International Journal of Computational Intelligence Systems, Vol. 6, No. 4 (July, 2013), 626-638

Enhancing False Alarm Reduction Using Voted Ensemble Selection in Intrusion Detection

Yuxin Meng , Lam-For Kwok

Department of Computer Science, City University of Hong Kong,
Tat Chee Avenue, Kowloon,
Hong Kong SAR, China
E-mail: {ymeng8@student.cityu.edu.hk; cslfkwok@cityu.edu.hk}

Received 2 November 2011

Accepted 6 April 2013

Abstract

Network intrusion detection systems (NIDSs) have become an indispensable component for the current network security infrastructure. However, a large number of alarms especially false alarms are a big problem for these systems which greatly lowers the effectiveness of NIDSs and causes heavier analysis workload. To address this problem, a lot of intelligent methods (e.g., machine learning algorithms) have been proposed to reduce the number of false alarms, but it is hard to determine which one is the best. We argue that the performance of different machine learning algorithms is very fluctuant with regard to distinct contexts (e.g., training data). In this paper, we propose an architecture of intelligent false alarm filter by employing a method of voted ensemble selection aiming to maintain the accuracy of false alarm reduction. In particular, there are four components in the architecture: data standardization, data storage, voted ensemble selection and alarm filtration. In the experiment, we conduct a study involved three machine learning algorithms such as support vector machine, decision tree and k-nearest neighbor, and use Snort, which is an open source signature-based NIDS, to explore the effectiveness of our proposed architecture. The experimental results show that our intelligent false alarm filter is effective and encouraging to maintain the performance of reducing false alarms at a high and stable level.

Keywords: Network Intrusion Detection, Intelligent False Alarm Reduction, Ensemble Selection.

1. Introduction

With the rapid development of Internet, intrusion attacks have become a big security problem in network communication. For instance, a lot of malware software such as Trojan, virus, worm and web-based malicious code always causes a number of network security threats. With the purpose of protecting network security, network intrusion detection systems (NIDSs) have been widely deployed aiming to detect various network exploits in current network environment and have become an essential part of the network security infrastructure.

In general, network intrusion detection systems can be roughly classified into two major types[a]: signature-based NIDS and anomaly-based NIDS. A signature-based NIDS[b] (also called *rule-based NIDS* or *misuse-based NIDS*) aims to discover an

[a]Another type of *stateful protocol analysis* [2] is the process of comparing predetermined profiles of generally accepted definitions of benign protocol activity for each protocol state against observed events to identify deviations.

[b]A signature is a kind of pattern that illustrates precise descriptions of a known attack.

attack by comparing its signatures with packet payloads and this kind of detection systems is very effective in identifying known threats but ineffective in detecting unknown exploits and variants of known threats. While an anomaly-based NIDS detects attacks by comparing its established normal profiles[c] against observed events to identify significant deviations, which is very good at detecting previously unknown threats and exploits. However, it is a very challenging task for these systems to build an accurate profile in most cases because of the complexity of the profile generation. In order to combine the merits of these two types of NIDSs, a hybrid NIDS [3] that contains both signature-based and anomaly-based detection techniques is developed and deployed in current detection strategy.

Problem. Although the NIDSs have become a significant anti-intrusion tool and proved their values in identifying network threats, a big problem of these systems is that too many alarms, especially false alarms, are produced during the detection. The false alarm rate is a key limiting factor to lower the effectiveness of the NIDSs [5]. What is more, it is a real challenge for a security administrator to analyze so many alarms (e.g., more than one thousand a day) most of which are false alarms [4, 6]. The big number of false alarms will definitely increase the chance of analysis mistakes (i.e., ignoring the real alarms of attacks) and reduce the analysis accuracy (i.e., it is hard to identify real alarms from a large number of false alarms).

In the previous work [8], we illustrated that constructing a false alarm filter by means of machine learning algorithms was an effective way to mitigate this problem. In the previous experiment, we used Snort [7, 9] and compared the performance of different machine learning algorithms (e.g., support vector machine, NaiveBayes, decision tree) in filtering out false alarms. The experimental results demonstrate that the algorithms of *SVM (LibSVM)*, *KNN (IBK)* and *DT (J48)* have a better single-algorithm performance than other selected algorithms. In this paper, we mainly investigate the performance of machine learning algorithms by using the notion of ensemble selection in false alarm reduction based on the fact that the ensemble method can usually perform better than any single algorithm [10].

Contributions. In this paper, we summarize our contributions as below. (1) We propose an architecture of intelligent false alarm filter that has the capability of keeping up the false alarm reduction at a high and stable level. (2) We design a method of voted ensemble selection by means of the notion of ensemble selection in our proposed false alarm filter to help refine false alarms based on the specific characteristics of network traffic. (3) We construct three types of Snort alarm datasets in the training and use a real dataset to validate our method. The experimental results indicate that our method can achieve great and stable performance in false alarm reduction. (4) At last, we give an analysis about the effects of weight values on our final results and describe how to determine an appropriate weight value.

The remainder of this paper is organized as follows: Section 2 describes the background of ensemble selection and the three selected machine learning algorithms. Section 3 describes the architecture of our proposed intelligent false alarm filter. In Section 4, we demonstrate the experimental methodology, the method of constructing training datasets and the test results with a real dataset. The analysis of weight values is presented in Section 5. Finally, we conclude our work in Section 6.

2. Background and Related Work

The use of computational intelligence systems (i.e, using machine learning algorithms) is a promising solution to help reduce the false alarms in the area of intrusion detection. Lee and Stolfo [18] first proposed a framework of using data mining algorithms to extract features of audit records and processing these records by means of machine learning algorithms. Then, machine learning algorithms have become epidemic in reducing the false alarms. Pietraszek [19] proposed a system of ALAC (Adaptive Learner for Alert Classification) to reduce false positives based on the confidence of alarm classification, their system could help classify alarms into true or false positives. Then, Law and Kwok [20]

[c] A normal profile is used to represent the normal behavior of a user, host or network connection.

proposed a method of reducing false alarms by using KNN classifier. They established a normal model to represent the sequence of incoming alarms and identified deviations from that model to detect anomalies. Later, Davenport *et al.* [21] presented an analysis on how to control false alarm rate within a desired scope and proposed to minimize the missing rate by using support vector machine (SVM).

