Information Sciences 231 (2013) 4–14

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Toward a more practical unsupervised anomaly detection system

Jungsuk Song [a,d,*], Hiroki Takakura [b], Yasuo Okabe [c], Koji Nakao [a]

[a] National Institute of Information and Communications Technology, 4-2-1, Nukui-Kitamachi, Koganei-shi, Tokyo 184-8795, Japan
[b] Information Technology Center, Nagoya University, Furo-cho, Chikusa-ku, Nagoya 464-8601, Japan
[c] Academic Center for Computing and Media Studies, Kyoto University, Sakyo-ku, Kyoto 606-8501, Japan
[d] Korea Institute of Science and Technology Information, 245 Daehangno, Yuseong, Daejeon, 305-806, Korea

ARTICLE INFO

Article history:
Available online 22 August 2011

Keywords:
Intrusion Detection System
Clustering
One-class SVM
Anomaly detection

ABSTRACT

During the last decade, various machine learning and data mining techniques have been applied to Intrusion Detection Systems (IDSs) which have played an important role in defending critical computer systems and networks from cyber attacks. Unsupervised anomaly detection techniques have received a particularly great amount of attention because they enable construction of intrusion detection models without using labeled training data (*i.e.*, with instances preclassified as being or not being an attack) in an automated manner and offer intrinsic ability to detect unknown attacks; *i.e.*, 0-day attacks. Despite the advantages, it is still not easy to deploy them into a real network environment because they require several parameters during their building process, and thus IDS operators and managers suffer from tuning and optimizing the required parameters based on changes of their network characteristics. In this paper, we propose a new anomaly detection method by which we can automatically tune and optimize the values of parameters without predefining them. We evaluated the proposed method over real traffic data obtained from Kyoto University honeypots. The experimental results show that the performance of the proposed method is superior to that of the previous one.

1. Introduction

With the rapid development of internet infrastructure and local networks, security threats such as distributed denial of service (DDoS), computer viruses and internet worms, are simultaneously growing for computer systems and networks. Many efforts have been taken to counteract these cyber attacks in the last decade; including cryptography, firewalls and Intrusion Detection Systems (IDSs). Among these techniques, IDS [6,2] is becoming increasingly significant in maintaining proper network security and defending crucial computer systems and networks from malicious attacks [1].

There are mainly two types of Intrusion Detection Methods: misuse detection and anomaly detection. In misuse detection-based IDSs, network traffic is monitored to detect illegal access using predefined attack signatures [13,10]. If it is observed that any intrusion or suspicious activities match patterns of predefined signatures, they raise the corresponding alerts. On the other hand, IDSs based on the anomaly detection method use normal patterns to detect abnormal activities from observed data. They typically attempt to identify deviations from predefined normal patterns, and regard them as potential attacks. The former can detect well-known attacks at relatively higher accuracy than the latter, but they have a fatal weakness in that they cannot detect unknown attacks (*i.e.*, 0-day attacks) that are not matched to any predefined signatures. Anomaly detection-based IDSs can potentially detect unforeseen attacks since their activities differ from normal patterns.

* Corresponding author. Address: National Institute of Information and Communications Technology, 4-2-1, Nukui-Kitamachi, Koganei-shi, Tokyo 184-8795, Japan. Tel.: +82 42 869 0729; fax: +82 42 869 1119.
E-mail addresses: song@kisti.re.kr (J. Song), takakura@itc.nagoya-u.ac.jp (H. Takakura), okabe@i.kyoto-u.ac.jp (Y. Okabe), ko-nakao@nict.go.jp (K. Nakao).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.08.011

During the last decade, many machine learning and data mining techniques have been applied to IDSs to improve their performance and construct them with low cost and effort. In particular, *unsupervised* anomaly detection techniques [18,7,8,16,15,22,20,31] have received a great deal of attention because they can construct intrusion detection models without using labeled training data (*i.e.*, with instances preclassified as being or not being an attack) in an automated manner. With their intrinsic ability to detect 0-day attacks, this advantage makes them more easily applied to real environments since labeled data or purely normal data cannot be readily obtained in practice.

Instead of using unlabeled training data, they require several parameters (*e.g.*, the ratio of attack data to normal data and a threshold to determine attack data and normal data) during their building process. Since normal patterns on distinct networks differ from each other, this drawback leads to efficiency deterioration when we deploy them into diverse network environments. In other words, IDS operators and managers must carefully tune and optimize parameters required for building intrusion detection models based on changes of their network characteristics. Considering the generality of misuse detection-based IDSs where attack signatures can be used for any IDSs operating on distinct networks, we need to cope with this fatal weakness in unsupervised anomaly detection techniques.

In previous research [20], we have proposed an unsupervised anomaly detection technique that outperforms those in existing research. However, it needs three parameters (α, β and k) for constructing an intrusion detection model as described in Section 2. Among the three parameters, we focus on only two α and k, because we have proven that parameter β barely affects the performance of the previous anomaly detection method as long as we choose a reasonable range of value in [20]. In this paper, we propose a new anomaly detection method by which we can automatically obtain the optimize values of two parameters without predefining them. We evaluated the proposed method over a real dataset of network connections gathered from Kyoto University honeypots [21]. The experimental results show that the proposed method is superior to the previous one.

The rest of the paper is organized as follows. Section 2 gives a brief description of the previous anomaly detection method. Section 3 describes the proposed anomaly detection model in detail, and Section 4 gives the experimental results including their analysis. Section 5 presents concluding remarks and suggestions for future study.

