SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2015; **8:**3883–3895
Published online 14 July 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1307

RESEARCH ARTICLE

Design of intelligent KNN-based alarm filter using knowledge-based alert verification in intrusion detection†

Weizhi Meng[1*‡], Wenjuan Li[2] and Lam-For Kwok[2]

[1]Infocomm Security Department, Institute for Infocomm Research, Singapore
[2]Department of Computer Science, City University of Hong Kong, Hong Kong SAR, China

ABSTRACT

Network intrusion detection systems (NIDSs) have been widely deployed in various network environments to defend against different kinds of network attacks. However, a large number of alarms especially unwanted alarms such as false alarms and non-critical alarms could be generated during the detection, which can greatly decrease the efficiency of the detection and increase the burden of analysis. To address this issue, we advocate that constructing an alarm filter in terms of expert knowledge is a promising solution. In this paper, we develop a method of knowledge-based alert verification and design an intelligent alarm filter based on a multi-class *k*-nearest-neighbor classifier to filter out unwanted alarms. In particular, the alarm filter employs a rating mechanism by means of expert knowledge to classify incoming alarms to proper clusters for labeling. We further analyze the effect of different classifier settings on classification accuracy with two alarm datasets. In the evaluation, we investigate the performance of the alarm filter with a real dataset and in a network environment, respectively. Experimental results indicate that our alarm filter can effectively filter out a number of NIDS alarms and can achieve a better outcome under the advanced mode. Copyright © 2015 John Wiley & Sons, Ltd.

KEYWORDS
intelligent system; alarm filtration; alert verification; network intrusion detection

***Correspondence**
Weizhi Meng, Infocomm Security Department, Institute for Infocomm Research, Singapore.
E-mail: yuxin.meng@my.cityu.edu.hk

1. INTRODUCTION

Network threats (e.g., malware, virus, and denial-of-service attacks) have become a big challenge with the rapid development of computer networks [1]. To resolve this issue, network intrusion detection systems (NIDSs) are being widely deployed in different kinds of network environments (e.g., an insurance company and a college) with the purpose of defending against various network attacks [2].

Network intrusion detection systems (NIDSs) can be roughly classified into two categories: signature-based NIDSs and anomaly-based NIDSs. A signature-based NIDS like in [3,4] detects an attack by comparing incoming packet payloads with its stored signatures. The signatures (or called *rules*) are a kind of descriptions for a known attack or exploit. By contrast, an anomaly-based NIDS like in [6–8] identifies a potential intrusion by detecting the great deviations between current network events and its pre-defined normal profile. A normal profile can be used to represent and describe a normal network connection.

Target problem. A big issue for these detection systems is that a large number of alarms, especially unwanted alarms such as false alarms and non-critical alarms[§] [9], could be generated during the detection, which would greatly decrease the effectiveness of detection and heavily increase the burden of analysis [10,11]. Here, an *unwanted alarm* mainly refers to an alarm that is not important to a

† A preliminary version of this paper appears in Proceedings of 20th International Symposium on Methodologies for Intelligent Systems (ISMIS 2012), pp. 115–124, 2012 [5].
‡ Corresponding author and is previously known as Yuxin Meng.

§ A non-critical alarm is either a false alarm or a non-critical true alarm [12].

security administrator including both a false-positive and a non-critical alarm. Both detection approaches suffer from this problem:

- Regarding the signature-based approach, its capability of detecting attacks is heavily depending on its available signatures. However, in real deployment, these signatures are not effective in representing multi-step attacks. Therefore, it is hard for such NIDSs to accurately decide the situation of an attack attempt (i.e., whether it is a successful attack or not). In this case, it has to report and alert all detected attack attempts aiming to reduce potential security risks [13].
- For an anomaly-based NIDS, it is very hard for them to establish an accurate normal profile so that many unwanted alarms like non-critical alarms could be generated during the detection. In real settings, the number of false alarms generated by this kind of NIDSs is far more than that generated by a signature-based system [14]. For instance, some traffic accidents such as a sudden increase of network traffic can easily violate and crack the normal profile, causing many false alarms.

Contributions. It is pointed out that the large number of alarms especially false alarms is a key limiting factor to impede the development of intrusion detection systems [15]. To address this issue, constructing an alarm filter is a promising method such as in [9,16]. In this work, we advocate that the filtration performance can be improved by using expert knowledge. In this paper, we thus develop a method of *knowledge-based alert verification* (shortly *KAV*) and further design an *intelligent KAV-based alarm filter* to reduce the number of unwanted alarms. The contributions of our work can be summarized as follows:

- In intrusion detection, *alert verification* is used to determine whether an attack is successful or not. In this work, we develop a notion of *KAV* that aims to determine whether an incoming NIDS alarm is critical or not based on expert knowledge and further design an *intelligent KAV-based alarm filter* to reduce NIDS alarms.
- To implement the *KAV*, we develop an independent component called *rating measurement* in the alarm filter to label NIDS alarms using a multi-class *k*-nearest-neighbor (KNN) classifier. This classifier can be trained with a set of rated alarms. In addition, we also conduct an evaluation on two datasets in order to choose appropriate settings for this classifier.
- The alarm filter can work under two modes: *basic mode* and *advanced mode*. In the *advanced mode*, the alarm filter can intelligently output a number of instances for labeling if given a pool of unlabeled data, by means of the KNN classifier. The *intelligence* here aims to reduce workload and improve efficiency of expert labeling.
- In the evaluation, we conduct three major experiments to investigate the performance of the alarm filter with a real dataset and in a network environment, respectively. Experimental results demonstrate that the alarm filter can lighten the burden of a security analyst by effectively filtering out a large number of unwanted NIDS alarms with affordable CPU workload.

The remaining parts of this paper are organized as follows. In Section 2, we review some related work about constructing an alarm filter in network intrusion detection. Section 3 presents the architecture of our designed *intelligent KAV-based alarm filter* and describes each component in detail. Section 4 describes our results of evaluating different settings of the KNN classifier. The experimental methodology and results are discussed in Section 5. Finally, we conclude our work in Section 6.

2. RELATED WORK AND MOTIVATION

In literature, there are several alternatives for reducing unwanted alarms in the area of intrusion detection. One direct approach is to improve the detection algorithms for a detection system like in [17–22]. By contrast, another approach is to filter out these unwanted alarms in a post-preprocessing manner like conducting alert verification and constructing an alarm filter.

Related work about alert verification. Because most NIDSs are often run without any (or very limited) information of the network resources, these detection systems are likely to produce many false alarms [23]. The method of *alert verification* is used to help filter out NIDS alarms by determining whether an attack is successful or not. For instance, Zhou *et al*. [24] described an approach to verify intrusion attempts by means of lightweight protocol analysis. This approach tracked responses from network applications and verified the results by analyzing the header information. Then Mu *et al*. [25] presented an alert verification approach based on multi-level fuzzy comprehensive evaluation, which considered not only the vulnerability relevance between alerts and target system topologies but also the operating system relevance and network service relevance. Their experiments showed that their approach could deal with the uncertainties better than other alert verification approaches. Later, Bolzoni *et al*. [26] presented *ATLANTIDES* (Architecture for Alert verification in Network Intrusion Detection Systems), an architecture for automatic alert verification exploiting in a structural way. This architecture could be used for reducing false positives both in signature-based and anomaly-based NIDSs. Some other related work can be referred to the work in [27–29].