In addition, Soleimani *et al.* [17] developed a self-adaptive controlling mechanism that archived the Snort alarms in a well-formed abstracted format aiming to manage the huge amount of alarms. Their key idea is to apply hash technique to make the abstraction process time-effective and then store the abstracted alarms in time-based hierarchical tables. Some other work can be found in [22, 23, 27, 28, 29, 30]. Differently, our work aims to propose an intelligent false alarm filter to enhance the false alarm reduction at a high and stable level by means of ensemble selection to incorporate the merits of several machine learning algorithms.

In the following, we first give a brief introduction of ensemble selection which is established and developed in the machine learning community. Intuitively, the main task of ensemble selection is to automatically detect and combine distinct algorithms to achieve better performance than the parts (e.g., a single algorithm[d]). Then, we roughly describe three machine learning algorithms such as *support vector machine (SVM)*, *decision tree (DT)* and *k-nearest neighbor (KNN)* that are used in our experiments to explore the performance of our developed intelligent false alarm filter. The selection process of these three algorithms is based on their performance in our previous work [8].

2.1. Ensemble Selection

Over the years, a lot of machine learning algorithms have been come up to improve existing approaches. But no one can claim the *best* since their performance depends heavily on the characteristics of the data they are working on. To obtain a better ensemble performance, the notion of ensemble selection was first proposed by Caruana *et al.* [11] as a technique to build ensembles from large diverse classifiers. It employs greedy forward selection to choose algorithms and adds to the ensemble. Compared with the other work in machine learning, ensemble selection considers many more algorithms and classifiers, allows optimizing to arbitrary performance metrics and includes refinements to avoid overfitting problem (which a big problem when choosing algorithms from a large database).

Specifically in ensemble selection, algorithms are trained using as many schemes and control parameters as that can be applied to the issues. Little or no attempt is made aiming to optimize the performance of a single algorithm. The expectation is that some of the algorithms will achieve better performance to solve the problem, in combination with or without other algorithms. In order to avoid the overfitting problem, Caruana *et al.* [11] gave advice that initializing ensembles with the N algorithms (models) that have the best single-algorithm (uni-model) performance, and their experiments [12] showed that the overfitting issue became negligible if the hillclimbing data is sufficient (around 5k points) by using a method of cross-validation.

To explore the performance of different combinations of algorithms, the process of ensemble selection becomes computationally expensive. In this case, we design a method of *voted ensemble selection* (see Section 3.3) with limited alternative algorithms that have the best performance in [8] to make ensemble selection applicable in this work (based on the suggestions from Caruana *et al.* [11]) aiming to reduce the huge workload and consuming time of training. The selected three algorithms are support vector machine (SVM), decision tree (DT) and k-nearest neighbor (KNN).

2.2. Support Vector Machine

This concept of support vector machine [24] contains a set of related supervised learning methods that analyze data and recognize patterns by mapping input feature vectors into a higher dimensional fea-

[d]The single-algorithm performance is used to distinguish the performance of ensemble selection, which usually involves more than one machine learning algorithm.

ture space. To keep the computational load reasonable, the mappings used by SVM algorithms are designed to ensure that dot products may be computed easily in terms of the variables in the original space by defining a kernel function $K(x,y)$. The vectors define the hyperplanes that can be chosen to be linear combinations with parameters α_i of images of feature vectors. With this choice of a hyperplane, the points x in the feature space can be mapped into the hyperplane by the relation specified as below:

$$\sum_i \alpha_i K(x_i, x) = constant$$

If $K(x,y)$ becomes small with y grows further from x, every element in the sum measures the degree of closeness of the test point x to the corresponding data base point x_i.

2.3. Decision Tree

Decision tree [25] is a tree-like model aiming to classify the given datasets according to the values of its attributes. A node of a decision tree shows an attribute and its relevant edges. Each edge connects *two nodes* or *a node and a leaf*. A leaf is explicated with a decision value to determine the class of the input data. The general algorithm for building decision trees is shown as follows:

- Checking for base cases;
- For each attribute a. Finding the normalized information gain from splitting on a;
- Let *a_best* be the attribute with the highest normalized information gain;
- Creating a *decision node* that splits on *a_best*;
- Recursing on the sublists obtained by splitting on *a_best*, and adding those nodes as children of node.

2.4. K-nearest Neighbor Algorithm

This algorithm [26] is a method of classifying objects based on closest training examples in the feature space. Also, it is a type of instance-based learning where the function is only approximate locally and all computation is deferred until classification. An object is classified by a majority vote of its neighbors with the object being assigned to the class, which is the most common among its k nearest neighbors (k is a positive integer). In the classification phase, k can be a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is the most frequent among the k training samples nearest to that query point. For example, if k=1, then the object is simply assigned to the class of its nearest neighbor.

3. The Architecture of Intelligent False Alarm Filter

In this section, we mainly give an in-depth description of our proposed architecture of intelligent false alarm filter. The architecture contains four major components: *data standardization*, *data storage*, *voted ensemble selection* and *alarm filtration*. A high-level diagram of our proposed architecture is demonstrated in Fig. 1.

