, SVM-AR can reduce the false positive rate by applying its association rules on the malicious side separated by a SVM.

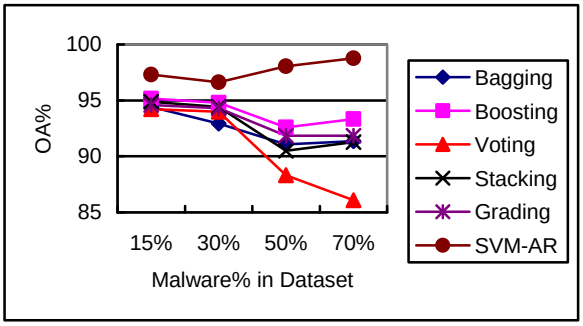

Fig.6. OA comparison between SVM-AR and popular ensemble algorithms.

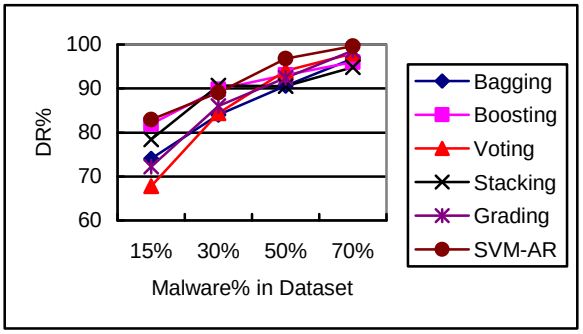

Fig.7. DR comparison between SVM-AR and popular ensemble algorithms.

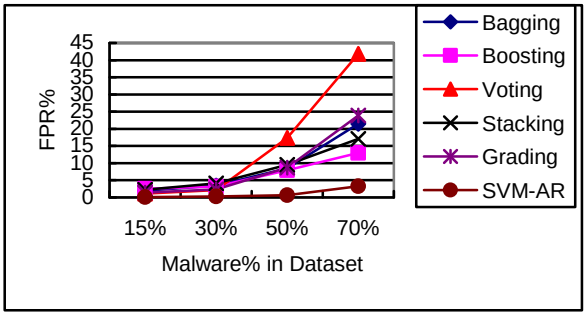

Fig.8. FPR comparison between SVM-AR and popular ensemble algorithms.

4.4 Execution Time

Figs.9-10 show the execution time used by each of the aforementioned algorithm during the training and testing phase.

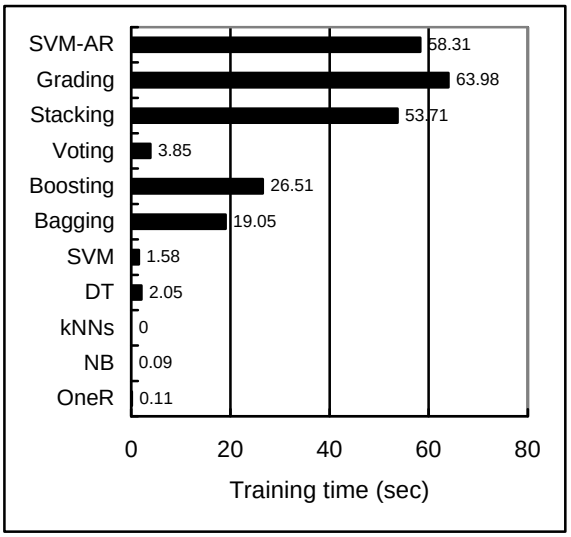

Fig.9. Training time comparison of learning algorithms.

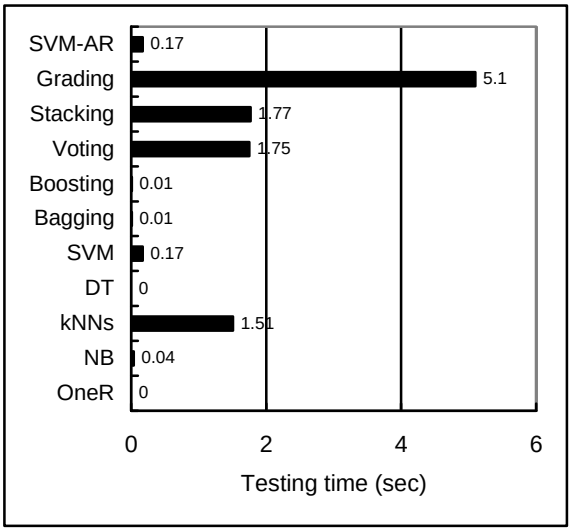

Fig.10. Testing time comparison of learning algorithms.