2. Previous anomaly detection method

Fig. 1 shows the training and testing phases of the previous anomaly detection method. The training phase is composed of three main steps: filtering, clustering and modeling. The training phase first filters out most of the attack data from the original training data (①Filtering) to guarantee that the majority of each cluster obtained from the following clustering process consists of normal data, and partitions the filtered training data into k clusters which indicate normal patterns (②Clustering). Clustering is carried out due to the existence of different types of normal patterns in the traffic data such as HTTP, SMTP and FTP. Thus, it is very important to cluster the traffic data into individual clusters to more accurately identify their activities. One-class SVM [16,4] is then applied to each normal cluster (③Modeling), and k SVM models (*i.e.*, k hyperspheres) are

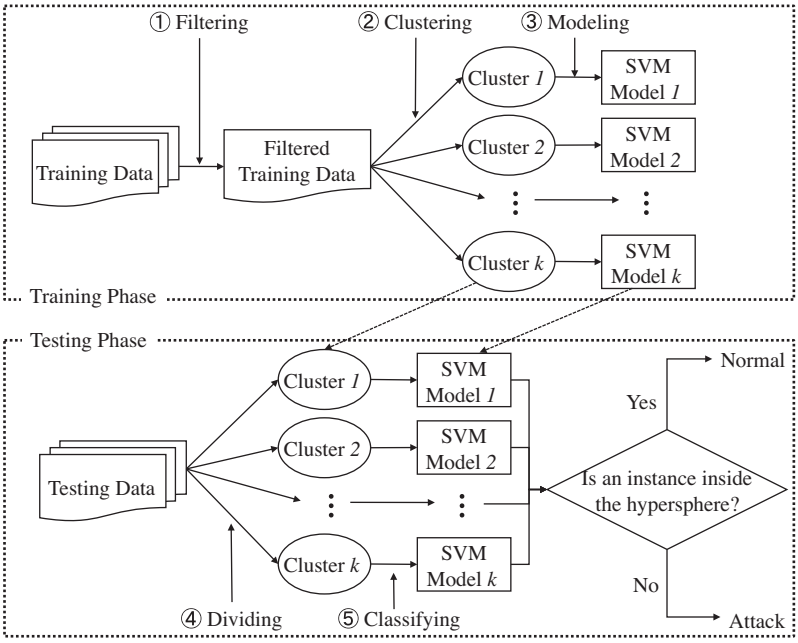

Fig. 1. Training and testing phases.

consequently obtained, where in the insides are regarded as normal areas. Note that one-class SVM is one of the typical unsupervised learning methods and is able to find a hypersphere in which most of the data points fall. Therefore, all data points inside the hypersphere can be regarded as the normal network activities in this approach, because the majority of each cluster is normal data.

In the testing phase, the distance is first computed between each data instance within the testing data and centers of k clusters created by the training phase, and each data instance is assigned to the closest cluster (④Dividing). The data instances assigned to the closest cluster are then input to the corresponding SVM model which is constructed from the training phase (⑤Classifying). If a data instance is inside the normal area of the corresponding SVM model, that instance is regarded as normal; otherwise, it is regarded as an attack.

The performance of this approach heavily depends on two processes, the filtering and clustering, because the insides of all hyperspheres discovered from the modeling process are regarded as normal areas, and are utilized during the testing phase. In other words, it must guarantee that not only the majority of each cluster consists of normal data, but also that each cluster indicates a distinct normal pattern which actually exists within the training data. To this end, the number of clusters (*i.e.*, k) to be created which are used in the clustering process is first set. Most of the attack data from the training data are then filtered to satisfy the requirement that each k cluster contains few attack data. In this filtering process, α which indicates the ratio of attack data to the total number of the training data, plays a significant role as a parameter in removing attack data. However, there is a fatal limitation in that, since the original training data has no label information, we cannot obtain the desired information about the number of attack data (*i.e.*, α) and number of normal patterns (*i.e.*, k) within the training data. As a result, the mis-setting of α and k leads to performance deterioration. Therefore, this paper presents an automatic configuration method for the two parameters, α and k. Note that the other parameter β is still adopted in the proposed method, and its role is described in Section 3.1.

3. Proposed method

The training and testing phases of the proposed method are basically the same as those shown in Fig. 1. Thus, we concentrate on the description of the filtering and clustering processes.

3.1. Filtering process

3.1.1. Motivation and comparison

In [20], the filtering process is based on the assumption that if a data instance is normal, there are many data instances with similar attribute values as the data instance; otherwise, there are few. Thus, it first separates the attribute values of all data instances into two regions: sparse and dense where attribute values belonging to the former are more frequent than those of the latter. It then partitions the training data into two groups: sparse and dense. If a data instance has at least one dimension which belonging to the sparse region (*i.e.*, attribute values with a low frequency), it is classified into the sparse group; otherwise it becomes a member of the dense group. The dense group is resultantly regarded as the filtering training data.

Two parameters, α and β, are required during this filtering process. For example, assume that we set α and β to 1 and 5, and 50 data instances of two-dimensional vectors are given as shown in Fig. 2, where 44 data instances are members of two normal clusters (*i.e.*, 1 and 2) and six data instances (marked A, B and C) are attacks. In this case, the process for separating the training data into the dense and sparse groups is as follows.

1. Divide the interval between the maximum and minimum value of each dimension into β(=5) equi-length *blocks*: $X_1 \sim X_5$ and $Y_1 \sim Y_5$.

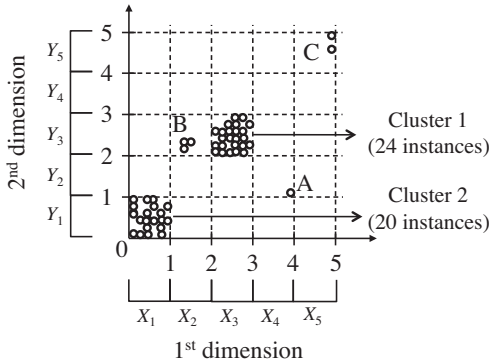

Fig. 2. Example of the filtering process.