Related work about alarm filter. In addition to alert verification, another widely used method for reducing unwanted alarms is to construct an alarm filter by means of computational intelligence (e.g., applying machine

DOI: 10.1002/sec

learning algorithms). For instance, Pietraszek [30] proposed and developed a system of adaptive alert classifier that could utilize the feedback from analysts and machine learning techniques to help reduce false positives. Their classifier could drop alerts in terms of their classification confidence. Our idea is similar to theirs, but we employ different ways of clustering alarms and extracting expert knowledge and extend false positive reduction to unwanted alarm reduction.

Then, Law and Kwok [31] designed a false alarm filter by using KNN classifier and achieved good filtration performance. Our work is different from theirs because we use distinct feature extraction and need multi-class classification. Later, Alharbt and Imai [32] constructed an alarm filter by using continuous and discontinuous sequential patterns to detect abnormal alarms. Chiu *et al.* [33] introduced a semi-supervised learning-based mechanism to build an alert filter, which could reduce up to 85% false alarms and still keep a high detection rate. Meng and Kwok [34] presented an adaptive false alarm filter to help NIDS filter out a large number of false alarms. By adaptively selecting the most appropriate machine learning algorithm, the filter can keep a good filtration rate. Meng and Li [12] further designed a non-critical alarm filter to detect and decrease non-critical alarms using contextual information such as application and operating system information. Several other related work regarding the reduction of NIDS false alarms by constructing alarm filters can be referred to the work in [9,16,35,36].

Motivation. Previous studies such as Pietraszek [30] have proven that expert knowledge is very crucial in determining whether an alarm is critical or not. For instance, the importance of an alarm depends on actual network deployment and settings. In this case, an expert (or security administrator) has the capability of determining whether an alarm is useful.

In this work, we thus develop a method of *KAV* and design an intelligent alarm filter to reduce unwanted NIDS alarms. That is, we employ expert knowledge to help determine whether an alarm is critical or not. Our proposed *KAV* and the use of a multi-class KNN classifier also distinguishes our work from others.

3. INTELLIGENT KAV-BASED ALARM FILTER

This section describes the architecture of the designed *intelligent KAV-based alarm filter*, gives an in-depth description of each component, and introduces the extraction of expert knowledge.

3.1. Filter architecture

In Figure 1, we illustrate the high-level architecture of the *intelligent KAV-based alarm filter*, which consists of three main components: *alarm database*, *rating measurement*, and *alarm filter*. The component of *alarm database* is responsible for storing rated alarms and training the KNN classifier. The component of *rating measurement* is responsible for classifying incoming NIDS alarms into appropriate clusters for labeling. The component of *alarm filter* is used to reduce unwanted NIDS alarms in terms of the *rating threshold*. The *expert knowledge* is very important in the filter, which has two main functions: one is to rate alarms in the process of training, while the other is to decide the *rating threshold*.

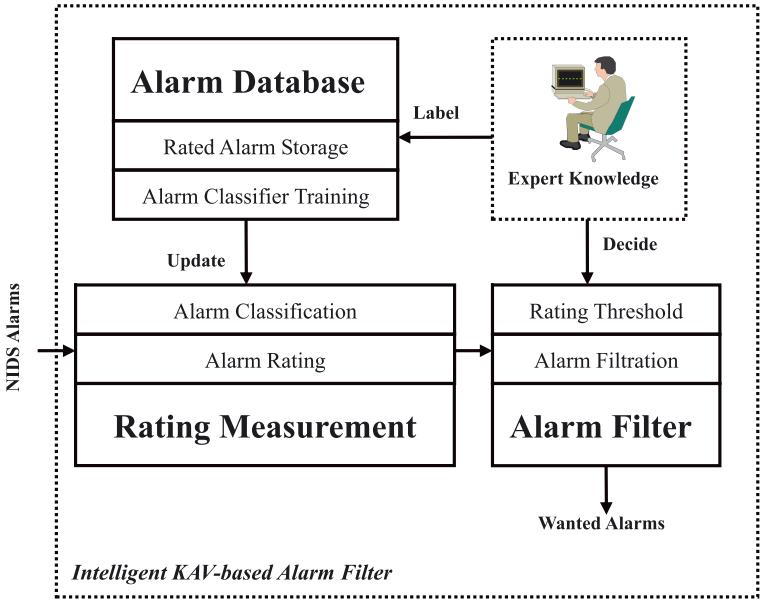

Figure 1. The high-level architecture of intelligent KAV-based alarm filter.

Table I. The rating values regarding classification and corresponding meanings.

Rate	(4.0, 5.0]	(3.0, 4.0]	(2.0, 3.0]	(1.0, 2.0]	(0, 1.0]
Meaning	Very critical	Critical	Important	Not important	Non-critical

In real deployment, there are two major phases in the filter: *preparation phase* and *filtration phase*.

- *Preparation phase*. In this phase, a security expert or administrator can label a limited number of NIDS alarms (e.g., 100 rated alarms) and store them into the component of *alarm database*. Then the alarm filter can train the KNN classifier to establish a KNN model based on these labeled instances. In addition, an expert should decide the *rating threshold* in the component of *alarm filter*.
- *Filtration phase*. In this phase, incoming alarms would first arrive at the component of *rating measurement*. Then this component would label each incoming NIDS alarm by means of a KNN classifier. Then, these labeled alarms will be forwarded to the component of *alarm filter*, which is responsible for conducting alarm filtration according to the predecided *rating threshold*. For example, if an alarm is rated below the *rating threshold*, then this alarm will be filtered out as an unwanted alarm.

3.2. The component of alarm database

This component is mainly responsible for storing the labeled NIDS alarms and utilizing these alarms to train the KNN classifier. Specifically, it is composed of two parts: *rated alarm storage* and *alarm classifier training*. In the part of *alarm classifier training*, the pretrained KNN classifier is used to classify incoming NIDS alarms. The selection of this classifier is based on the following reasons:

- The *KNN* algorithm is a method of classifying objects based on the number (K) of closest training examples in the feature space. That is, a target object can be classified in terms of its distances to the nearest cluster. Previous work like Law and Kwok [31] has shown that this classifier is good at classifying incoming alarms in intrusion detection.
- In our previous work [34], we found that this classifier could achieve a higher filtration rate and a higher classification accuracy (CA) as compared with other algorithms like *decision tree*. In addition, we also found that this classifier could achieve a fast speed in the phase of training and classification, which is a desirable property when deploying in a resource-limited platform (e.g., a mobile phone and an agent-based network).