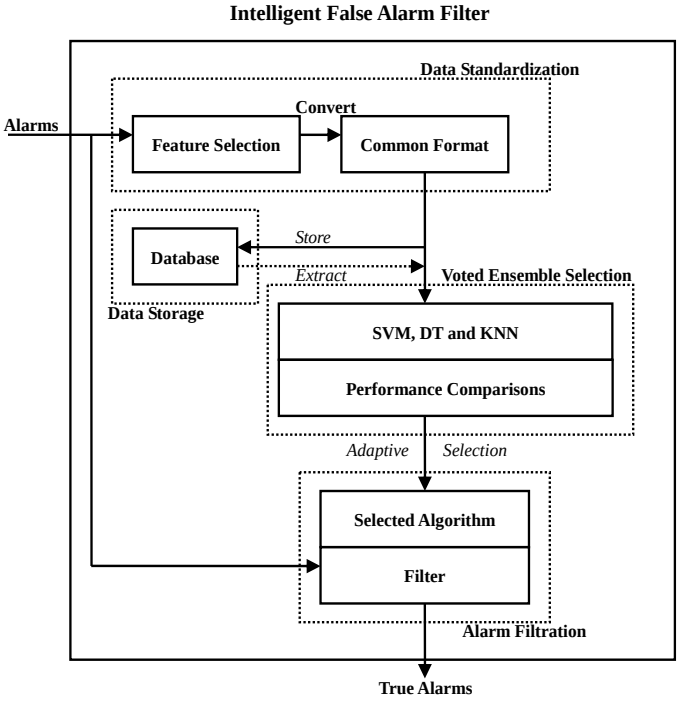

Fig. 1. The architecture of our proposed intelligent false alarm filter.

3.1. Data Standardization

By using different types of NIDSs (e.g., Snort [9], Bro [13]), the formats of produced alarms are definitely not the same. Therefore, we should first standardize these distinct alarms into a common format as defined in the alarm filter. For the component of *data standardization* as shown in Fig. 1, its first work is to choose some proper features from NIDS's alarms. Then, format conversion can be performed after the process of feature selection.

Take Snort (version 2.9.0.5) [9] as an example, we present three specific instances of its alarms in Fig. 2.

03/08-21:10:44.016397 [**] [1:648:10] SHELLCODE X86 NOOP [**] [Classification: Executable Code was Detected] [Priority: 1] {TCP} 172.16.114.148:20 -> 197.182.91.233:1170

(a) ICMP PING

03/08-21:09:20.951492 [**] [1:382:5] ICMP PING [**] [Classification: Misc activity] [Priority: 3] {ICMP} 172.16.114.148 -> 197.218.177.69

(b) ICMP Echo Reply

03/08-21:01:59.803587 [**] [1:1201:8] ATTACK-RESPONSES 403 Forbidden [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 172.16.112.100:80 -> 206.48.44.18:1062

(c) ATTACK-RESPONSES Invalid URL

Fig. 2. Instances of Snort alarms.

In Fig.2 (a), this Snort alarm is to alert *SHELLCODE x86*, which is a type of executable code. The target type of packets is TCP and the priority of this alarm is 1. In general, a lower priority number indicated a higher priority alarm. "172.16.114.148:20 $->$ 197.182.91.233:1170" shows the source IP address and the destination IP address. "20" is the source port number while "1170" is the destination port number. The time of occurrence for this alarm is "03/08-21:10:44.016397".

The Snort alarm in Fig. 2 (b) is similar, which is classified as *Miscellaneous activity* of ICMP PING. The priority number of 3 presents that the priority of this alarm is lower than that in Fig. 2 (a). Whereas the priority of the Snort alarm in Fig. 2 (c) is between the above two alarms.

With the understanding of the format of Snort alarms, we then select and construct 8 features to represent the Snort alarms as follows: *description*, *classification*, *priority*, *packet type*, *source IP address*, *source port number*, *destination IP address* and *destination port number*. The selected features for the corresponding alarms in Fig. 2 are presented in Fig. 3.

In Fig. 3, the value "0" in the features of *source port* and *destination port* means that no specific port numbers are indicated in the relevant Snort alarms. In the experiment (see Section 4), we used Snort as the NIDS and all its alarms were converted into the common format represented by the above 8 features. We further define the notion of *standard alarms* as the alarms represented by the selected 8 features.

Features	**Figure 2**		
	(a)	***(b)***	***(c)***
Descriptionn	SHELLCODE x86 NOOP	ICMP PING	ATTACK-RESPONSES 403 Forbidden
Classification	Excutable Code was Detected	Misc activity	Attempted Information Leak
Priority	1	3	2
Packet type	TCP	ICMP	TCP
Source IP	172.16.114.148	172.16.114.148	172.16.112.100
Source port	20	0	80
Destination IP	197.182.91.233	197.218.177.69	206.48.44.18
Destination port	1170	0	1062

Fig. 3. Feature Selection for Snort alarms.

3.2. Data Storage

The main purpose of this component is to provide a passive storage space for the *standard alarms*. Note that different NIDS *standard alarms* may be presented by a distinct set of features. All the standard alarms will be stored in this database after the process of format conversion in the component of *data standardization*. Another function of this database is to provide training data (corresponding to the "Extract" action as illustrated in Fig. 1) to train the fil-

ter. This *extract* action requires expert knowledge (i.e., security administrators can decide what kinds of alarms they are interested in) to label the standard alarms (by indicating *true* or *false* to the attribute of *label_value* as shown in Fig. 4) in advance. Although this action is expensive in some cases and should be arranged in a fixed time, it is very useful and essential to adaptively select the best ensemble algorithms by re-training the alarm filter periodically. We give some example contents of the database in Fig. 4.

@attribute description
@attribute classification
@attribute priority
@attribute type
@attribute source_ip numeric
@attribute source_port numberic
@attribute destination_ip numeric
@attribute destination_port numeric
@attribute label_value

`SHELLCODE x86 NOOP`, `Executable Code was Detected`, 1, TCP, 17216114148, 20, 19718291233M 1170, label_value

`ICMP PING`, `Misc activity`, 2, TCP, 17216112100, 80, 206484418, 1057, label_value

`ATTACK-RESPONSES 403 Forbidden`, `Attempted Information Leak`, 2, TCP, 2061322551, 80, 17216117111, 3457, label_value

Fig. 4. Example contents of standard alarms in the component of data storage.

