ELSEVIER

Expert Systems with Applications 29 (2005) 713–722

Expert Systems with Applications

www.elsevier.com/locate/eswa

An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks

Ozgur Depren, Murat Topallar, Emin Anarim, M. Kemal Ciliz*

Bogazici University, Electrical and Electronics Engineering Department, Information and Communications Security (BUICS) Lab, Bebek, Istanbul, Turkey

Abstract

In this paper, we propose a novel Intrusion Detection System (IDS) architecture utilizing both anomaly and misuse detection approaches. This hybrid Intrusion Detection System architecture consists of an anomaly detection module, a misuse detection module and a decision support system combining the results of these two detection modules. The proposed anomaly detection module uses a Self-Organizing Map (SOM) structure to model normal behavior. Deviation from the normal behavior is classified as an attack. The proposed misuse detection module uses J.48 decision tree algorithm to classify various types of attacks. The principle interest of this work is to benchmark the performance of the proposed hybrid IDS architecture by using KDD Cup 99 Data Set, the benchmark dataset used by IDS researchers. A rule-based Decision Support System (DSS) is also developed for interpreting the results of both anomaly and misuse detection modules. Simulation results of both anomaly and misuse detection modules based on the KDD 99 Data Set are given. It is observed that the proposed hybrid approach gives better performance over individual approaches.

Keywords: Intrusion detection; Anomaly detection; Misuse detection; SOM; Decision trees; J.48; KDD Cup 99; Hybrid intrusion detection

1. Introduction

As interconnections among computer systems grow rapidly, network security is becoming a major challenge. Computer networks against denial-of-service (DoS) attacks, unauthorized disclosure of information and the modification or destruction of data have to be protected and the availability, confidentiality and integrity of critical information systems have to be provided.

An IDS system is a defense system, which detects hostile activities or exploits in a network.[1,2] Existing IDS systems can be divided into two catagories according to the detection approaches: anomaly detection and misuse detection or signature detection (Anderson, 1995; Rhodes, Mahaffey, & Cannady, 2000; Tiwari, 2002). Misuse detection systems try to match computer activity to stored signatures of known exploits or attacks. In other words, misuse detection systems use a priori knowledge on attacks to look for attack traces. Anomaly detection is an approach to detect intrusions by first learning the characteristics of normal activity. Then systems are designed to detect anything that deviates from normal activity (Kemmerer & Vigna, 2002; Ingham, 2003).

According to the resources they monitor, IDS systems are divided into two categories: Host based IDS systems and Network Based IDS systems (Anderson, 1995; Tiwari, 2002). Host based IDS systems are installed locally on host machines. Host based IDS systems evaluate the activities and access to key servers upon which a Host based IDS agent has been placed (Lichodzijewski, Zincir-Heywood, & Heywood, 2002). The network based IDS systems inspect the packets passing through the network (Ingham, 2003; Lichodzijewski, 2002).

In this work, a hybrid intrusion detection system (HIDS) utilizing both anomaly and misuse detection is proposed. The principle interest of this work is to benchmark the performance available from the proposed IDS architecture by using KDD 99 dataset, the benchmark dataset used by IDS researchers. The proposed IDS architecture consists of an anomaly detection module, a misuse detection module and a decision support system combining the results of the two detection modules.

* Corresponding author. Tel.: +90 212 359 6414.
E-mail address: ciliz@boun.edu.tr (M.K. Ciliz).

[1] http://www.windowsecurity.com/articles/Intrusion_Detection_Systems.html.

[2] http://www.windowsecurity.com/articles/Why_is_a_firewall_alone_not_enough_What_are_IDSes_and_why_are_they_worth_having.html.

0957-4174/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2005.05.002

KDD 99 dataset is used as the main intrusion detection dataset for both training and testing (Stolfo et al., 1999; DARPA Intrusion Detection Evaluation) different Intrusion Detection schemes. Anomaly Detection Module is responsible for detecting anomalous behaviors based on the trained normal behavior model. A SOM is used to build normal behavior model. Misuse Detection Module is responsible for detecting pre-defined attacks based on their attack signatures. The goal is to train the system with different types of attacks data and model different types of attack signatures. A C4.5 decision tree algorithm is used for classifying attacks. The Decision Support System (DSS) is responsible for interpreting the results of both anomaly and misuse detection modules and reporting intrusion detection activity. A rule-based DSS module is implemented and successfully tested.