In real deployment, an expert or security administrator can label a number of NIDS alarms in advance and store them in the *alarm database*. Then, the component of *alarm database* can train the KNN classifier and build a KNN classification model. The trained classifier later can be used in the component of *rating measurement* for labeling incoming alarms. Moreover, adding new labeled alarms, *alarm database* can keep updating the KNN model periodically.

3.3. The component of rating measurement

This component is responsible for classifying incoming alarms into proper clusters based on the trained KNN classifier and then labeling these alarms. As shown in Figure 1, it consists of two parts: *alarm classification* (i.e., classifying alarms into clusters) and *alarm rating* (i.e., labeling clustered alarms). The rating mechanism is a key element in our designed alarm filter, and we show the rating classification and corresponding meanings in Table I. In this work, there are totally five classes of labels: *rate (4.0, 5.0]*, *rate (3.0, 4.0]*, *rate (2.0, 3.0]*, *rate (1.0, 2.0]*, and *rate (0, 1.0]*.

The rating value of 5 is the highest score in the mechanism that means that this kind of alarms is *very critical* to a security administrator, while the rating value of 0 is the lowest score. Generally, a higher score indicates that an alarm is more important. As described in Table I, different labels indicate various levels of importance for an alarm such as *very critical*, *critical*, *important*, *not important*, and *non-critical*.

In this work, we set the numerical accuracy for expert rating to 0.1, so that each category has a range. For example, for a *very critical* alarm, its rating value can be ranged from 4.0 to 5.0. For an *important* alarm, its rating value can be ranged from 2.0 to 3.0, and for a *non-critical* alarm, its rating value can be ranged from 0 to 1.0. In this work, we introduce how to rate alarms in terms of expert knowledge in Section 3.5.

To better illustrate the alarm classification using the KNN classifier, we give a simple example in Figure 2. The white point is an incoming NIDS alarm waiting for classification. The black points are rated alarms and are gathered into three clusters rated as *critical*, *important*, and *not important*. To classify the white point, the KNN classifier would calculate the Euclidean distance (e.g., D1, D2, and D3) between the white point and the other three clusters, respectively. The distance can represent the similarity between the white point and the clusters. The shorter the distance, the more similar they are. The calculation of the Euclidean distance can be described as follows:

$$[Distance\,(P1, P2)]^2 = \sum_{0}^{N}(P1_i - P2_i)^2 \qquad (1)$$

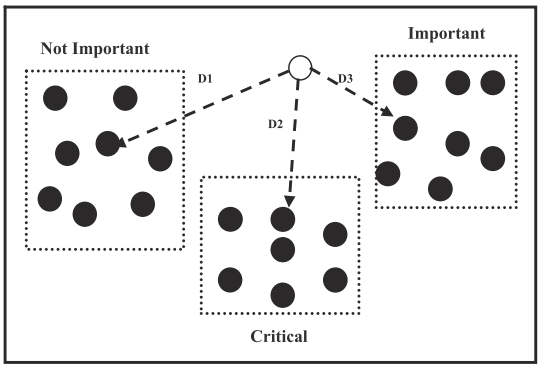

Figure 2. A case of classifying alarms using KNN classifier in the component of rating measurement. The white point is an alarm waiting for classification, while the black points are rated alarms and are gathered into three clusters.

$P1_i$ and $P2_i$ are the values of the ith attribute of points $P1$ and $P2$, respectively. As there are five classes (from *non-critical* to *very critical*), the classification problem in this work is actually a multi-class problem. Formally, let X denote the domain of instances and Y be the finite set of labels. Given a training set $T = \{(x1, y1), (x2, y2), \ldots, (xm, ym)\,(xi \in X,\ yi \in Y)\}$, the goal of the learning system is to output a multi-class classifier $h : X(x_i) \rightarrow Y(y_i)$. The designed multi-class KNN algorithm can be described as follows:

(1) For a new point p, calculating its Euclidean distance from other points;
(2) identifying the nearest N point, until there are K points that belong to the same class c; and
(3) classifying the new point to the class c.

The multi-class problem is simpler than the multi-label problem, where we can still use traditional metrics such as accuracy, precision, recall, and F-measure [37]. Thus, classifying an alarm into one cluster can rely on the closest K nearest points (an evaluation of K can be seen in Section 4). If an alarm is classified into a specific cluster, then this alarm would be given the same label as that cluster. For instance, if an alarm is classified to a cluster labeled as *important*, then this alarm will also be labeled as *important*.

3.4. The component of alarm filter

This component is used to filter out unwanted NIDS alarms according to the pre-decided *rating threshold*. As shown in Figure 1, it includes two parts: *rating threshold* and *alarm filtration*. In the part of *rating threshold*, the rating threshold would be decided by expert knowledge, while in the part of *alarm filtration*, the filtration procedure can be described as follows:

- If the rating value of an alarm is smaller than the *rating threshold*, then this alarm will be filtered out as unwanted alarm.
- If the rating value of an alarm is higher than the *rating threshold*, then this alarm will be output to alert the security administrator.

Discussions. As mentioned earlier, an expert or security administrator can pre-decide the *rating threshold* in the *preparation phase*, by considering the specific network information such as the network structure and network deployment. It is noted that expert knowledge is a key factor in affecting the filtration performance of the alarm filter. For example, Pietraszek [30] has shown that a security administrator is critical to determine which alarms are useful for them during their analysis. That is, the use of expert knowledge is important in alarm reduction. In this work, we use a simple but efficient method to extract expert knowledge, and the details will be discussed next.

3.5. Extraction of expert knowledge

In this work, we employ a simple approach of knowledge extraction by computing *average rating value* for each alarm. That is, for an alarm AL, we require N experts to give a rating value R_i $(i = 1, 2, \ldots)$, respectively; then the *average rating value* (ARV_{AL}) for this alarm can be calculated as follows:

$$ARV_{AL} = \frac{\sum_1^i R_i}{N} \tag{2}$$

Invalid rating. As experts may have different backgrounds, we notice that the rating may be very different sometimes. To further improve the accuracy of classifying an alarm, we define an invalid rating as follows:

$$|ARV_{AL} - R_i| > 0.9 \ (i = 1, 2, \ldots) \tag{3}$$

In Equation (3), $|ARV_{AL} - R_i|$ means the *absolute difference* between the *average rating value* and each rating value R_i $(i = 1, 2, \ldots)$. If the *absolute difference* is bigger than 0.9, then we regard this rating is invalid for this alarm because the disagreement of rating is larger than a *classification category*. If an invalid rating is identified, we then require experts to rate this alarm again. In real cases, we believe that invalid rating should be finally determined by a security administrator who takes charge of the specific network environment.

3.6. Modes of the alarm filter

In this work, the designed alarm filter can work under two modes: *basic mode* and *advanced mode*.

- *Basic mode*. In this mode, the alarm filter would conduct alarm reduction by means of the KNN classifier

with *one-time training*. That is, the alarm filter works without new labeled alarms.