3.3. Voted Ensemble Selection

Ensemble selection is an ensemble learning method in machine learning and its major advantage is that it can combine a large set of diverse algorithms (models) into a high-performance ensemble through determining proper weights for the combinations. To make this concept of ensemble selection applicable in our architecture, we design and develop a voted ensemble selection with some adaptations and settings for the original ensemble selection as below:

(1) Only applying the three algorithms (SVM, DT and KNN) with default parameter settings for the ensemble. The selection of these three algorithms are based on their single-algorithm performance in our previous work [8];

(2) Randomly choosing algorithms to add to the initialized ensemble rather than the use of greedy forward-stepwise selection;

(3) Using *majority voting* as the decision rule for the voted ensemble selection;

(4) Evaluating and selecting the best suitable ensemble by means of the three measures: *classification accuracy*, *precision of false alarm* and *recall of false alarm*.

The *classification accuracy* is the correct classification rate of both false alarms and true alarms, while the measures of *precision of false alarm* and *recall of false alarm* can be defined as below:

$$precision\ of\ false\ alarm = \frac{N_1}{N_2}. \quad (1)$$

$$recall\ of\ false\ alarm = \frac{N_1}{N_3}. \quad (2)$$

where N_1 represents the number of false alarms classified as false alarm, N_2 represents the number of alarms classified as false alarm, N_3 represents the number of false alarms.

In particular, a perfect ensemble will have a *precision* of *1* showing that no true alarms will be classified as a false alarm; and a *recall* of *1* indicating that every false alarm will be classified as a false alarm. In addition, the decrease of the precision (e.g., a true alarm is classified as a false alarm) is more harmful than the decrease of recall (e.g., a false alarm is classified as a true alarm).

Moreover, we define a *decision value* by using the above three measures to determine which is the best ensemble for our intelligent false alarm filter as follows.

$$decision\ value = w_1 \times A + w_2 \times B + w_3 \times C \quad (3)$$

where A represents *classification accuracy*, B represents *precision of false alarm*, C represents *recall of false alarm*. w_i represents the weight values assigned to the corresponding measures.

In the experiment, we implemented a set of *average weight values* $\{1/3, 1/3, 1/3\}$ for the calculation of *decision values* to explore the initial performance

of ensembles (the analysis of the weight values can be found in Section 5). With the understanding of the above settings and voted ensemble selection, we give the ensemble steps as below:

- To input training data;
- To initialize the three algorithms;
- To establish the ensembles by randomly selecting one or more algorithms from those three algorithms;
- To set a decision rule, we use the *majority voting* (which is a widely used rule) in this work;
- To output results and evaluate the ensemble performance.

3.4. Alarm Filtration

Based on the results of the *voted ensemble selection*, we can determine and choose an ensemble with the best performance in filtering out false alarms. Apparently, the task of this component is to filter out false alarms by using the selected ensemble algorithms. As shown in Fig. 1, the outputs of this component are the true alarms while the false alarms will be filtered out and can be stored in another database for the future use of security administrators (i.e., analyzing the domains of these false alarms, examining whether there are true alarms amongst these false alarms, labeling the true and false alarms as training data).

4. Experiments and Results

In this section, we first describe our experimental environment and experimental methodology. We then describe the method of constructing training datasets for the intelligent false alarm filter. Finally, we show the test results with a real dataset.

4.1. Experimental Methodology and Environment

To evaluate our proposed architecture, we deployed the intelligent false alarm filter off-line with Snort under a constructed environment to explore its performance. The environment is shown in Fig. 5. The Snort is deployed between a source network (consists of several hosts with four VM machines) and a target network (consists of a host with two VM machines) to monitor network traffic and detect attacks. The generated Snort alarms will be forwarded to the intelligent false alarm filter, and a packet generator [14] is installed in the source network to generate malicious packets in triggering Snort to generate true alarms in constructing the training dataset.

Experimental Methodology. To better evaluate our proposed architecture, we construct three types of datasets (*labeled Snort alarms*) for training so as to cover three situations.

(a) *Situation1*: the number of false alarms and true alarms is nearly equivalent.

(b) *Situation2*: the number of false alarms is more than the number of true alarms (most common in real scenarios).

(c) *Situation3*: the number of false alarms is less than the number of true alarms (not common in real scenarios).

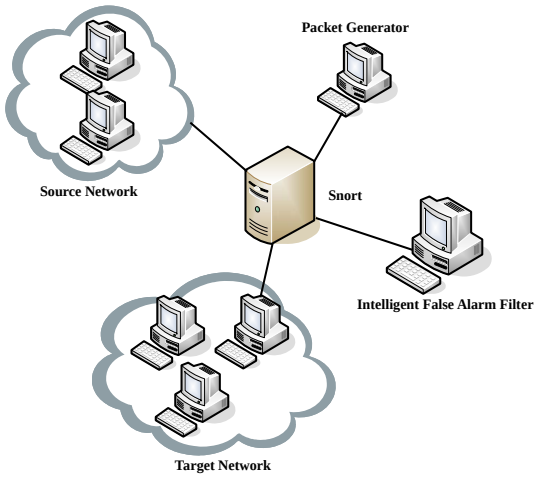

Fig. 5. The deployment of experimental environment.

To implement the three algorithms (SVM, DT and KNN), we used a unified platform WEKA [15] which is an open-source suite of machine learning software providing a lot of implemented machine learning algorithms. In this experiment, we choose specific algorithms of *LibSVM*, *J48* and *IBK* which are implemented in WEKA to represent the corre-

sponding algorithms of SVM, DT and KNN (The implementation is the same as our previous work [8]). This unified platform can avoid the negative effects that may be caused by different implementations of algorithms. By training with the above three types of datasets respectively, we evaluate the performance of ensembles based on the *decision values* (calculated by measures of *classification accuracy*, *precision of false alarm* and *recall of false alarm*) and then explore the performance of the intelligent false alarm filter with a real dataset.