2. System architecture

In this work, an HIDS utilizing both anomaly and misuse detection is proposed. As shown in the Fig. 1, the proposed IDS architecture consists of an anomaly detection module, a misuse detection module and a decision support system combining the results of the two detection modules. In the following sections, each module is explained in more detail.

2.1. Anomaly detection module

Anomaly detection is a key element of intrusion detection in which deviations from normal behavior indicate the presence of intentionally or unintentionally excited attacks or faults. Anomaly detection approaches are based on building models of normal data and detect deviations from the normal model in observed data. Anomaly detection algorithms have the advantage that they can detect new types of intrusions as deviations from normal usage.

In this work, in order to model network traffic, each connection record is examined and basic traffic features are extracted. After preprocessing, the goal of the intrusion detection algorithm becomes to train the system with normal data and model normal network traffic from the given set of normal data. Then, the task would be to determine whether the test data belong to 'normal' or to an 'abnormal' behavior from a given new test data. The proposed Anomaly Detection Module is composed of three submodules,

1. Preprocessing Module,
2. Anomaly Analyzer Modules and
3. Communication Module.

Each Anomaly Analyzer Module (TCP Anomaly Analyzer, UDP Anomaly Analyzer, ICMP Anomaly Analyzer) uses the SOM algorithm which is described in Section 2.1.1 to built profiles of normal traffic. The profile built in the Anomaly Analyzer Module will later be used to determine if a network connection is normal or abnormal. Communications Module handles the communications through the Decision Support System (DSS). Fig. 2 displays the block diagram of the Anomaly Detection Module.

2.1.1. Self-organizing maps

The SOM is a neural network model proposed by Kohonen for analyzing and visualizing high dimensional data (Kohonen, 2001). It belongs to the category of competitive learning models which are commonly used for various clustering problems successfully (Kohonen, 2001). The SOM is based on unsupervised learning to map nonlinear statistical relationships between high-dimensional

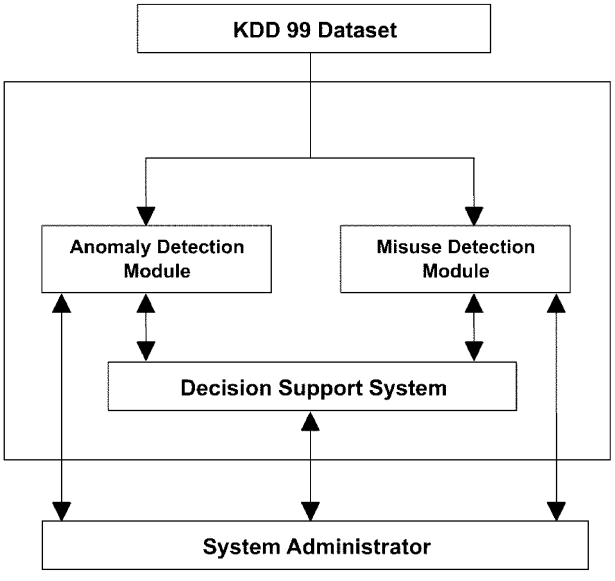

Fig. 1. Proposed hybrid IDS architecture.

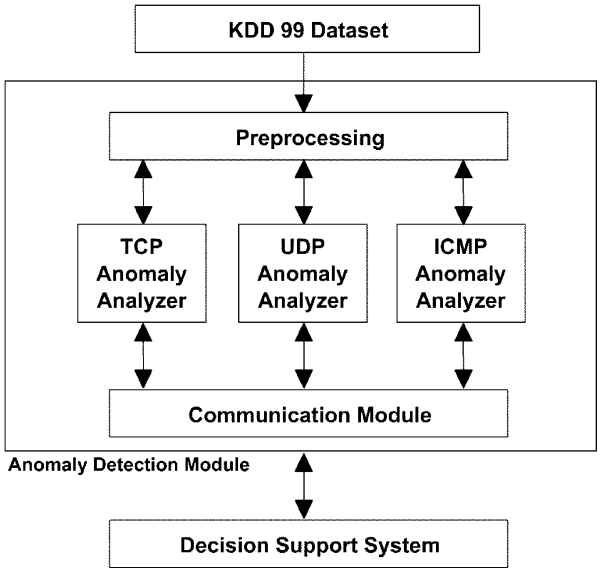

Fig. 2. Proposed anomaly detection module block diagram.

input data into two-dimensional lattice or grid which is also called the output space. Self Organizing Maps efficiently place similar patterns to adjacent locations in the output space and provides projection and visualization options for high dimensional data. In other words SOM provides topology preserving mapping from the input space to the two-dimensional lattice or grid of nodes[3] (Kayacik, 2003).