- *Advanced mode*. In this mode, the alarm filter would reduce unwanted alarms with *continuous training*, which means that some new labeled alarms can be added and retrain the classifier. To reduce workload and improve efficiency, the designed alarm filter can automatically output a number of instances for labeling if given a pool of unlabeled data, based on the KNN classifier.

In the *advanced mode*, we design a decision algorithm to decide which instances should be output for labeling as follows. Given a set of unlabeled data, the KNN classifier computes its Euclidean distance from other labeled data, among the nearest K' instances, and the classifier would output this instance if no label can be made using the majority voting. Taking $K' = 5$ as an example, if there are two critical alarms, two important alarms, and one critical alarm, then the KNN classifier cannot decide the label by means of majority voting. In this case, the KNN classifier would output this instance for expert labeling. In the evaluation, we find that this labeling mechanism can greatly improve the efficiency of labeling and leverage classification results.

Discussions. Intuitively, the alarm filter can perform well in the *basic mode* when the training data are good enough. In most cases, a large number of labeled data is required to train a good classifier. However, it is usually hard to obtain such a large number of good labeled instances. If labeled data are not enough during the training, the alarm filter can perform better in the *advanced mode* as new labeled data can generally improve the classification performance. In Section 5, we analyze the performance of the alarm filter under these two modes.

4. *k*-NEAREST-NEIGHBOR CLASSIFIER SETTINGS

As described in Section 3, our designed KNN algorithm is to classify an object based on a majority vote of its neighbors (i.e., identifying the nearest points until there are K points that belong to the same class). Thus, its performance is highly depending on the value of K. In this section, we aim to conduct a simulation on two Snort alarm datasets to explore and select an appropriate value K.

In this work, we use Snort [38] in the simulation and following evaluation, which is an open-source signature-based NIDS and is widely adopted in research and industry. It is capable of conducting real-time traffic analysis such as content searching and content matching, and detecting various attacks by comparing incoming packets with its own set of attack signatures [3]. To better illustrate its alarm format, we give three concrete examples as follows:

(A) 03/15-21:18:16.784772 ATTACK-RESPONSES Invalid URL [Classification: Attempted Information Leak] [Priority: 2] {TCP} 209.1.224.190:80 – > 172.16.112.149:5279.

(B) 03/15-21:20:58.731850 ICMP Echo Reply [Classification: Misc activity] [Priority: 3] {ICMP} 172.16.112.194 – > 197.182.91.233.

(C) 03/15-21:35:31.655765 CHAT IRC nick change [Classification: Potential Corporate Privacy Violation] [Priority: 1] {TCP} 172.16.113.84:14456 – > 192.168.1.20:6667.

The examples earlier are extracted from Snort alarm database. In (A), "03/15-21:18:16.784772" is the generation time of this alarm. "ATTACK-RESPONSES Invalid URL" is the *alarm description*, while "Attempted Information Leak" is the *alarm classification* defined by Snort. The *priority* of this alarm is 2. Note that a bigger number means a higher priority. "TCP" is the *packet type*. "209.1.224.190" is the *source IP address*, while "80" is the *source port number*. "172.16.112.149" is the *destination IP address*, while "5279" is the *destination port number*.

The formats of (B) and (C) are similar to (A). Note that there is no specific source port number and destination port number in (B) (which means that any port numbers are satisfied), and the target packet in (B) is Internet Control Message Protocol (ICMP). The priority of (B) is 3, while the priority of (C) is 1. Thus, based on the rules of the priority, the priority order of the previous three examples is (B)>(A)>(C).

Based on the structure of Snort alarms, in this work, we use eight features to represent them as *alarm description*, *alarm classification*, *priority*, *packet type*, *source IP address*, *source port number*, *destination IP address*, and *destination port number*. In the phase of training, all alarm features will be converted into *appearance possibility*. For example, a source IP address of "209.1.224.190" could be represented as 13/34, while a destination port number of "80" could be represented as 5/11 according to their appearance times in the dataset.

Simulation. In order to explore the effect of the value K, we construct two alarm datasets, which are obtained by replaying the DARPA1999 dataset[¶] to Snort. Because the first and third weeks of DARPA1999 dataset are free-attack traffic, any triggered alarms can be regarded as false-positive alarms. The true alarms can be triggered by conducting scanning attacks to Snort (the details of attack simulation can be referred to our previous work [34]). The basic information of true and false alarms can reduce the burden of experts during the process of labeling. All alarms will be stored in a pool, so we can construct two alarm datasets by randomly selecting alarms from the pool as follows.

- *DATASET A*: This dataset contains a total of 3184 alarms (by randomly selecting 2174 false alarms and 1010 true alarms). The number of false alarms is

[¶] Available at http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/ 1999data.html.

DOI: 10.1002/sec

Table II. The distribution of labeled instances in different datasets.

Data	Very critical	Critical	Important	Not important	Non-critical
DATASET A	167	276	567	1028	1146
DATASET B	243	432	453	101	582
Training dataset	10	13	22	15	40

Table III. The average classification accuracy and *F*-measure for DATASET A (the bold number means the best value for each column).

Average classification accuracy	Very critical	Critical	Important	Not important	Non-critical
$k = 3$	0.765	0.793	0.854	**0.901**	0.856
$k = 4$	0.810	0.842	0.872	0.884	0.881
$k = 5$	0.88	**0.932**	**0.885**	0.865	**0.905**
$k = 6$	0.862	0.887	0.764	0.830	0.885
$k = 7$	**0.893**	0.832	0.72	0.845	0.876
$k = 8$	0.752	0.804	0.741	0.823	0.842
Average F-measure					
$k = 3$	0.753	0.784	0.834	**0.889**	0.832
$k = 4$	0.796	0.833	0.868	0.876	0.874
$k = 5$	0.852	**0.918**	**0.864**	0.865	**0.893**
$k = 6$	0.824	0.876	0.764	0.817	0.876
$k = 7$	**0.863**	0.823	0.703	0.845	0.863
$k = 8$	0.738	0.798	0.741	0.801	0.839

larger than the number of true alarms (the ratio is about 2:1).

- *DATASET B*: This dataset contains a total of 1811 alarms (by randomly selecting 683 false alarms and 1128 true alarms). The number of true alarms is larger than the number of false alarms (the ratio is about 2:1).

We manually label all the items, and the labeling information of these two datasets are described in Table II. For a multi-class classifier, it is known that traditional metrics are still applicable. We denote true positive (TP) as the number of true positive instances predicted as positive, true negative (TN) as true negative instances predicted as negative, false positive (FP) as true positive instances predicted as negative, and false negative (FN) as true negative instances predicted as positive. Therefore, four metrics can be used to evaluate the performance: *CA*, *precision*, *recall*, and *F-measure*, which are defined as follows:

$$CA = \frac{TP + TN}{TP + TN + FN + FP} \tag{4}$$

$$Precision = \frac{TP}{TP + FPz} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$F - measure = \frac{2Precision \times Recall}{Precision + Recall} \tag{7}$$

The metric of *CA* is used to measure the capability of classifying both positive instances and negative instances; the higher, the better. The metric of *precision* is used to measure the proportion of predicted positive instances that are actually positive. A higher *precision* means lower false-positive rate. The metric of *recall* is to measure the proportion of actual positive instances that are correctly identified. The metric of *F-measure* is a harmonic mean between precision and recall. In general, the higher the value of *F-measure*, the better performance is achieved.