4.2. Training Data Construction

According to the *ensemble steps* and above *experimental methodology*, we first use the 1999DARPA dataset [16] as the basis to construct the false alarms for the experiment and then trigger some true Snort alarms by using the packet generator in our constructed experimental environment.

In particular, we replay the packets of Week1 and Week3 in the 1999DARPA dataset passing through Snort to obtain the false alarms since the traffic of these two weeks is free of attacks. In this case, we treat all the triggered Snort alarms as false alarms. The number of the false alarms and the top 5 most frequent alarms are shown in Table 1 and Table 2 respectively.

In Table 1, we find that at least one thousand alarms are triggered everyday, which is indeed a big challenge for a security administrator. The Snort alarms with "ICMP Destination Unreachable Port Unreachable" are the most frequent false alarms according to Table 2. Thus, we have obtained a total number of 13669 false alarms for Week 1 and Week3 as shown in Table 1.

For the true alarms, we used the packet generator in the constructed environment (as shown in Fig. 5) to trigger Snort to produce real alarms (*true alarms*) by artificially sending it a number of malicious packets (i.e., through simulating kinds of network attacks). We treat the corresponding Snort alarms produced in this scenario as true alarms based on the fact that real attacks have been occurred. We randomly select 8576 alarms of them as *real alarms* for the training phase.

Table 1. The number of false alarms in Week1 and Week3.

Week Day	**Week1**	**Week3**
Monday	1305	1173
Tuesday	1015	1243
Wednesday	2759	1344
Thursday	1090	1290
Friday	1178	1272
Total number	7347	6322

Table 2. The Top 5 frequent types of false alarms in Week1 and Week3.

False Alarm Type	*Occurrence*
ICMP Destination Unreachable Port Unreachable	6596
ICMP Ping	2511
ICMP Echo Reply	2511
ATTACK-RESPONSES 403 Forbidden	636
CHAT IRC message	267

The construction of the three types of datasets is illustrated as below.

(a) *DATASET1* (corresponding to *Situation1*) : This dataset contains 6588 false alarms and 6450 true alarms which are randomly selected from the above false and true alarms. The occupancy ratio of false alarms and true alarms is close to 1:1.

(b) *DATASET2* (corresponding to *Situation2*) : This dataset contains both randomly selected 12546 false alarms and 6743 true alarms. The occupancy ratio of false alarms and true alarms is close to 2:1

(c) *DATASET3* (corresponding to *Situation3*) : This dataset contains 4087 false alarms and 8109 true alarms by the use of random selection. The occupancy ratio of false alarms and true alarms is close to 1:2.

The use of random selection makes the selected alarms in all these three datasets different in order to better explore the performance of the intelligent false alarm filter.

4.3. Experimental Results

In this section, we first present the training results of the above three datasets (*DATASET1*, *DATASET2* and *DATASET3*) and then we show the results of us-

ing a real dataset (extracted from real network traffic) to test the performance of our approach.

4.3.1. Training Results

The measures of *classification accuracy* , *precision of false alarm* and *recall of false alarm* are important factors for the evaluation of our approach. The results of these measures are illustrated in Table 3, Table 4 and Table 5 corresponding to the above three constructed datasets (*DATASET1*, *DATASET2* and *DATASET3*) respectively.

Table 3. The training results with *DATASET1*.

Ensembles	*A*	*B*	*C*
J48	0.9120	0.8900	0.9670
KNN	0.9095	0.8970	0.9530
LibSVM	0.8821	**0.9160**	0.8760
J48+KNN	0.9113	0.8930	0.9610
J48+LibSVM	0.8990	0.9020	0.9250
KNN+LibSVM	0.8984	0.9060	0.9190
J48+KNN+LibSVM	**0.9158**	0.8940	0.**9690**

Table 4. The training results with *DATASET2*.

Ensembles	*A*	*B*	*C*
J48	0.9270	0.9090	**0.9980**
KNN	0.9257	0.9140	0.9900
LibSVM	0.8120	**0.9380**	0.7910
J48+KNN	0.9270	0.9120	0.9940
J48+LibSVM	0.8731	0.9230	0.8980
KNN+LibSVM	0.8725	0.9260	0.8940
J48+KNN+LibSVM	**0.9294**	0.9120	**0.9980**

Table 5. The training results with *DATASET3*.

Ensembles	*A*	*B*	*C*
J48	0.8551	0.7020	0.8920
KNN	0.8534	0.7050	0.8730
LibSVM	0.8400	0.7030	0.8000
J48+KNN	0.8546	0.7030	0.8840
J48+LibSVM	0.8488	0.7030	0.8520
KNN+LibSVM	0.8492	**0.7070**	0.8420
J48+KNN+LibSVM	**0.8589**	0.7030	**0.9100**

For the above tables, *A* represents *classification accuracy*, *B* represents *precision of false alarm*, *C* represents *recall of false alarm*. Moreover, 10-folder cross-validation is used in all training processes to ensure more reliable results.

Based on the training results, the ensemble of *J48+KNN+LibSVM* has the best performance regarding to the measures of *A* (*classification accuracy*) and *C* (*recall of false alarm*) for all the three datasets. We has emphasized the best results with bold in these tables. For other ensembles and single algorithms, their performance is not stable. Overall, the training results conform to the *findings* that ensembles can improve the performance of a single algorithm. For example, the *classification accuracy* of *LibSVM* is 0.8821 in Table 3, while the improvement is made by ensemble with *J48* (0.8990) or *KNN* (0.8984). What is more, our ensemble method can achieve higher values of *classification accuracy*, *precision of false alarm* and *recall of false alarm* in using *DATASET2* than the other two datasets. *DATASET2* represents a common case in real scenarios which is our focus.

To further compare the performance of different ensembles and illustrate the ensemble selections of the intelligent false alarm filter, we present the results of *decision values* (the used weight values are $\{1/3, 1/3, 1/3\}$) for all ensembles and single algorithms in Table 6.