In general, the execution time of an ensemble algorithm is longer than an individual algorithm in both phases. It is because the ensemble learning algorithms must generate multi-classifiers during the training phase and combine the predictions of multi-classifiers to make the final decision. The boosted DT and bagging DT in testing execution time are exceptions. They are even better than some individual algorithms. The reason is that the time to execute decision tree for testing is negligible, hence the time needed to run it several times is still very short.

The training execution time for SVM-AR is longer than most of the algorithms. The

reason is that it uses Apriori scheme to identify the related association rules, which is a time consuming process. However, SVM-AR's testing execution time is better than most of the ensemble algorithms. It is because the time for applying association rules to test samples is negligible, the testing time of SVM-AR is dominated the execution time of SVM. Therefore, it makes SVM-AR even comparable to individual learning algorithm.

Practically speaking, once the detection model is generated from training phase, it didn't need to retrain the model for each test case. Thus, testing time is more critical than training time for machine learning application. From this viewpoint, our method, SVM-AR, is very efficient in terms of execution time.

V. CONCLUSIONS

Machine learning can be used to detect new and unknown malware. However, the limitation of this technique is its high false rate. The performance of machine learning is influenced by two main factors: features and algorithms. In this research, we set out to improve the precision of machine learning through these two factors. The conclusions drawn from the experimental results are:

(1) Content-based and behavior-based features are complementary to each other. Combining both of them can improve the overall accuracy of malware detection for most of classification algorithms.
(2) In general, ensemble learning algorithms outperform individual learning algorithms in accuracy, but at a cost of longer execution time.
(3) A new ensemble learning model, SVM-AR, was proposed in this article. The experimental results revealed that SVM-AR offers a better performance than other well-known ensemble algorithms in terms of accuracy, detection rate, and false positive rate. Moreover, SVM-AR is comparable to any individual learning algorithm in test execution time.

The limitation in this research is the analysis process used to detect unknown executables' dynamic behaviors within a VMWare environment with certain sharewares. As a result, this study was forced to adopt a semi-automated approach, even though the efficiency of SVM-AR, a hybrid classification algorithm proposed by this study, is promising in real-time malware detection. Future work should focus on how to conduct similar classification processes in an automated manner.

ACKNOWLEDGMENT

This work was supported by international Collaboration for Advancing Security Technology (iCast) project sponsored by the National Science Council, Taiwan, under the Grants No. NSC96-3114-P-606-001Y.

REFERENCES

[1] Schultz, M. G., Eskin, E., Zadok, E., and Stolfo, S. J., "Data Mining Methods for Detection of New Malicious Executables," Proc. of the 2001 IEEE Symposium on Security and Privacy, Los Alamitos, pp. 38-49, 2001.
[2] Shih, D. H., Chiang, H. S., and Yen, C. D., "Classification Methods in the Detection of New Malicious Emails," Information Sciences, Vol. 172, Issue 1-2, pp. 241-261, 2005.
[3] Kolter, J. Z. and Maloof, M. A., "Learning to Detect and Classify Malicious Executables in the Wild," Journal of Machine Learning Research, Vol. 7, pp. 2721-2744, 2006.
[4] Zhang, B. Y., Yin, J. P., Hao, J. B., Zhnag, D. X., and Wang, S. Lin, "Using Support Vector Machine to Detect Unknown Computer Viruses," International Journal of Computational Intelligence Research, Vol. 2, No. 1, pp. 100-104, 2006.
[5] Moskovitch, R., Elovici, Y., and Rokach, L., "Detection of Unknown Computer Worms Based on Behavioral Classification of the Host," Computational Statistics and Data Analysis, Vol. 52, Issue 9, pp. 4544-4566, 2008.
[6] Hu, Y. T., Chen, L. A., Xu, M., Zheng, N., and Guo, Y. H., "Unknown Malicious Executables Detection Based on Run-time Behavior," Proc. of the 5th International Conference on Fuzzy Systems and Knowledge Discovery, Shandong, China,

pp. 391-395, 2008.