The designed KNN classifier was implemented by means of WEKA [39], which provides a collection of machine learning algorithms, with the purpose of avoiding any implementation bias. During the simulation, we use 100 labeled alarms to train the filter (the distribution of the training dataset can be seen in Table II) and test the classifier by gradually increasing the value of K from 3 to 8. The specific results regarding average *CA* and average *F-measure* using 10-fold cross validation are shown in Tables III and IV, respectively.

Results analysis. As shown in Table III, it is easily visible that for *DATASET A*, both the *CA* and *F-measure* can generally achieve a better value when $K = 5$. Either before or after this point, the performance of both metrics would gradually decrease. We thus consider that setting $K = 5$ is appropriate for *DATASET A*.

For *DATASET B* in Table IV, we can find similar results. When $K = 5$, the classifier can achieve an overall better performance of *CA* and *F-measure* as compared with other settings. In addition, *DATASET A* contains more false alarms than *DATASET B*, where the ratio of false alarms

Table IV. The average classification accuracy and *F*-measure for DATASET B (the bold number means the best value for each column).

Average classification accuracy	Very critical	Critical	Important	Not important	Non-critical
$k = 3$	0.768	0.801	0.842	0.832	0.836
$k = 4$	0.772	0.834	0.852	0.845	0.843
$k = 5$	0.788	**0.854**	**0.864**	**0.848**	**0.886**
$k = 6$	0.788	0.0.851	0.867	**0.848**	0.853
$k = 7$	0.782	0.0.831	0.853	0.832	0.864
$k = 8$	**0.791**	0.823	0.823	0.812	0.852
Average F-measure					
$k = 3$	0.732	0.792	0.821	0.823	0.822
$k = 4$	0.753	0.821	0.836	0.836	0.834
$k = 5$	0.762	**0.843**	**0.843**	**0.841**	**0.876**
$k = 6$	0.762	0.841	0.845	**0.841**	0.833
$k = 7$	0.756	0.817	0.832	0.831	0.843
$k = 8$	**0.777**	0.801	0.803	0.793	0.832

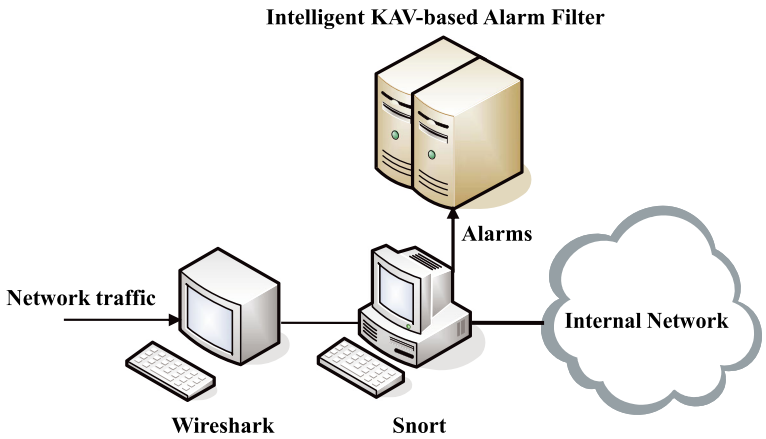

Figure 3. The deployment of network environment.

and true alarms is nearly 2:1. In real scenarios, the number of false alarms is usually bigger than that of true alarms [30]. Thus, based on the results regarding *DATASET A* and *DATASET B*, we consider that setting $K = 5$ is appropriate for the classifier in this work.

5. EVALUATION

In this section, we aim to evaluate the performance of the *intelligent KAV-based alarm filter* with a real dataset and in a network environment that is constructed by Snort and Wireshark [40]. The network deployment is described in Figure 3.

The Snort is deployed in front of the *Internal Network* aiming to detect various network attacks. The *intelligent KAV-based alarm filter* is deployed close to Snort in order to filter out Snort alarms. The Wireshark is implemented in front of Snort, which is responsible for recording network packets and providing statistical information.

In real-world applications, network traffic would pass through Wireshark and arrive at Snort. The Snort examines network packets and generates alarms. Later, all the Snort alarms generated would be forwarded into the *intelligent KAV-based alarm filter* for alarm filtration.

5.1. Experimental methodology

In the evaluation, we mainly conduct three experiments to investigate the performance of the *intelligent KAV-based alarm filter* as follows.

- *Experiment 1 with real dataset*. This experiment uses a real dataset to investigate the performance of the alarm filter as compared with several base classifiers like decision tree and neural networks. This dataset is extracted from a Honeypot project. In the preparation phase, we use 100 labeled Snort alarms to train the classifier.

Table V. Several examples of rated Snort alarms given by experts.

Alarm description	Rater 1	Rater 2	Rater 3	ARV_{AL}
WEB-MISC /doc/ access	4.0	3.8	4.5	4.1
SNMP public access udp	3.4	3.6	3.4	3.5
WEB-MISC apache DOS attempt	3.7	3.3	3.5	3.5
TELNET login incorrect	2.8	3.0	2.5	2.8
ICMP digital island bandwidth query	1.5	1.0	1.1	1.2
BAD-TRAFFIC IP Proto 103 PIM	1.5	1.2	2.2	1.6
FTP CWD	0.5	0.5	0.8	0.6

Table VI. The distribution and the number of rated alarms in the training.

Rate value	(4.1, 5.0]	(3.0, 4.0]	(2.0, 3.0]	(1.0, 2.0]	(0, 1.0]
The number of alarms	10	11	9	33	37

- *Experiment 2 in basic mode*. In this experiment, we implement the alarm filter in the deployed network environment for 1 month aiming to evaluate the performance of the alarm filter under the *basic mode*, in which *one-time training* is used. Additionally, the traffic captured in this experiment will be used in *experiment 3*.
- *Experiment 3 in advanced mode*. In this experiment, we deploy the alarm filter under *advanced mode* where the filter can be trained periodically. Then, we replay the captured traffic earlier to the alarm filter and evaluate the performance of the alarm filter under the *advanced mode*.

In the experiment, we require three experts to rate 100 Snort alarms as the training data. Two experts are from industry (IDS-related and Honeypot-related profession), and the other is from academia (computer science department and security-related major). All of them have over 5 years of experience in the area of intrusion detection.

In Table V, we provide several rating examples in the training. It is found that each expert may give distinct rating values for the same alarm based on their own knowledge and preference, so that *average rating value* can minimize the gap among these ratings. For example, the alarm of "BAD-TRAFFIC IP Proto 103 PIM" obtains three different values: 1.5 (not important), 1.2 (not important), and 2.2 (important), so its average rating value is 1.6 (not important). As two experts rated that this alarm is not important, the average rating value of 1.6 also shows that this alarm is not important. This decision process is known as majority rules.