(In Table 6, *DS1* represents *DATASET1*, *DS2* represents *DATASET2* and *DS3* represents *DATASET3*).

Table 6. The results of *decision values* with average weight values $\{1/3, 1/3, 1/3\}$.

Decision Values	*DS1*	*DS2*	*DS3*
J48	0.9230	0.9447	0.8164
KNN	0.9198	0.9432	0.8105
LibSVM	0.8914	0.8470	0.7810
J48+KNN	0.9218	0.9443	0.8139
J48+LibSVM	0.9087	0.8980	0.8013
KNN+LibSVM	0.9078	0.8975	0.7994
J48+KNN+LibSVM	**0.9263**	**0.9465**	**0.8240**

In Table 6, it is easily visible that the ensemble of *J48+KNN+LibSVM* has the best decision values for all the three datasets (0.9263 for the DATASET1, 0.9465 for the DATASET2 and 0.8240

for the DATASET3) which means that this ensemble will be selected in our intelligent false alarm filter as the best choice in filtering out false alarms.

Moreover, as shown in Fig. 6, we present the training times of building models for all the three training datasets by using different ensembles and single algorithms. The training time is a major and critical factor to evaluate whether the intelligent false alarm filter is applicable in real scenarios. As illustrated in Fig.6, the single algorithm of *LibSVM* and the ensembles of *J48+LibSVM*, *KNN+LibSVM* and *J48+KNN+LibSVM* can decide the time consumption in the training for the alarm filter. In general, more data instances require more training time. Inspiringly, the whole time consumption of our false alarm filter is less than 3 minutes for the *DATASET2* (contains about 20k instances) which is very acceptable in real deployment.

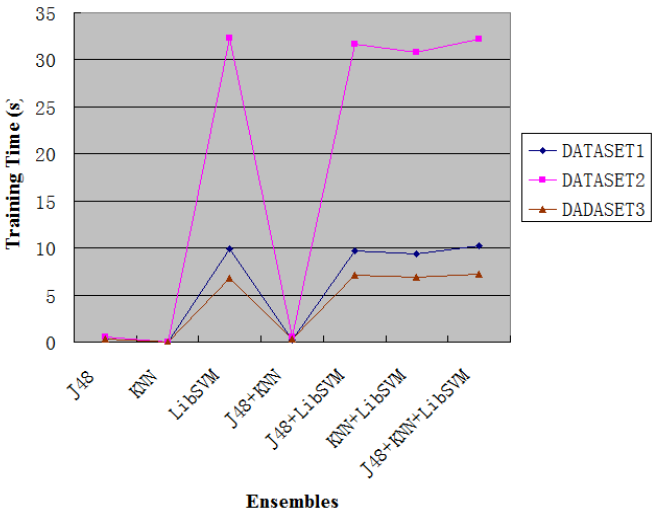

Fig. 6. The training time for different ensembles and single algorithms.

4.3.2. *Testing Results*

To evaluate the intelligent false alarm filter in real scenarios, we used a real dataset (contains the Snort alarms from real network traffic) for the testing. The traffic in the real dataset was captured by the Honeypot system (deployed with Snort) which was deployed in our CSlab by HoneybirdHK. The honeypot can counteract attempts at unauthorized use of information systems and capture all relevant events. The settings of our intelligent false alarm filter is the same as that in the training phase. The specific testing results are illustrated in Fig. 7.

In the testing, we first randomly selected 20% (about 6k instances) of the total Snort alarms in the training, and then we gradually increase the number of alarms to evaluate the performance of our intelligent false alarm filter. After every hour, we then use new training data (manually labeled alarms) to re-train the alarm filter. The action of labeling was guided by the HoneybirdHK.

In Fig. 7, we use the algorithm of *J48* (which has the best *single-algorithm performance* in the training phase according to the *decision values* in Table 6) in the comparison with our method. The results indicate that the performance of our method is gradually improved (from 78.98% to 88.68%) and becomes more and more stable, which achieves a better performance than that of *J48*. Overall, the testing results indicate that our intelligent false alarm filter can achieve a high filtration rate (over 80% in average) in real scenarios.

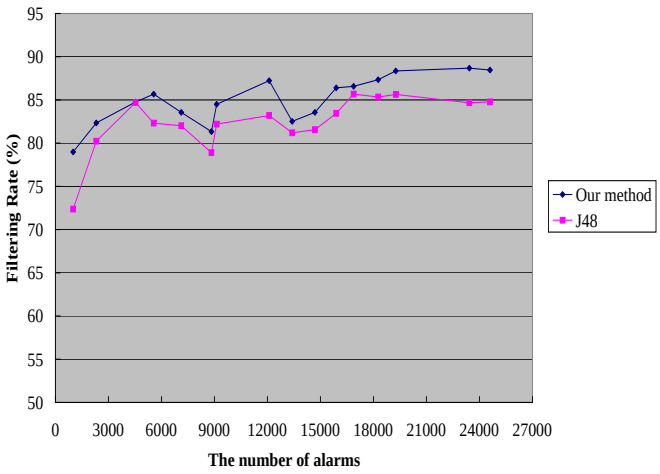

Fig. 7. The filtration rate for the real dataset.

5. Discussions and Analysis on Weight Values

The weight value is one of the key element in affecting the performance of our intelligent false alarm filter since the selection of ensembles is determined

by Eq. (3). The three measures are *classification accuracy*, *precision of false alarm* and *recall of false alarm*. A higher filtration rate does not mean a better set of weight values. In general, a good and appropriate set of weight values should maintain the false alarm filter have a high confidence in the results of ensemble selection.

In our experiment, we used the average weight values {1/3, 1/3, 1/3} in Eq. (3) to explore the initial performance of the false alarm filter. This set of average weight values emphasizes the natural impact of each measure on our filer and does not bias towards any particular measure.