2. Count the number of data instances belonging to each *block*: [$X_1 = 20$, $X_2 = 3$, $X_3 = 24$, $X_4 = 1$, $X_5 = 2$], [$Y_1 = 20$, $Y_2 = 1$, $Y_3 = 27$, $Y_4 = 0$, $Y_5 = 2$].
3. Rearrange five *block*s of each dimension in ascending order: [$X_4 = 1$, $X_5 = 2$, $X_2 = 3$, $X_1 = 20$, $X_3 = 24$], [$Y_4 = 0$, $Y_2 = 1$, $Y_5 = 2$, $Y_1 = 20$, $Y_3 = 27$].
4. Starting from the head *block*, sum the number of data instances within each *block*, and find the first *block* where the summation exceeds α% of the training data; X_5 and Y_5 become two *block*s found by this step, because α% (=1%) of the training data (*i.e.*, 50 instances) is 2.
5. Regard all *block*s located before two *block*s found from step 4 as the sparse region, and the others as the dense region: X_4, Y_4 and Y_2 are the sparse region, and the other seven *block*s are the dense region.
6. Regard data instances as members of the sparse group, if they have at least one attribute value which belongs to the sparse region. Only one data instance (marked A) becomes the member of the sparse group, because its attribute values fall in the sparse region, *i.e.*, X_4 and Y_2.
7. Regard data instances as members of the dense group if their all attribute values belong to the dense region; the other 49 data instances become the members of the dense group, *i.e.*, the filtered training data.

In many cases, however, the ratio of attack data to normal data is different for each dimension and it is very difficult to determine its exact value, because we cannot predict the amount attack data included in the training data. We, therefore, propose a new method to automatically adjust the value of α according to each dimension's distribution. The proposed method is based on the assumption there is a large gap between normal *block*s and attack *block*s in terms of the number of data instances belonging to them. In order to apply this property of the traffic data to the proposed method, we replace steps 4 and 5 of the above example with the following five steps.

- For each dimension, find the first *block* where the number of its members is non-zero: X_4 and Y_2.
- Starting from the two *block*s (X_4 and Y_2), calculate the ratio between two adjacent *block*s in each dimension and their average values. In this example, the average values of the 1st and 2nd dimensions are 2.84 and 4.45, respectively.
- For each dimension, starting from the two *block*s (X_4 and Y_2), find the first *block* where the ratio between two adjacent *block*s exceeds the average value: X_2 and Y_5 become two *block*s found by this step because their respective ratios are 20/3 = 6.66 and 20/2 = 10.
- If there is no such *block* we find a *block* where the ratio is the maximum.
- Regard all *block*s located before X_2 and Y_5 *block*s including themselves as the sparse region, and the others as the dense region: X_4, X_5, X_2, Y_4, Y_2 and Y_5 are the sparse region, and the other four *block*s are the dense region.

As a result, six data instances (marked A, B and C) which are not members of two normal clusters are correctly regarded as the sparse group, and the other 44 data instances are regarded as the dense group.

3.1.2. Formal description

Here we examine the proposed method in greater detail. The filtering process begins by defining $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, which denotes the set of data instances (*i.e.*, training data) to be clustered and n, which denotes the number of all data instances in the training data. The algorithm treats each instance $\mathbf{x}_j = (x_{j1}, x_{j2}, \ldots, x_{jd})$ as a vector in real d-dimensional space, $\Re^d$.

Let $D = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_d\}$ denote the set of dimensions in the feature space. For each dimension $\mathbf{d}_i (1 \leqslant i \leqslant d)$ where i denotes the ith dimension, the minimum and maximum values of the training data are searched, and the intervals between them are divided into small equi-length partitions called *block*s, where the number of *block*s is determined by parameter β supplied by user. That is,

$$\mathbf{d}_i = \mathbf{d}_{i1} \cup \mathbf{d}_{i2}, \ldots, \cup \mathbf{d}_{i\beta}.$$

The algorithm then counts the number of instances included in each *block*. Let n_{ib} denote the number of data instances falling in *block* $\mathbf{d}_{ib} (1 \leqslant b \leqslant \beta)$ and n'_{ib} denote its ascending order; that is,

$$n'_{i1} \leqslant n'_{i2} \ldots \leqslant n'_{i\beta}.$$

For each dimension, the algorithm finds the first *block*, n'_{ig}, whose value is nonzero, and calculates the average value, AVG_i, of the ratio between two adjacent *block*s:

$$AVG_i \leftarrow \frac{1}{(\beta - g - 1)} \sum_{b=g}^{\beta-1} \frac{n'_{ib+1}}{n'_{ib}}$$

For each dimension, starting from the *block* n'_{ig}, the algorithm finds the first *block*, n'_{ip}, which satisfies Eqs. (1) and (2).

$$\frac{n'_{ib+1}}{n'_{ib}} < AVG_i, \frac{n'_{ip+1}}{n'_{ip}} > AVG_i \quad \textbf{subject to}: \ \forall b (g \leqslant b \leqslant p - 1) \tag{1}$$

$$\sum_{b=g}^{p} n'_{ib} < \gamma\% \tag{2}$$

where γ is the *maximum* ratio of attack data to the original training data n. In other words, Eq. (2) restricts the *block* n'_{ip} within $\gamma\%$ of n, and thus means that the number of attack data within the original training data does not exceed $\gamma\%$. If there is no such *block* which satisfies the above conditions, we find *block* n'_{ip} under Eqs. (2) and (3).

$$n'_{ip} = max\left(\frac{n'_{ib+1}}{n'_{ib}}\right) \quad \textbf{subject to}: \; b \geqslant g \tag{3}$$

The algorithm then defines the following two regions.

- **Sparse region:** $\{\mathbf{d}'_{i1}, \mathbf{d}'_{i2}, \ldots, \mathbf{d}'_{ip}\}$
- **Dense region:** $\{\mathbf{d}'_{ip+1}, \mathbf{d}'_{ip+1}, \ldots, \mathbf{d}'_{i\beta}\}$

By using the sparse and dense regions of each dimension, the algorithm partitions the training data into the following two groups.

- **Sparse group:** if $\mathbf{x}_j = (x_{j1}, x_{j2}, \ldots, x_{jd}) \in$ {sparse region}, $\mathbf{x}_j \in$ sparse group, for $\exists d$
- **Dense group:** if $\mathbf{x}_j = (x_{j1}, x_{j2}, \ldots, x_{jd}) \in$ {dense region}, $\mathbf{x}_j \in$ dense group, for $\forall d$

The members of the dense group are used afterward as the filtered training data.