The distribution of rated Snort alarms is described in Table VI. It is noted that this distribution is similar to the real scenario, in which the number of non-critical alarms is far higher than that of critical alarms [10]. In Table VI, it shows that the number of non-critical alarms is 37, the number of not important alarms is 33, while the numbers of very critical alarms and critical alarms are 10 and 11, respectively.

Table VII. The number of produced alarms for each day in the real dataset.

Day	No.
Day 1	1232
Day 2	873
Day 3	1832
Day 4	2991
Day 5	321
Day 6	3241
Day 7	431
Day 8	1282
Day 9	3391
Day 10	1982
Day 11	832
Day 12	2921
Day 13	1798
Day 14	3281
—	—

5.2. Experiment 1

In this experiment, a real alarm dataset is used to explore the performance of the designed alarm filter. This dataset comes from a Snort-included Honeypot project,[‖] which was deployed in our CSLab but with a public IP domain. The Honeypot can log all incoming traffic while Snort can produce alarms by analyzing these traffic. Based on this, we extracted an alarm dataset of 2 weeks. The distribution of alarms for each day is shown in Table VII.

To calculate the metrics, the real dataset was labeled by the same experts. In the evaluation, we compare the performance between decision tree (denoted as F1), neural networks (denoted as F2), and our designed KNN classifier (denoted as F3). To extend F1 and F2 to multi-class

[‖] This Honeypot project is established and managed by HoneybirdHK (http://www.honeybird.hk/).

Table VIII. The results of average classification accuracy for the real dataset (the bold number means the best value for each row).

Day	F1	F2	F3
Day 1	0.854	0.765	**0.876**
Day 2	0.821	0.801	**0.865**
Day 3	0.873	0.831	**0.901**
Day 4	0.832	0.811	**0.853**
Day 5	0.782	0.742	**0.824**
Day 6	0.812	0.804	**0.843**
Day 7	**0.845**	0.821	**0.845**
Day 8	0.762	0.732	**0.802**
Day 9	0.887	0.821	**0.889**
Day 10	0.901	0.832	**0.910**
Day 11	0.824	0.789	**0.832**
Day 12	0.862	0.783	**0.897**
Day 13	0.872	0.838	**0.881**
Day 14	0.904	0.831	**0.913**
—	—	—	—

Table IX. The results of average *F*-measure for the real dataset (the bold number means the best value for each row).

Day	F1	F2	F3
Day 1	0.774	0.623	**0.789**
Day 2	0.762	0.681	**0.783**
Day 3	0.802	0.732	**0.843**
Day 4	0.754	0.721	**0.803**
Day 5	0.732	0.701	**0.793**
Day 6	0.734	0.689	**0.797**
Day 7	0.789	0.731	**0.813**
Day 8	0.710	0.673	**0.756**
Day 9	0.823	0.746	**0.842**
Day 10	0.832	0.782	**0.872**
Day 11	0.778	0.712	**0.809**
Day 12	0.814	0.734	**0.854**
Day 13	0.834	0.782	**0.863**
Day 14	0.852	0.792	**0.892**
—	—	—	—

classification, these two classifiers are trained using one-versus-all strategy, where a single classifier is trained per class to distinguish that class from all other classes [41]. The experimental results of *average CA* and *average F-measure* are described in Tables VIII and IX, respectively.

The *average CA* and *average F-measure* are computed by considering all five classes (from *very critical* to *non-critical*) on average. It is visible that F3 can achieve a better average CA and average *F*-measure as compared with F1 and F2. The experimental results indicate that our designed KNN classifier can achieve a good performance in correctly labeling the alarms.

5.3. Experiment 2

In this experiment, we attempt to evaluate the performance of the alarm filter under the *basic mode*, namely

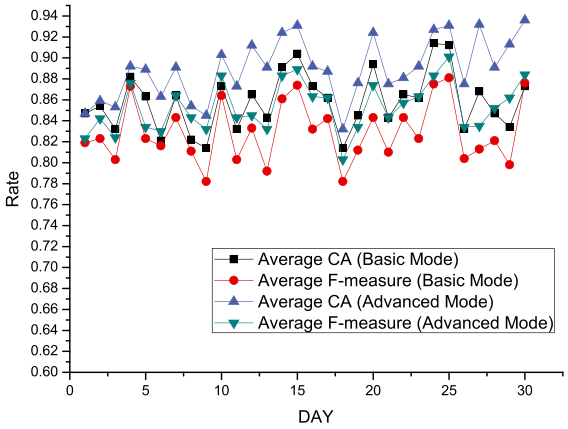

Figure 4. The results of average classification accuracy and average *F*-measure under basic mode and advanced mode.

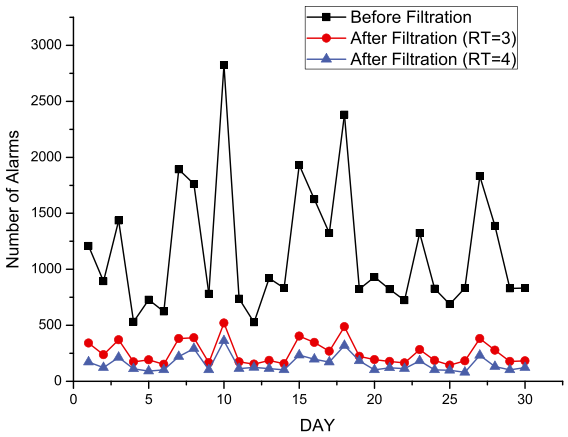

Figure 5. The results of alarm filtration for *experiment* 2.

one-time training, in the network environment. The evaluation was performed for 1 month, and the experimental results of *average CA* and *average F-measure* are presented in Figure 4.

Under the *basic mode*, Figure 4 shows that the average CA is generally above 0.81, while the average *F*-measure is above 0.78. In day 24 and day 25, our classifier can achieve a best *average CA* of 0.914 and *average F-measure* of 0.881, respectively. To reduce unwanted alarms, we choose two different rating thresholds ($RT = 3$ and $RT = 4$) and present the filtration rate in Figure 5.

Intuitively, the filtration rate would vary with different rating thresholds. It is noted that the alarms with the labels of critical alarms and very critical alarms would be more useful for a security administrator. Therefore, we set the rating threshold to 3 (*critical*) and 4 (*very critical*), respectively. The filtration results are analyzed as follows:

- *Rating threshold of* 3 (*label of critical*). For this threshold, Figure 5 shows that the filtration rate is

in the range from 67% to 80%. The average rate is about 74%.

- *Rating threshold of* 4 (*label of very critical*). For this threshold, the filtration rate ranged from 75% to 90%, while the average rate is nearly 83%.

It is visible that the specific filtration rate may be different each day and the rate will increase when adjusting the threshold from 3 to 4. We consider that the filtration rate achieved in this experiment is very encouraging for real-world applications. For instance, for *day* 26 under RT = 4, only 80 alarms would be output where the filtration rate is 90.4%.