2.1.1.1. Structure of the SOM model. A SOM model is generally formed of neurons located on a regular, (usually 1- or 2-dimensional) lattice or grid. In the 2-dimensional case the neurons of the map can be arranged either on a rectangular or a hexagonal lattice.

Each neuron i of the SOM has an associated d-dimensional prototype (aka weight, reference, codebook or model) vector,

$$\boldsymbol{m}_i = [m_{i1} m_{i2} \cdots m_{id}],$$

where d is equal to the dimension of the input vectors[3] (Kayacik, 2003).

The neurons that are adjacent, belong to the 1-neighborhood N_{i1} of the neuron i. A neighborhood function determines how strongly the neurons are connected to each other. Neighborhood function and the number of neurons determine the accuracy and the generalization capability of the SOM mapping. The most commonly used neighborhood function is the Gaussian neighborhood function and given as

$$h_{ci}(t) = \exp\left(\frac{\|r_c - r_i\|^2}{2\sigma^2(t)}\right) \tag{1}$$

where r_c is the location of unit c on the map grid and the r_i is the neighborhood radius at time t. The learning rate $\sigma(t)$ is a decreasing function of time between [0,1]. Two commonly used forms are a linear function and a function inversely proportional to time[3] (Kayacik, 2003). For a detailed treatment of SOM models, the reader is referred to (Kohonen, 2001).

2.1.1.2. SOM training algorithm. Before the training, initial values are given to the prototype vectors (this is usually referred to as the weight initialization of the SOM model). The SOM is very robust with respect to the initialization, but proper initialization makes the algorithm converge faster to a better solution. Typically one of the three following initialization procedures is used: random initialization, sample initialization and linear initialization[3] (Kayacik, 2003).

In each training step, one sample vector x from the input data set is chosen randomly and a similarity measure is calculated between this chosen sample and all the weight vectors of the map. The Best-Matching Unit (BMU), denoted as c, is the unit whose weight vector has the greatest similarity with the input sample x. The similarity is usually defined by means of a distance measure, typically the Euclidian distance. Formally, the BMU is defined as the neuron for which the following condition holds,

$$\|x - w_c\| = \min_i\{\|x - w_i\|\} \tag{2}$$

where $\|\cdot\|$ is chosen as the Euclidian norm.

After finding the BMU, the prototype vectors of the SOM are updated. The prototype vectors of the BMU and its topological neighbors are moved closer to the input vector in the input space. This adaptation procedure stretches the prototypes of the BMU and its topological neighbors towards the sample vector. The SOM update rule for the weight vector of the unit i is given as

$$w_i(t+1) = w_i(t) + \alpha(t) h_{ci}(t)[x(t) - w_i(t)] \tag{3}$$

where t denotes time, $\alpha(t)$ is learning rate and $h_{ci}(t)$ is the neighborhood kernel around the winner unit c, with neighborhood radius $r(t)$, as defined in (1).

Also neighborhood radius typically decreases with time. Since large neighborhood radius makes the SOM more rigid, it is usually large in the beginning of training, and then it is gradually decreased to a suitable final radius, for example, one. Notice that if neighborhood radius is set to zero (i.e. $r=0$), the SOM algorithm reduces to k-means algorithm.

The weighting factor mask can be used for weighting variables according to their importance. With these changes, the distance measure becomes:

$$\|x - m\|^2 = \sum_{k \in K} w_k |x_k - m_k|^2 \tag{4}$$

where K is the set of variables of sample vector x. x_k and m_k are the kth components of the sample and prototype vectors, respectively, and w_k is the kth mask value. Note that the update rule does not change, therefore the 'weighting action' only affects finding of the BMU.