Intuitively, the weight values can be partially decided by the definitions of the three measures. As illustrated in Section 3.3, we note that the decrease of the *precision of false alarm* (e.g., a true alarm is classified as false alarm) is more harmful than the decrease of the *recall of false alarm* (e.g., a false alarm is classified as true alarm). This impact is based on the definitions of both *precision of false alarm* and *recall of false alarm*. In this case, a greater weight value can be given to *precision of false alarm* rather than *recall of false alarm*.

Moreover, the false alarm filter prefers a higher *recall of false alarm* (that the number of false alarms classified as false alarm divides by the number of false alarms) than the *classification accuracy* since a higher detection rate of false alarms can greatly improve the filtration performance of our filter. For the *classification accuracy*, it mainly provides an accuracy in classifying both true alarms and false alarms, therefore, a greater weight value should be given to the measure of *recall of false alarm*. Thus, the weighted ranking of these three measures is: *precision of false alarm* $>$ *recall of false alarm* $>$ *classification accuracy*.

Overall, a higher *decision value* is expected by the false alarm filter, which means a higher confidence in the ensemble selection. To further validate our analysis of the weight values, we implemented a set of weight values {1/6, 1/3, 1/2} by means of the same training datasets in Section 4.3.1. The selection of the weight values {1/6, 1/3, 1/2} is providing an example according to the weighted ranking: *precision of false alarm* $>$ *recall of false alarm* $>$ *classification accuracy*. The results are presented in Table 7.

Table 7. The results of the *decision values* with the weight values {1/6, 1/3, 1/2}.

Decision Values	*DS1*	*DS2*	*DS3*
J48	0.9325	0.9565	0.8225
KNN	0.9271	0.9540	0.8137
LibSVM	0.8904	0.8435	0.7743
J48+KNN	0.9301	0.9555	0.8188
J48+LibSVM	0.9130	0.9022	0.8018
KNN+LibSVM	0.9112	0.9011	0.7982
J48+KNN+LibSVM	**0.9351**	**0.9579**	**0.8325**

As shown in Table 7, nearly all the decision values are increased. For instance, the decision value of the ensemble of *J48+KNN+LibSVM* increases from 0.9263 to 0.9351 for DATASET1, from 0.9456 to 0.9579 for DATASET2 and from 0.8240 to 0.8325 for DATASET3. Note that a higher decision value means a higher confidence in selecting the ensembles so that the weight values of {1/6, 1/3, 1/2} is preferred by the filter than the average weight values of {1/3, 1/3, 1/3}. But the results of the best ensemble are not affected because the ensemble of *J48+KNN+LibSVM* still has the greatest decision values for all the three datasets. The reason is that the ensemble of *J48+KNN+LibSVM* achieves the best performance with regard to measures of *precision of false alarm* and *classification accuracy* compared to the other algorithms. Therefore, this ensemble definitely has the highest decision values according to Eq. (3).

On the contrary, we present a negative example with the weight values {1/2, 1/3, 1/6} (*precision of false alarm* $<$ *recall of false alarm* $<$ *classification accuracy*) that break our suggested weighted ranking. The results of the decision values are shown in Table 8. Based on the results, we find that the decision values of most ensembles and algorithms are decreased (i.e., the decision value of the ensemble of *J48+KNN+LibSVM* decreases from 0.9263 to 0.9174 for the DATASET1, from 0.9456 to 0.9350 for the DATASET2 and from 0.8240 to 0.8155 for the DATASET3). These lower decision values show a lower confidence in the ensemble selection than

our example with weight values {1/6, 1/3, 1/2}.

Table 8. The results of the *decision values* with the weight values {1/2, 1/3, 1/6}.

Decision Values	*DS1*	*DS2*	*DS3*
J48	0.9138	0.9328	0.8102
KNN	0.9126	0.9325	0.8072
LibSVM	0.8924	0.8505	0.7877
J48+KNN	0.9135	0.9332	0.8090
J48+LibSVM	0.9043	0.8939	0.8007
KNN+LibSVM	0.9044	0.8939	0.8006
J48+KNN+LibSVM	**0.9174**	**0.9350**	**0.8155**

Both the results in Table 7 and Table 8 conform to our analysis that the weighted ranking (*precision of false alarm* > *recall of false alarm* > *classification accuracy*) is an appropriate and correct way to decide the weight values with high confidence in ensemble selection for our false alarm filter system. In real settings, we suggest that administrators should choose the specific weight values based on the above weighted ranking.

6. Conclusions

The large number of false alarms is a big problem in intrusion detection that greatly reduces the effectiveness of NIDSs and causes heavier analysis workload for security administrators.

In this paper, we proposed an architecture of intelligent false alarm filter and designed a method of voted ensemble selection to enhance the performance of false alarm reduction at a high and stable level. Specifically, our proposed architecture consists of four major components: data standardization, data storage, voted ensemble selection and alarm filtration. The designed method of voted ensemble selection makes the notion of ensemble selection applicable in the filter. In the evaluation, we use three measures such as *classification accuracy*, *precision of false alarm* and *recall of false alarm*. Based on these three measures, we define a *decision value* to determine which ensemble can be selected by the false alarm filter. We further conducted an analysis on how to choose appropriate weight values for the filter in achieving a high confidence in the ensemble selection.

In the evaluation, the experimental results indicate that our proposed approach can improve the performance of false alarm reduction and has the capability of maintaining the filtration rate of false alarms at a high and stable level by selecting the best ensemble of machine learning algorithms in actual settings. In addition, a better algorithm can be easily added to the pool without affecting the architecture of the intelligent false alarm filter.

Acknowledgments

We thank the HoneybirdHK for helping provide the real dataset and all reviewers for their valuable comments.