Overall, these experimental results indicate that the designed *intelligent KAV-based alarm filter* is promising in reducing NIDS alarms with a promising CA and filtration rate under the *basic mode*.

5.4. Experiment 3

In this experiment, we aim to explore the performance of the designed alarm filter under the *advanced mode*, where the *continuous training* is applied. That is, at the end of each day, we retrained the KNN-based classifier with some new labeled Snort alarms. In this work, we add 20 new labeled alarms each day during the process of retraining.

In order to reduce the workload of selecting instances and improve the efficiency of labeling, our classifier in the *advanced mode* can automatically output a set of instances for labeling if given a pool of unlabeled data. The decision algorithm is introduced in Section 3.6, and in this work, we set $K' = 5$. The results of *average CA* and *average F-measure* can be referred to Figure 4. It is found that by means of *continuous training*, the average CA and average F-measure can be both increased in the range from 3% to 7%.

Effect of our labeling. To verify the effect of the used decision algorithm, we compare the performance with the situation of using random selection, where we randomly select 20 alarms for labeling. The comparison results are shown in Figure 6. It is visible that our algorithm can achieve a better result than random selection.

Similarly, we also set the rating threshold to 3 (*critical*) and 4 (*very critical*), respectively. The filtration results are described in Figure 7.

- *Rating threshold of* 3 (*label of critical*). For this threshold, Figure 7 shows that the filtration rate is in the range from 77% to 88% and that the average rate is about 82%.
- *Rating threshold of* 4 (*label of very critical*). For this threshold, it is seen that the filtration rate ranges from 87% to 93%, while the average rate can reach nearly 89%.

Compared with the results obtained in *experiment* 2, these experimental results show that by means of *continuous training*, our designed alarm filter can achieve a better performance regarding the filtration rate and CA under the *advanced mode*.

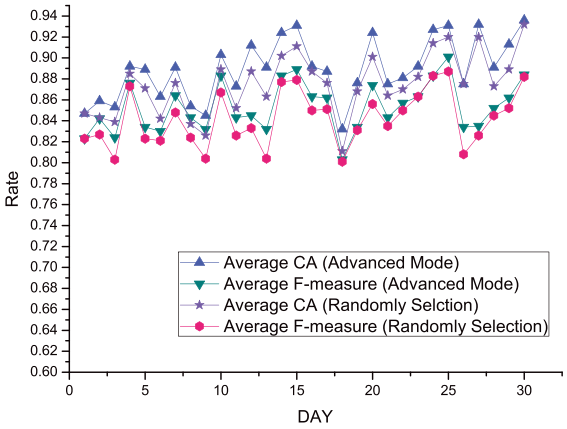

Figure 6. Comparison between the *advanced mode* and the random selection.

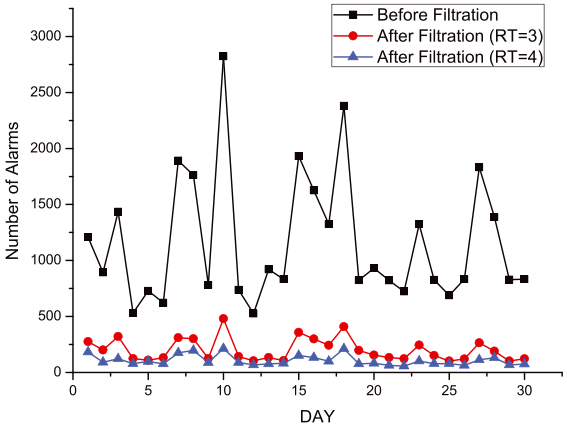

Figure 7. The results of alarm filtration for *experiment* 3.

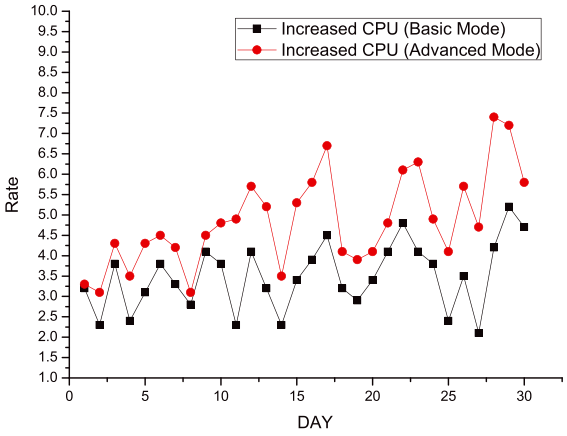

Figure 8. The average increased CPU workload by the alarm filter.

5.5. CPU workload

The designed alarm filter can provide alarm reduction under two modes. In the previous experiments, we deploy the alarm filter on a Windows platform with Intel(R) Core(TM)2 Duo CPU, processor 2.8 GHz and 4 GB of RAM. The results of the average CPU workload caused by the filter are described in Figure 8.

It is visible that under the *basic mode*, the increased CPU ranges from 2% to 5%. By contrast, the increased CPU workload is in the range from 3% to 7.5% under the *advanced mode*. The higher workload under the *advanced mode* is caused by that the alarm filter is required to retrain the classifier periodically. In this case, we consider that the deployment of the alarm filter would not bring much burden on computers in real settings.

6. CONCLUSION

Network intrusion detection systems are widely deployed in current network environments to detect various network attacks. However, a big suffering problem is that a large number of alarms, especially unwanted alarms, could be generated during the detection.

To address this issue, in this paper, we develop a method of KAV and design an intelligent alarm filter (named *intelligent KAV-based alarm filter*) to filter out unwanted NIDS alarms. In particular, the alarm filter consists of three major components: *alarm database*, *rating measurement*, and *alarm filter*. The expert knowledge is a key element, which aims to rate alarms and decide rating threshold. We further design a multi-class KNN classifier and conduct a simulation to explore the effect of value K. We find that $K = 5$ is appropriate for this work.

In the evaluation, we conduct three experiments to investigate the performance of the alarm filter with a real dataset and in a network environment (under two modes), respectively. The experimental results indicate that the alarm filter can achieve good filtration results with affordable CPU usage and reduce the workload of a security analyst by effectively filtering out many unwanted alarms. Future work could include investigating how to accurately and efficiently extract expert knowledge in a more systematic way.

ACKNOWLEDGEMENTS

We would like to thank HoneybirdHK for providing the real dataset and expert knowledge. The authors were fully funded by the Innovation to Realization Funding Scheme of the City University of Hong Kong (under the project number 6351018).