3.2. Clustering process

In [20], during the clustering process, the filtered training data (*i.e.*, the dense group) are divided into k clusters as follows:

1. Initialization: The average of all data instances in the dense group becomes the initial cluster center.
2. Extracting: The furthest outside instance is selected from the quantitatively largest cluster among intermediate K clusters, and is regarded it as the new cluster center.
3. Assignment: The distance between each instance in the dense group and intermediate $K + 1$ cluster centers is computed, and is allocated to the closest cluster.
4. Updating: Every cluster's center is replaced with the mean of its members.
5. Iteration: Extracting and Updating are repeated until k clusters are created.

However, the difficulty lies in the fact that we cannot determine the exact number of the clusters which exist in the dense group. In order to cope with this limitation, we propose a new clustering method which enables extraction of all normal clusters from the dense group without specifically setting the value of k. The proposed method is based on the splitting method of clusters proposed in [22], in which the following two values are calculated with respect to cluster C_j.

- Δ_j: Average distance between data instances in cluster C_j and its cluster center $\mathbf{c}_j$
- σ_j: Standard deviation of the distance between data instances in cluster C_j and its cluster center $\mathbf{c}_j$

By using those two values, a data instance furthest from center $\mathbf{c}_j$ of cluster C_j is first found. After that, a new cluster with the data instance as its member, if the distance between the data instance and center $\mathbf{c}_j$ is larger than $\Delta_j + \sigma_j$ is created. This means that if a data instance within cluster C_j is its member, the distance between the data instance and center $\mathbf{c}_j$ must be smaller than the sum of Δ_j and σ_j. Finally, in order to avoid over-splitting of clusters, the intermediate K clusters are merged according to the similarity of the standard deviation between clusters and the closeness of distance between their centers.

However, we cannot directly apply this splitting method to the proposed clustering method, because the ratio of attack data to normal data in the dense group is extremely low and that study assumed that the merging process is needed after the splitting process due to the possibility of over-splitting. In many cases, a dense group consists of only normal data, but [22] assumed that each cluster contains outliers, *i.e.*, attack data. Since the distance between the outlier and its cluster center leads to the increase of Δ_j and σ_j, their summation causes overestimation of the size of a normal cluster. We, therefore, have to consider the following two conditions during the clustering process: (1) mitigation of the splitting criterion and (2) prevention of over-splitting. In order to satisfy the first condition, we adopt only Δ_j as the splitting criterion in the proposed clustering method. In other words, if the distance between a data instance and its center is larger than Δ_j, we regard it as the center of a new cluster. As the countermeasure for over-splitting, we adopt the maximum Δ as the splitting criterion, which is obtained from all data instances within the dense group. The basic strategy is that we regard all data instances with distance to the initial center smaller than the maximum Δ as an initial cluster, and only create clusters from the other data instances. The process of the proposed clustering method is as follows.

1. Initialization: The average of all data instances in the dense group becomes the initial cluster center.
2. Calculation Δ: This is the average distance between all data instances in the dense group and the initial cluster center.
3. Selection: The furthest outside instance is selected from the quantitatively largest cluster among intermediate K clusters.
4. Splitting: If the distance between the furthest outside instance and its cluster center is larger than Δ, it is regarded as the new cluster center.
5. Assignment: The distance between each instance in the dense group and intermediate $K + 1$ cluster centers is computed, and is allocated to the nearest cluster.
6. Updating: Every cluster's center is replaced with the mean of its members.
7. Iteration: Selection to Updating is repeated until there is no data instance which meets the splitting criterion in the Splitting phase.

4. Experimental results

4.1. Data set descriptions

In the intrusion detection field, the KDD Cup 1999 dataset [24] has been used for a long time as benchmark data for evaluating IDS performance. However, there is a fatal drawback in that this dataset is unable to reflect current network situations and the latest attack trends, because it was generated by simulation over a virtual network more than 10 years ago, thus its attack types are largely outdated. Despite this drawback, researchers have used it as their evaluation data, because it is quite difficult to acquire high-quality evaluation data due to privacy and competition issues. Many organizations share very few data with other institutions and researchers. To make matters worse, labeling traffic data as either normal or intrusion requires an enormous amount of time for many experts, even if real traffic data is available.

In order to provide more practical and useful evaluation results, we carried out the experiments over real traffic data obtained from several types of honeypots deployed on five different networks, *i.e.*, one class A and four class B networks, which are inside and outside Kyoto University [21]. For instance, certain honeypots are based on Windows XP with no patch, Windows XP with SP2, fully patched Windows XP, Windows Vista, Symantec honeypots based on Solaris, network printers, or home electronics such as televisions and video recorders. In addition to traditional honeypots which operate only in passive mode (*i.e.*, only receive attacks), we deployed proactive systems which access malicious web servers and join botnets to receive various types of commands. We collected all the traffic data to/from the honeypots, and observed that most traffic was involved in attack activities, and thus logged as attack data. We also investigated each connection for the presence of a buffer overflow attack in the collected traffic data. If an attack occurred, a shellcode and exploit code were identified using the dedicated software [12]. To improve the identification accuracy, we also used IDS alerts obtained by the SNS7160 IDS system [23] and malware information obtained by ClamAV [11]. With this diverse information, we thoroughly inspected the traffic data collected from honeypots and identified what happened on the networks.

On the other hand, we should prepare a great deal of *normal* data to fairly and effectively evaluate the performance of the proposed method, because most traffic data of honeypots were attack data. To this end, we deployed a mail server on the networks and regarded the traffic data it recorded as normal data. To manage the mail server, we operated it with several communication protocols, *e.g.*, SSH, HTTP and HTTPS, thus various attacks were also received. Though all these activities were included with the traffic data, they do not affect the performance of machine learning techniques due to their negligible amount.