[7] Dolev, S. and Elovici, Y., "Unknown Malcode Detection Using OPCODE Representation," Proc. of the 1st European Conference on Intelligence and Security Informatics, Esbjerg, Denmark, pp. 204-215, 2008.

[8] Assaleh, T. A., Cercone, N., Keselj, V., and Sweidan, R., "N-gram-based Detection of New Malicious Code," Proc. of the 28th Annual International Computer Software and Application Conference, HongKong, pp. 41-42, 2004.

[9] Zhang, B. Y., Yin, J. P., and Hao, J. B., "Using Fuzzy Pattern Recognition to Detect Unknown Malicious Executables Code," Proc. of the Second International Conference on Fuzzy Systems and Knowledge Discovery, Changsha, China, pp. 629-634, 2005.

[10] Henchiri, O. and Japkowicz, N., "A Feature Selection and Evaluation Schedule for Computer Virus Detection," Proc. of the 6th International Conference on Data Mining, HongKong, pp. 1-6, 2006.

[11] Ye, Y. F., Wang, D. D., Li, T., and Ye, D. Y., "An Intelligent PE-malware Detection System Based on Association Mining," Journal in Computer Virology, Vol. 4, No.4, pp. 323-334, 2008.

[12] Lai, Y. X., "A Feature Selection for Malicious Detection," Proc. of the 9th International Association for Computer and Information Science Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/ Distributed Computing, Phuket, Thailand, pp. 365-370, 2008.

[13] Masud, M. M., Khan, L., and Thuraisingham, B., "A Hybrid Model to Detect Malicious Executables," Proc. of the IEEE International Conference on Communications, Glasgow, UK, pp. 1443-1448, 2007.

[14] Masud, M. M., Khan, L., and Thuraisingham, B., "A Scalable Multi-level Feature Extraction Technique to Detect Malicious Executables," Information Systems Frontiers, Vol. 10, No. 1, pp. 33-45, 2008.

[15] Reddy, K.S., Dash, S.K., and Pujari, A.K., "New Malicious Code Detection Using Variable Length n-grams," Lecture Notes in Computer Science, Vol. 4332, pp. 276-288, 2006.

[16] Ye, Y.Y., Chen, L.F., Wang, D.D., Li, T., Jiang, Q.S., and Zhao, M., "SBMDS: an interpretable string based malware detection system using SVM ensemble with bagging,'' Journal in Computer Virology, Vol. 5, No.4, pp. 283-293, 2009.

[17] Shafiq, M.Z., Tabish, S.M., Mirza, F., and Farooq, M., "PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime,'' Lecture Notes in Computer Science, Vol. 5758, pp. 121-141, 2009.

[18] Liu L. and Shao, K., "An Approach of Malicious Executables Detection on Black & Gray Based on AdaBoost Algorithm," Proc. of the 2nd International Conference on Anti-counterfeiting, Security, and Identification, Guyang, China, pp. 88-92, 2008.

[19] Liu, Z. and Zhang, X. S., "A Novel Approach to Malicious Executables Detection and Containment Based on Distributed System Architecture," Proc. of the 4th International Conference on Natural Computation, Jinan, China, pp. 160-164, 2008.

[20] Zhang, B. Y., Yin, J. P., Tang, W. S., Hao, J. B., and Zhang, D. X., "Unknown Malicious Codes Detection Based on Rough Set Theory and Support Vector Machine," Proc. of 2006 International Joint Conference on Neural Networks, Vancouver, Canada, pp. 2583-2587, 2006.

[21] Wang, T. Y., Horng, S. J., Su, M. Y., Wu, C. H., Wang, P. C., and Su, W. Z., "A Surveillance Spyware Detection System Based on Data Mining Methods," Proc. of 2006 International Joint Conference on Neural Networks, Vancouver, Canada, pp. 3236-3241, 2006.

[22] Moskovitch, R., Nissim, N., and Elovici, Y., "Malicious Code Detection Using Active Learning," Lecture Notes in Computer Science, Vol. 5456, pp. 74-91, 2009.

[23] Zhang, B. Y., Yin, J., Zhang, D., Hao, J., "Unknown Computer Virus Detection

Based on K-Nearest Neighbor Algorithm," Computer Engineering and Applications, Vol. 6, pp. 7-10, 2005.