References

1. V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, 2435-2463 (1999). doi:10.1016/S1389-1286(99)00112-7
2. K. Scarfone and P. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," *NIST Special Publication*, 800-94 (Feb 2007).
3. K. Hwang, M. Cai, Y. Chen, and M. Qin, "Hybrid Intrusion Detection with Weighted Signature Generation over Anomalous Internet Episodes," *IEEE Trans. on Dependable and Secure Computing*, **4 (1)**, 41-55 (January 2007). doi:10.1109/TDSC.2007.9
4. J. McHugh, "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory," *ACM Trans. on Information System Security*, **3 (4)**, 262-294 (2000). doi:10.1145/382912.382923
5. S. Axelsson, "The Base-rate Fallacy and the Difficulty of Intrusion Detection," *ACM Trans. on Information and System Security*, **3 (3)**, 186-205 (August 2000). doi:10.1145/357830.357849
6. R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M.A. Zissman, "Evaluating Intrusion Detection Systems: the 1998 DARPA off-line Intrusion Detection Evaluation," *DARPA Information Survivability Conference and Exposition*, 12-26 (2000). doi:10.1109/DISCEX.2000.821506
7. M. Roesch, "Snort-lightweight Intrusion Detection for Networks," *Large Installation System Administration Conference*, 229-238 (1999).

8. Y. Meng and L.-F. Kwok, "Adaptive False Alarm Filter Using Machine Learning in Intrusion Detection," *International Conference on Intelligent Systems and Knowledge Engineering*, 573-584 (2011). doi:10.1007/978-3-642-25658-5_68
9. Snort: The Open Source Network Intrusion Detection System. Homepage, 2011. http://www.snort.org/.
10. T.G. Dietterich, "Ensemble Methods in Machine Learning," *International Workshop on Multiple Classifier Systems*, 1-15 (2000). doi: 10.1007/3-540-45014-9_1
11. R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble Selection from Libraries of Models," *International Conference on Machine Learning*, 137-144 (2004).
12. R. Caruana, A. Munson and R. Niculescu-mizil, "Getting the Most out of Ensemble Selection," *International Conference on Data Mining*, 828-833 (2006). doi:10.1109/ICDM.2006.76
13. Bro: The Powerful Network Analysis Framework. 2011. Homepage, http://bro-ids.org/.
14. Packet Generator: Colasoft Packet Builder. 2011. Homepage, http://www.colasoft.com/packet_builder/
15. WEKA: Waikato Environment for Knowledge Analysis (an open-source tool). Homepage, 2011. http://www.cs.waikato.ac.nz/ml/weka/
16. R. Lippmann, J.W. Haines, D.J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line Intrusion Detection Evaluation," *The International Journal of Computer and Telecommunications Networking*, **34 (4)**, 579-595 (October 2000). doi:10.1016/S1389-1286(00)00139-0
17. M. Soleimani, E.K. Asl, M. Doroud, M. Damanafshan, A. Behzadi, and M. Abbaspour, "RAAS: A Reliable Analyzer and Archiver for Snort Intrusion Detection System," *ACM symposium on Applied computing*, 259-263 (2007). doi:10.1145/1244002.1244067
18. W. Lee and S.J. Stolfo, "A Framework for Constructing Features and Models for Intrusion Detection Systems," *ACM Trans. on Information and System Security (TISSEC)*, **3 (4)**, 227-261 (2000). doi:10.1145/382912.382914
19. T. Pietraszek, "Using Adaptive Alert Clasification to Eeduce False Positives in Intrusion Detection," *International Symposium on Recent Advances in Intrusion Detection*, 102-124 (2004).
20. K.H. Law and L.-F. Kwok, "IDS False Alarm Filtering Using KNN Classifier," *International Workshop on Information Security Applications*, 114-121 (2005). doi: 10.1007/978-3-540-31815-6_10
21. M.A. Davenport, R.G. Baraniuk, and C.D. Scott, "Controlling False Alarms with Support Vector Machines," *International Conference on Acoustics, Speech and Signal (ICASSP)*, 589-592 (May 2006). doi:10.1109/ICASSP.2006.1661344
22. J.W. Ryu, M. Kantardzic, and C. Walgampaya, "Ensemble Classifier based on Misclassified Streaming Data," *IASTED International Conference on Artificial Intelligence and Applications*, 347-354, (2010). doi:10.1007/s10462-009-9124-7
23. S. Aljahdali, "An Effective Intrusion Detection Method using Optimal Hybrid Model of Classifiers," *Journal of Computational Methods in Sciences and Engineering*, **10**, 51-60 (2010).
24. C. Cortes and V. Vapnik, "Support-Vector Networks," *Mach. Learn*, **20 (3)**, 273-297 (1995). doi:10.1023/A:1022627411411
25. Y. Yuan and M.J. Shaw, "Induction of Fuzzy Decision Trees," *Fuzzy Sets Syst*, **69 (2)**, 125-139 (1995). doi:10.1016/0165-0114(94)00229-Z
26. T. Cover and P. Hart, "Nearest Neighbor Pattern Classification," *IEEE trans. on Information Theory*, **13**, 21-27 (1967). doi:10.1109/TIT.1967.1053964
27. C.-Y. Chiu, Y.-J. Lee, C.-C. Chang, W.-Y. Luo, and H.-C. Huang, "Semi-Supervised Learning for False Alarm Reduction," *Industrial Conference on Advances in Data Mining: Applications and Theoretical Aspects*, 595-605, (2010). doi:10.1007/978-3-642-14400-4_46
28. Y. Meng and L.-F. Kwok, "Intrusion Detection using Disagreement-based Semi-Supervised Learning: Detection Enhancement and False Alarm Reduction," *International Symposium on Cyberspace Safety and Security*, 483-497, (2012). doi:10.1007/978-3-642-35362-8_36
29. N. Hubballi, S. Biswas, S. Nandi, "Towards Reducing False Alarms in Network Intrusion Detection Systems with Data Summarization Technique," *Security and Communication Networks*,**6 (3)** , 275-285 (2013). doi:10.1002/sec.562
30. F. Geramiraz, A.S. Memaripour, M. Abbaspour, "Adaptive Anomaly-based Intrusion Detection System Using Fuzzy Controller," *International Journal of Network Security*, **14 (6)**, 352-361 (2012).