As mentioned above, it is extremely difficult labeling all traffic data either as normal or attack with 100% accuracy, and even if it is possible, a tremendous amount of time and cost is required. However, in further inspection for identifying whether honeypot traffic data are a real attack or not using the dedicated software, we observed that the almost all honeypot traffic data used for the experiments were real attack data and there were few unidentified traffic data. Also, in the case of mail server traffic data, most were normal data and there were few attack data. For this reason, in the experiments, we regarded all traffic data gathered from Kyoto University honeypots as attack data and all traffic data caused by the mail server as normal data. Since the performance of the proposed method and Song's method is evaluated under the same condition (*i.e.*, with the same unidentified traffic data and mis-classified traffic data) even if unidentified traffic data and mis-classified traffic data are scattered in the evaluation dataset or can be treated just as noisy data, it could be said that the experiments were carried out fairly. Since the attack ratio of the evaluation data is less than 5%, it is also quite difficult for attack data to be grouped into a dense group. Of course, in the case of DDoS attacks, they can become members of the dense group, and the proposed method consequently fails. However, in such a case, they can easily be detected as an anomaly even without applying data mining and machine learning techniques to network traffic data.

4.2. Preprocessing and data preparation

In the experiments, we used the traffic data from three days (Nov. 1–3, 2007) to build the training data. The ratio of attack data occupied 80% of the original traffic data on average. In many cases, the ratio of attack data or extremely dangerous attack data observed on the network was significantly smaller than that of normal data. Considering this, we adjusted the ratio of attack data to normal data. To investigate the performance of the proposed method based on the different ratio, we also prepared three training data with different attack rates, *i.e.*, 1%, 2% and 5%. As a result, 58,294, 58,866 and 60,597 data

instances were randomly and fairly selected from the training data, and the numbers of attack data were 582, 1154 and 2885, respectively. As the testing data, we used the traffic data from 10 days: December 1, 8, 15 and 22 2007, January 10, 17 and 23 2008 and February 9, 16 and 23 2008. Note that the training and testing data were constructed from the same period of the traffic data used in [20].

With respect to the training and testing data, we extracted 14 statistical features from each session data, and used 13 continuous features excluding one categorical feature (*i.e.*, "flag") for the evaluation data. These 14 features were selected via the evaluation results of Mukkamala et al. which verified that many of the original 41 features of the KDD Cup 99 dataset [24] are substantially redundant [17]. The benchmark data is open to the public at [14]. Visit the website for more details.

4.3. Results

4.3.1. Quality of dense group

First, we compared the quality of the dense group (*i.e.*, filtered training data) extracted in the filtering phase, because in the proposed method, clustering performance heavily depends on the quality, *i.e.*, most data instances within the dense group must be normal data. Tables 1–3 show the comparison results according to the three sets of training data. In the comparison experiments, we set parameter β to the four reasonable values (100, 200, 500 and 1,000) verified from [20]. We additionally set the parameter α to the optimized values (1%, 2% and 5%) for each of the training datum in Song's method [20]. For the Eq. (2), we fixed the value of γ to 10%. This assumes that the ratio of attack data to original training data is lower than 10%. This assumption is fairly reasonable in the intrusion detection field, because the ratio of attack data to normal data in real traffic data typically is extremely small. Of course, it is best to set γ to the actual ratio of attack data within the traffic data to be analyzed, but it is very difficult to determine such a desired value beforehand, and the number of attack data also changes dynamically each time on a real network. On the other hand, for a network where the attack ratio of the traffic data exceeds 10%, it is better to inspect all traffic data observed from the network rather than apply data mining techniques to them.

Tables 1–3 clearly show that the filtering capability for attack data in the proposed method is superior to Song's. In other words, the number of attack data within the dense group obtained from the proposed method is always smaller than in Song's, while maintaining a great deal of normal data. In the proposed method, even if we set the value of γ to a constant one (*i.e.*, 10%), there is no attack data within the dense group when the ratio of attack data is 1% and 2%, and there are only two attack data when the ratio is 5%. We can also observe that Song's method is very weak against an increased attack rate (*i.e.*, 1%, 2% and 5%) within the training data. The number of normal data within the dense group decreases with an increase of the attack rate.

4.3.2. Quality of clusters

Table 4 shows the number of clusters generated by the proposed clustering method with respect to the three sets of training data. From Table 4, we can see that the number of clusters is $18 \sim 21$, and this means that there are $18 \sim 21$ normal patterns in the training data.

In order to investigate the accuracy of the proposed clustering method, we measured the Sum of Square Error (SSE) during the clustering process in that we created a total number of 50 clusters. Fig. 3 shows the SSE according to the number of clusters when we used the training data with 2% attack data. In general, as the number of clusters increases, the SSE of the entire clustering space decreases [5], and the turning point can be regarded as the optimal number we want to find [8]. From Fig. 3 clearly shows that the SEE turning point is near the number of clusters found by the proposed clustering method. Due to space limitations, only the results of the training data with 2% attack data are shown here, but similar results were observed for the other two sets of training data. This means that the proposed clustering method is able to discover the optimal number of normal patterns (clusters) scattered over the training data.

4.3.3. Performance evaluation

The ROC curve, which shows the detection rate according to changes in the false positive rate, is one of the most typical methods used to depict the performance of a system or algorithm. However, in the case of the proposed method, we cannot obtain such a curve, because the values of two parameters, (α and k) are automatically fixed during the training phase, and its performance is not influenced by parameter β, and consequently the detection rate and false positive rate are nearly

Table 1
Quality of dense group (attack ratio = 1%, α = 1% in Song's method).

β	Total number of normal data		Number of attack data	
	Proposal	Song	Proposal	Song
100	53,575	53,732	0	166
200	53,974	53,654	0	166
500	53,965	53,622	0	166
1000	53,861	53,936	0	166

Table 2
Quality of dense group (attack ratio = 2%, α = 2% in Song's method).

β	Total number of normal data		Number of attack data	
	Proposal	Song	Proposal	Song
100	53,575	50,860	0	426
200	53,974	50,652	0	426
500	53,965	50,666	0	426
1000	53,861	50,622	0	426

Table 3
Quality of dense group (attack ratio = 5%, α = 5% in Song's method).

β	Total number of normal data		Number of attack data	
	Proposal	Song	Proposal	Song
100	53,685	42,908	2	406
200	54,065	42,183	2	406
500	54,075	41,777	2	406
1000	53,971	41,761	2	406

Table 4
Number of clusters.