[24] Stopel, D., Boger, Z., Moskovitch, R., Shahar, Y., and Elovici, Y., "Improving Worm Detection with Artificial Neural Networks through Feature Selection and Temporal Analysis Techniques," Transactions on Engineering, Computing and Technology, Vol. 15, pp. 202-208, 2006.

[25] Stopel, D., Boger, Z., Moskovitch, R., Shahar, Y., and Elovici, Y., "Application of Artificial Neural Networks Techniques to Computer Worm Detection," Proc. of 2006 International Joint Conference on Neural Networks, Vancouver, Canada, pp. 2362-2369, 2006.

[26] Menahem, E., Shabtai, A., Rokach, L., and Elovici, Y., "Improving Malware Detection by Applying Multi-inducer Ensemble," Computational Statistics and Data Analysis, Vol. 53, Issue 4, pp. 1483-1494, 2009.

[27] Opitz, D. and Maclin, R., "Popular Ensemble Methods: A Empirical Study," Journal of Artificial Intelligence Research, Vol. 11, pp. 169-198, 1999.

[28] Seewald, A. K., "Towards Understanding Stacking—Studies of a General Ensemble Learning Scheme," Ph. D. Dissertation, Vienna University of Technology, 2003.

[29] Seewald, A. K. and Furnkranz, J., "An Evaluation of Grading Classifiers," Proc. of 4th International Symposium on Intelligent Data Analysis, Lisbon, Portugal, pp. 115-124, 2001.

[30] Kotsiantis, S. B. and Pintelas, P. E., "A Hybrid Decision Support Tool—Using Ensemble of Classifiers," Proc. of the 6th International Conference on Enterprise Information Systems, Porto, Portugal, Vol. 2, pp. 448-453, 2006.

[31] Treinen, J. J. and Thurimella, R., "A Framework for the Application of Association Rule Mining in Large Intrusion Detection Infrastructures," Proc. of 9th International Symposium on Recent Advances in Intrusion and detection, Hamburg, Germany, pp. 1-18, 2006.

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

[33] DARPA Datasets, http://www.ll.mit.edu/mission/Communications/ist/corporal/ideval/data/ index.html

[34] PEView, http://hs.hosp.ncku.edu.tw/~cww/html/download.html

[35] VMWare, http://www.vmware.com

[36] PEiD, http://www.softpedia.com/progDownload/PEiD-updated-Download-4102.html

[37] AutoRuns, http://technet.microsoft.com/en-us/sysinternals/bb963902.aspx

[38] ProcessExplorer, http://technet.microsoft.com/en-us/ sysinternals/bb896653.aspx

[39] FileMon,http://technet.microsoft.com/en-us/ sysinternals/bb896642.aspx

[40] Rootkitrevealer, http://technet.microsoft.com/en-us/sysinternals/bb897445.aspx

[41] IceSword,http://antirootkit.com/software/IceSword.htm

[42] Fport, http://www.foundstone.com/us/resources/ proddesc/fport.htm

[43] Burges, C., "A Tutorial on Support Vector Machine for Pattern Recognition," Data Mining and Knowledge Discovery, Vol. 2, No. 2, pp. 121-167, 1998.

[44] Witten, I. H. and Eibe, F., Data Mining: Practical Machine Learning Tools and Techniques, Second Edition, Morgan Kaufmann Press, Chap. 10, p. 413, 2005.

[45] Witten, I. H. and Eibe, F., Data Mining: Practical Machine Learning Tools and Techniques, Second Edition, Morgan Kaufmann Press, Chap. 3, pp. 62-65, 2005.

[46] Witten, I. H. and Eibe, F., Data Mining: Practical Machine Learning Tools and Techniques, Second Edition, Morgan Kaufmann Press, Chap. 4, pp. 84-85, 2005.

[47] Agrawal, R. and Srikant, R., "Fast Algorithms for Mining Association Rules," Proc. of the 20th Very Large Databases Conference, Santiago, Chile, pp. 487-499, 1994.

[48] WEKA,http://www.cs.waikato.ac.nz/ml/weka/.

[49] Chang, C. C. and Lin, C. J., "LIBSVM: a Library for Support Vector Machines," Software available at http://www.csie.ntu.

edu.tw/~cjlin/libsvm/ index.html