REFERENCES

1. Symantec. Internet security threat report (vol. 17), 2012. Available: http://www.symantec.com/business/threatreport/index.jsp [Accessed on November 2014].
2. Scarfone K, Mell P. Guide to intrusion detection and prevention systems (IDPS), NIST Special Publication, 2007; 800–894.
3. Roesch M. Snort: lightweight intrusion detection for networks, In *Proceedings of the 13th Large Installation System Administration Conference (LISA)*, Seattle, WA, 1999; 229–238.
4. Vigna G, Kemmerer RA. NetSTAT: a network-based intrusion detection approach, In *Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC)*, Scottsdale, AZ, USA, 1998; 25–34.
5. Meng Y, Li W, Kwok LF. Intelligent alarm filter using knowledge-based alert verification in network intrusion detection, In *Proceedings of the 20th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, Macau, China, 2012; 115–124.
6. Ghosh AK, Wanken J, Charron F. Detecting anomalous and unknown intrusions against programs, In *Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC)*, Scottsdale, AZ, USA, 1998; 259–267.
7. Paxson V. Bro: a system for detecting network intruders in real-time. *Computer Networks* 1999; **31**(23-24): 2435–2463.
8. Valdes A, Anderson D. Statistical methods for computer usage anomaly detection using NIDES. *Technical Report*, SRI International, 1995.
9. Meng Y, Kwok LF. Adaptive non-critical alarm reduction using hash-based contextual signatures in intrusion detection. *Computer Communications* 2014; **38**: 50–59.
10. Lippmann RP, Fried DJ, Graf I, Haines JW, Kendall KR, McClung D, Weber D, Webster SE, Wyschogrod D, Cunningham RK, Zissman MA. Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation, In *Proceedings of DARPA Information Survivability Conference and Exposition*, Hilton Head, South Carolina, 2000; 12–26.
11. McHugh J. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln laboratory. *ACM Transactions on Information and System Security* 2000; **3**(4): 262–294.
12. Meng Y, Li W. Constructing context-based non-critical alarm filter in intrusion detection, In *Proceedings of the 7th International Conference on Internet Monitoring and Protection (ICIMP)*, Stuttgart, Germany, 2012; 75–81.
13. Ptacek TH, Newsham TN. Insertion, evasion, and denial of service: eluding network intrusion detection. *Technical Report, Secure Networks*, 1998.
14. Sommer R, Paxson V. Outside the closed world: on using machine learning for network intrusion detec-

tion, In *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, California, 2010; 305–316.
15. Axelsson S. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security* 2000; **3**(3): 186–205.
16. Meng Y, Kwok LF. Adaptive blacklist-based packet filter with a statistic-based approach in network intrusion detection. *Journal of Network and Computer Applications* 2014; **39**: 83–92.
17. Geramiraz F, Memaripour AS, Abbaspour M. Adaptive anomaly-based intrusion detection system using fuzzy controller. *International Journal of Network Security* 2012; **14**(6): 352–361.
18. Goel R, Sardana A, Joshi RC. Parallel misuse and anomaly detection model. *International Journal of Network Security* 2012; **14**(4): 211–222.
19. Hubballi N, Biswas S, Nandi S. Towards reducing false alarms in network intrusion detection systems with data summarization technique. *Security and Communication Networks* 2013; **6**(3): 275–285.
20. Jamdagni A, Tan Z, He X, Nanda P, Liu RP. RePIDS: a multi tier real-time payload-based intrusion detection system. *Computer Networks* 2013; **57**(3): 811–824.
21. Tabia K, Leray P. Alert correlation: severe attack prediction and controlling false alarm rate tradeoffs. *Intelligent Data Analysis* 2011; **15**(6): 955–978.
22. Mohammadi M, Raahemi B, Akbari A, Nassersharif B. New class-dependent feature transformation for intrusion detection systems. *Security and Communication Networks* 2012; **5**(12): 1296–1311.
23. Kruegel C, Robertson W. Alert verification: determining the success of intrusion attempts, In *Proceedings of Workshop on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, Dortmund, Germany, 2004; 25–38.
24. Zhou J, Carlson AJ, Bishop M. Verify results of network intrusion alerts using light-weight protocol analysis, In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, Tucson, AZ, USA, 2005; 117–126.
25. Mu C, Huang H, Tian S. Intrusion detection alert verification based on multi-level fuzzy comprehensive evaluation, In *Proceedings of International Conference on Computational Intelligence and Security (CIS)*, Xi'an, China, 2005; 9–16.
26. Bolzoni D, Crispo B, Etalle S. ATLANTIDES: an architecture for alert verification in network intrusion detection systems, In *Proceedings of the 21st Large Installation System Administration Conference (LISA)*, Dallas, Texas, USA, 2007; 141–152.
27. Nguyen TH, Luo J, Njogu HW. An efficient approach to reduce alerts generated by multiple IDS products. *International Journal of Network Management* 2014; **24**(3): 153–180.
28. Njogu HW, Luo J, Kiere JN. Network specific vulnerability based alert reduction approach. *Security and Communication Networks* 2013; **6**(1): 15–27.
29. Njogu HW, Luo J, Kiere JN, Hanyurwimfura D. A comprehensive vulnerability based alert management approach for large networks. *Future Generation Computer Systems* 2013; **29**(1): 27–45.
30. Pietraszek T. Using adaptive alert classification to reduce false positives in intrusion detection, In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sophia Antipolis, France, 2004; 102–124.
31. Law KH, Kwok LF. IDS false alarm filtering using KNN classifier, In *Proceedings of the 5th International Workshop on Information Security Applications (WISA)*, Jeju Island, Korea, 2005; 114–121.
32. Alharbt A, Imai H. IDS false alarm reduction using continuous and discontinuous patterns, In *Proceedings of the 3rd International Conference on Applied Cryptography and Network Security (ACNS)*, New York, NY, USA, 2005; 192–205.
33. Chiu CY, Lee YJ, Chang CC, Luo WY, Huang HC. Semi-supervised learning for false alarm reduction, In *Proceedings of the 10th Industrial Conference on Advances in Data Mining (ICDM)*, Sydney, Australia, 2010; 595–605.
34. Meng Y, Kwok LF. Adaptive false alarm filter using machine learning in intrusion detection, In *Proceedings of the 6th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, Shanghai, China, 2011; 573–584.
35. Mao CH, Lee HM, Parikh D, Chen T, Huang SY. Semi-supervised co-training and active learning based approach for multi-view intrusion detection, In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC)*, Hawaii, USA, 2009; 2042–2048.
36. Meng Y, Kwok LF. Enhancing false alarm reduction using pool-based active learning in network intrusion detection, In *Proceedings of the 9th Information Security Practice and Experience Conference (ISPEC)*, Lanzhou, China, 2013; 1–16.
37. Zhang ML, Zhou ZH. ML-KNN: a lazy learning approach to multi-label learning. *Pattern Recognition* 2007; **40**: 2038–2048.
38. Snort. The open source network intrusion detection system. Available: http://www.snort.org/ [Accessed on May 2014].
39. WEKA: data mining software in Java. Available at: http://www.cs.waikato.ac.nz/ml/weka/ [Accessed on March 2014].
40. Wireshark. Network protocol analyzer. Available: http://www.wireshark.org [Accessed on July 2014].
41. Aly M. Survey on multiclass classification methods. *Technical Report*, 2005; 1–9.