β	Ratio of attack data		
	1%	2%	5%
100	20	20	21
200	19	19	20
500	18	18	20
1000	19	19	20

unchanged. Thus, in the performance evaluation, we adopt another parameter, *N*, used in one-class SVM. This parameter represents the ratio of data points located outside of the hypersphere discovered by one-class SVM. In other words, if *N* is larger (or smaller), the size of the hypersphere is smaller (or larger). The number of data points inside the hypersphere decreases (or increases).

We compared the proposed method with Song's method using two criteria, *i.e.*, attack data and normal data detection accuracy as shown in Figs. 4–6. During the evaluation process, we fixed $\beta = 100$, and used the Libsvm [3] library to carry out the experiments with one-class SVM. Figs. 4–6 clearly show that the proposed method's detection capability is always superior to that of Song's method. We can also observe that the proposed method is robust in regard to an increase of attack data within the training data, while in the case of Song's method the detection rate for normal data decreases.

On the other hand, we have to take notice of the fact that the detection rate for normal data is better with smaller values of *N*. This is because the smaller values for *N* make the size of the hypersphere larger. In other words, since most data instances within *k* clusters generated by the proposed clustering method consist of normal data, the larger size of the hypersphere better represents normal patterns. The proposed method could resultantly identify normal data more accurately with smaller values of *N*, thereby achieving performance improvement.

5. Related work

During the last decade, many efforts have been made to improve the performance of IDSs based on *unsupervised* anomaly detection techniques. Eskin et al. [7] proposed a geometric framework for unsupervised intrusion detection and demonstrated that their IDS models built by unlabeled data are able to detect intrusions effectively. Laskov et al. [9] proposed an IDS model based on a quarter-sphere SVM that is one variant of one-class SVM with moderate success. Leung and Leckie [15] proposed a new density-based and grid-based clustering algorithm, called fpMAFIA, for intrusion and they showed that it is suitable for unsupervised anomaly detection. Song et al. [22] proposed a K-means-based clustering algorithm for intrusion detection, which improves the performance of IDSs by overcoming the shortcomings of the K-means clustering algorithm. We also proposed a new anomaly detection method based on clustering and multiple one-class SVM to improve the detection rate while maintaining a low false positive rate [20].

Although those IDSs are able to construct intrusion detection models without using any labeled training data and have the capability to detect 0-day attacks, there is a realistic problem in that they require some parameters for training and modeling them, and thus it is not easy to directly deploy them into a real network environment due to the difficulty of configuring and managing the suitable parameters.

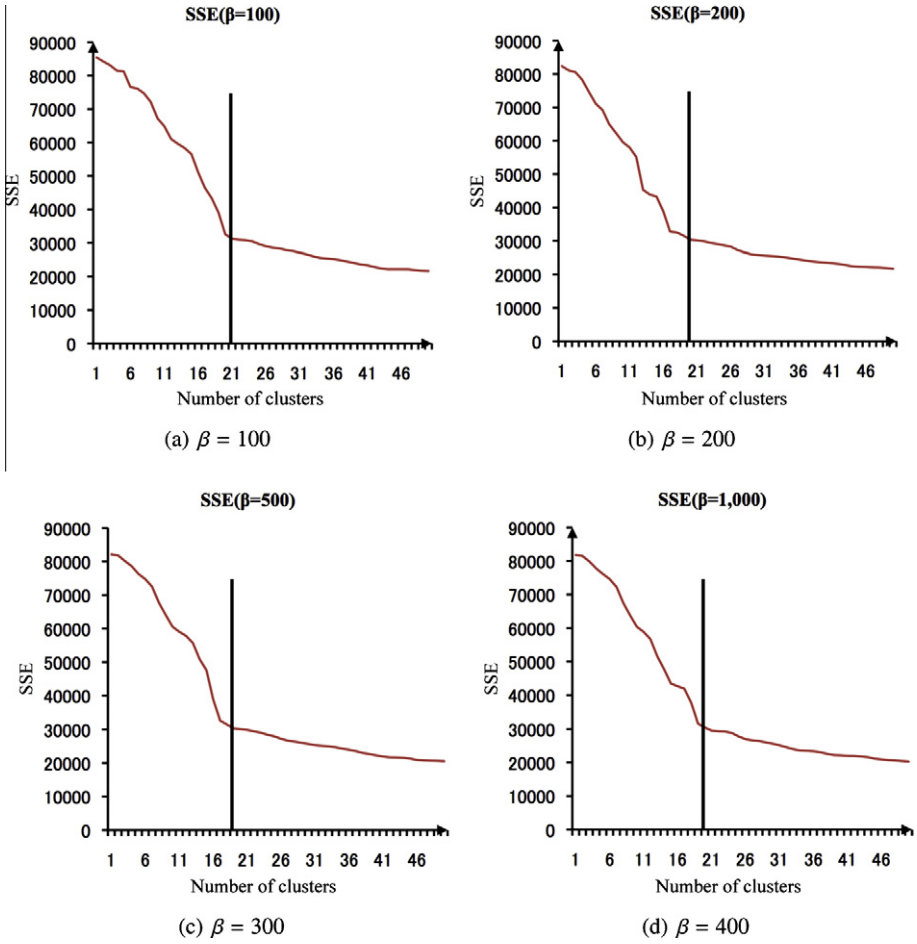

Fig. 3. Sum of Square Error based on number of clusters; ratio of attack data is 2%.

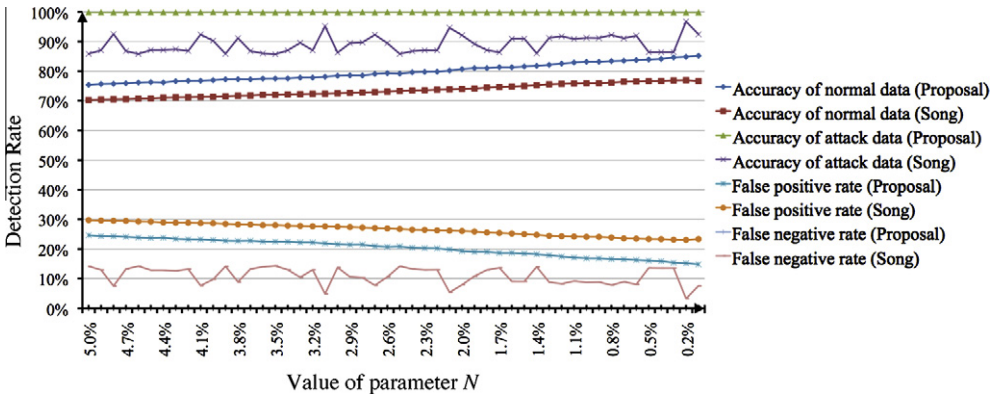

Fig. 4. Performance comparison of two algorithms; attack ratio is 1%.

Similar to the proposed method, on the other hand, some approaches have been proposed for tuning and optimizing IDS parameters. Since a large amount of alerts triggered by IDSs make operators difficult to analyze and manage efficiently – specially if 99% of them are false positives [25] – most previous research has been carried out for the purpose of reducing the number of false positives.

Clifton and Gengo [26] identified sequences of IDS alerts that likely result from normal behavior using data mining techniques, and then filtered out false positives from original alerts based on them. Yu et al. proposed a framework for correlating

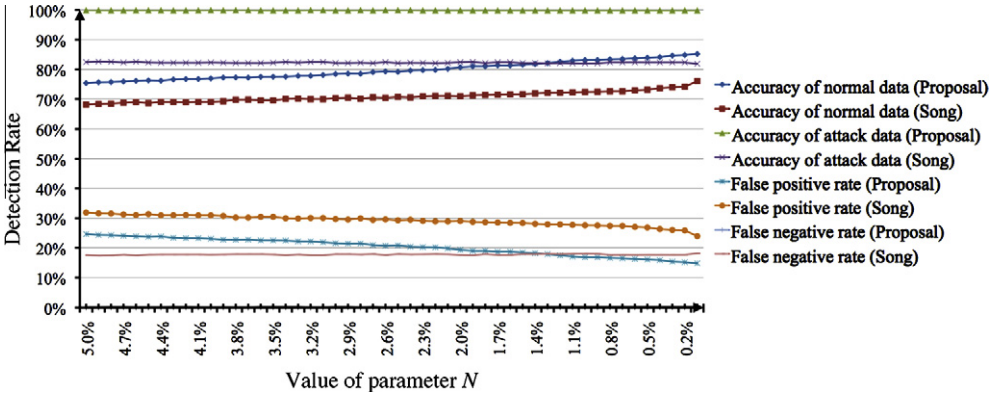

Fig. 5. Performance comparison of two algorithms; attack ratio is 2%.

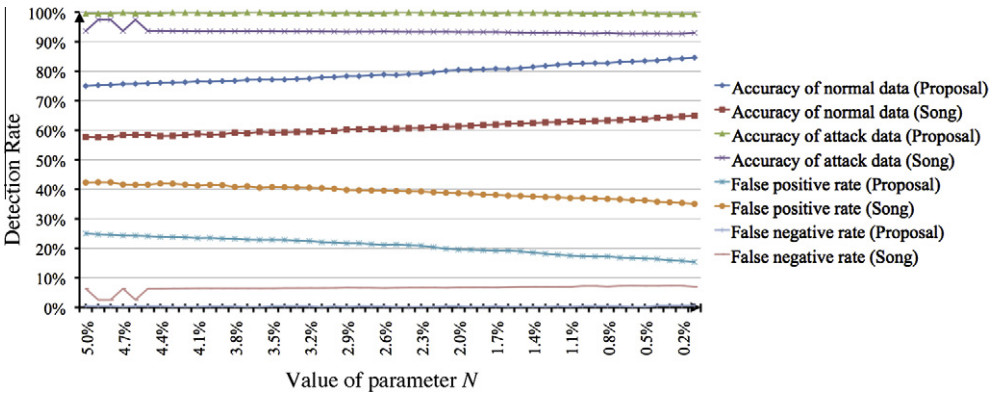

Fig. 6. Performance comparison of two algorithms; attack ratio is 5%.

and understanding IDS alerts, and their experimental results show that their method can reduce false positives and negatives, and provide better understanding of the intrusion progress by introducing confidence scores [27]. Pietraszek proposed Adaptive Learner for Alert Classification (ALAC), which is based on intrusion detection analyst's feedback based on their knowledge, for reducing false positives by classifying alerts into true positives and false positives using machine learning techniques [28]. Giacinto, et al. clustered IDS alerts to extract unified description of attacks from multiple alerts to attain a high-level description of threats [29]. In [30], Al-Mamory used root cause analysis to reduce IDS alarms and designed a new clustering technique to group IDS alarms and produce clusters in that IDS alarms have similar features.

These approaches enable IDS operators to analyze and manage IDS alerts more effectively, and some tried tuning and optimizing the IDS parameters. However, there are two significant differences between the previous research and the proposed method. First, the proposed method aims to build IDS models, which are based on an unsupervised anomaly detection method, by automatically tuning and optimizing the required parameters without manually configuring and managing them, while the previous research aimed to reduce only IDS alerts which were triggered by already deployed or molded IDSs. Therefore, in the case of the previous research, a problem still exists in that IDS operators have to tune and optimize the required parameters for building IDS models suitable for their network environments. Second, the previous research is based on supervised or semi-supervised approaches: tuning and optimizing IDS parameters heavily depends on domain knowledge or IDS operators' experience used during the analysis process. However, the proposed method is capable of finding out the optimized parameters for IDSs without using any prior knowledge, and we consequently demonstrated that the proposed method contributes to not only to reducing the false positive rate, but also to improving detection accuracy.

6. Conclusion and Future Work

In previous research [20], we proposed an unsupervised anomaly detection method that outperforms those in existing research. It is able to construct intrusion detection models without using labeled training data in an automated manner and has a detection capability of unforeseen attacks; *i.e.*, 0-day attacks. However, it needs three parameters (α, β and k) for constructing an intrusion detection model as described in Section 2. Considering IDS operators and managers suffer from

tuning and optimizing the required parameters based on changes of their network characteristics, it is still not easy to deploy them into a real network environment.

In this paper, we have proposed a new anomaly detection method by which we are able to optimize the values of two parameters, *i.e.*, α and k, without predefining them. Among the three parameters, we have focused on only two α and k, because we have proven that parameter β barely affects the performance of the previous anomaly detection method as long as we choose a reasonable range of value in [20]. The proposed method let us construct intrusion detection models based on anomaly detection without tuning the parameters, and this advantage can thus contribute to more practical operations in the real environment.

We evaluated the proposed method over a real dataset of network connections obtained from Kyoto University honeypots [21]. The experimental results show that the proposed method outperforms Song's method. Particularly, we observed that the proposed method is robust with regard to an increase of attack data within the training data, while Song's method is weak against it. Since the training and testing time of Song's method is very short [20] and the basic process of the proposed method is the same, we can also conclude that the time complexity of the proposed method is comparable to that of Song's. This advantage in computational complexity therefore makes the proposed method more promising.

In future work, we need to verify the performance of the proposed method using a greater range and more real traffic data, so that we are able to provide more accurate and useful evaluation results. Though our method does not require the predefined values of α and k, one-class SVM also uses its independent parameters, *e.g.*, N, during its learning process. Therefore, in order to make our method more useful and easily apply it to a real network environment, we need to devise a method by which we need not set any parameters during the building process of intrusion detection models.

References

[1] J. Allen, A. Christie, W. Fithen, State of the Practice of Intrusion Detection Technologies, Technical Report, CMU/SEI-99-TR-028, 2000.
[2] R. Bace, P. Mell, Intrusion Detection Systems, NIST Special Publications SP 800-31, November, 2001.
[3] Chih-Chung Chang, Chih-Jen Lin, LIBSVM: a library for support vector machines, 2001. Software Available from: <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
[4] N. Cristianini, J. Shawe-Taylor, An Introduction To Support Vector Machines, Cambridge University Press, 2000.
[5] R.O. Duda, P. Hart, D. Stork, Pattern Classification, John Wiley and Sons, Inc., New York, 2001.
[6] D.E. Denning, An intrusion detection model, IEEE Transactions on Software Engineering SE-13 (1987) 222–232.
[7] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, S. Stolfo, A Geometric Framework for Unsupervised Anomaly Detection: Intrusion Detection in Unlabeled Data, in Applications of Data Mining in Computer Security, 2002.
[8] Y. Guan, A. Ghorbani and N. Belacel, Y-means: A clustering method for intrusion detection, in: IEEE Canadian Conference on Electrical and Computer Engineering, Proceedings, 2003.
[9] P. Laskov, C. Schäfer, & I. Kotenko, Intrusion detection in unlabeled data with quarter-sphere support vector machines, in: Proc. DIMVA, 2004 pp. 71–82.
[10] <http://www.bro-ids.org/>.
[11] <http://www.clamav.net/>.
[12] <http://www.secure-ware.com/contents/product/ashula.html>.
[13] <http://www.snort.org/>.
[14] <http://www.takakura.com/Kyoto_data/>.
[15] K. Leung, C. Leckie, Unsupervised anomaly detection in network intrusion detection using clusters, in: ACSC2005, 2005.
[16] K.L. Li, H.K. Huang, S.F. Tian, W. Xu, Improving one-class SVM for anomaly detection, in: International Conference on Machine Learning and Cybernetics, Vol. 5, 2003, pp. 3077–3081.
[17] S. Mukkamala, A.H. Sung, Identifying significant features for network forensic analysis using artificial intelligent techniques, International Journal of Digital Evidence 1 (4) (2003).
[18] L. Portnoy, E. Eskin and S. Stolfo, Intrusion detection with unlabeled data using clustering", in: Proceedings of ACM CSS Workshop on Data Mining Applied to Security, 2001.
[20] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Yongjin Kwon, Unsupervised Anomaly Detection Based on Clustering and Multiple One-class SVM, IEICE Transactions on Communications E92-B (06) (2009) 1981–1990.
[21] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Cooperation of Intelligent Honeypots to Detect Unknown Malicious Codes, in: WOMBAT Workshop on Information Security Threat Data Exchange (WISTDE 2008), The IEEE CS Press, April, 2008.
[22] Jungsuk Song, Kenji Ohira, Hiroki Takakura, Yasuo Okabe, Yongjin Kwon, A clustering method for improving performance of anomaly-based Intrusion Detection System, IEICE Transactions on Information and Communication System Security E91-D (5) (2008) 1282–1291.
[23] Symantec Network Security 7100 Series. http://www.symantec.com/press/2004/no40726.html.
[24] The third international knowledge discovery and data mining tools competition dataset KDD99-Cup <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>, 1999.
[25] K. Julisch, Clustering intrusion detection alarms to support root cause analysis, ACM Transactions on Information and System Security, Vol. 6, ACM Press, 2003, pp. 443–471 (4).
[26] C. Clifton, G. Gengo, Developing custom intrusion detection filters using data mining, 21st Century Military Communications Conference Proceedings(MILCOM2000), vol. 1, pp. 440–443, Los Angeles, California, USA, 2000.
[27] Dong Yu, Deborah Frincke, A novel framework for alert correlation and understanding, in: ACNS 2004, LNCS 3089, 2004, pp. 452–466.
[28] T. Pietraszek, Using adaptive alert classification to reduce false positives in intrusion detection, in: Proc. of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID '04),Sept. 2004, pp.102–124.
[29] Giorgio Giacinto, Roberto Perdisci, Fabio Roli, Alarm clustering for intrusion detection systems in computer networks, in: MLDM 2005, LNAI 3587, 2005, pp. 184–193.
[30] S. Al-Mamory, H. Zhang, Intrusion detection alarms reduction using root cause analysis and clustering, ACM Computer Communications 32 (2) (2009) 419–430.
[31] Dominik Fisch, Alexander Hofmann, Bernhard Sick, On the versatility of radial basis function neural networks: A case study in the field of intrusion detection, Information Sciences 180 (12) (2010) 2421–2439.