After the training of the SOM structure is finished, the final map converges to the probability density function of the input data (Kohonen, 2001). This final structure is simple and most importantly easy to visualize. Visualization of the SOM structure can be done by using U-matrix or Sammon's mapping[3] (Kayacik, 2003).

2.1.2. Preprocessing

In the KDD 99 Competition, raw packet based network traffic data is collected from the network by a network sniffer and it is processed into a stream of connections to form the intrusion detection dataset. In the KDD 99 intrusion detection dataset, 41 features are derived to summarize each connection information (Stolfo et al., 1999; DARPA Intrusion Detection Evaluation). From each connection, *six basic features* are used in this work. These are listed as follows

[3] http://www.cis.hut.fi/projects/somtoolbox/documentation/.

1. Duration of the connection
2. Protocol type such as TCP, UDP or ICMP;
3. Service type such as FTP, HTTP, Telnet;
4. Connection status flag;
5. Total bytes sent to destination host;
6. Total bytes sent to source host;

In order to train the SOM architecture, several data normalization and enumeration operations are necessary. SOMs usually treat each feature independently and work on normalized numeric variables. Therefore alphanumeric variables in the data set have to be enumerated and then all variables have to be normalized. Connection features—Protocol type, Service type and Connection status flag—are alphanumeric. As the SOM structure treats each feature independently, each instance of an alphanumeric character is mapped to sequential integer values. After enumeration operation takes place, normalization is performed. The goal of data normalization is that, none of components of input vectors has an overwhelming influence on the training result. Standard [0 1] normalization is used in this work.

The pre-simulation results of SOM training using either [0 1] or [−1 1] normalization did not give acceptable results. This is because, numerical features (like connection duration, total bytes set to destination/source host) of the connection feature vector have dynamic range values. As connection features are normalized, feature values become closer and cannot be detected by the SOM structure. For example, in DoS attacks destination byte values have 0 bytes and source byte values have 40–50 bytes. However, in normal connections both features have 40–50 bytes. For this case, if a [0 1] normalization is performed,

$$50/5000000 = 10^{-5} \text{ and } 0/5000000 = 0 \tag{5}$$

we get two extreme values as shown in (5). These two distinct values cannot be differentiated by the SOM structure. If remaining four parameters are the same, which is the case mostly in DoS attacks, this kind of attacks are interpreted as normal connections and cannot be detected. Therefore especially these two features, *source byte and destination byte*, have to be examined in detail. In order to accomplish this task, for both connection features the '*k-means algorithm*', which is an unsupervised learning algorithm like SOM, is used. The learning procedure is explained below:

1. Place k points into the source byte space to be clustered. These points represent initial group centroids.
2. By using k-means algorithm, obtain k cluster centers or groups (group of source byte values) and corresponding variances.

$$\text{(Source byte values)} \underset{k\text{-means algorithm}}{\longrightarrow} [(m_1, \sigma_1^2), (m_2, \sigma_2^2) \cdots (m_k, \sigma_k^2)] \tag{6}$$

3. For each cluster center value and variance pair, a Gaussian function is defined. Therefore k gaussian functions need to be defined.
4. For each source byte value, Gaussian function outputs are calculated and each source byte value is represented as Gaussian function outputs. Gaussian function outputs represent the probability of belonging to each cluster center for each source byte value.
5. This procedure is repeated for destination byte feature, as well.

As a result, for each source byte value the probabilities of belonging to each cluster center are calculated. Bu using the above proposed approach, normalization problem defined above has been overcome. Hence source byte value that would be hard to distinguish after normalization can now be differentiated by the SOM structure. Now, connection feature length has become 16.

2.1.3. Anomaly analyzer modules

Anomaly analyzer modules—TCP Anomaly Analyzer, UDP Anomaly Analyzer, ICMP Anomaly Analyzer—operate on different protocols, however their processing procedures are the same. Each Anomaly Analyzer Module uses the SOM algorithm to build profiles of normal behavior. Each SOM structure is trained with the corresponding normal traffic data and the profile of normal behavior is modeled. Our hypothesis is that normal traffic that represents normal behavior would be clustered around one or more cluster centers on the SOM lattice and any anomalous traffic representing abnormal and possibly suspicious behavior would be clustered outside of the normal clustering or would be clustered inside the normal clustering with high quantization error. Then, the profile built later is used to determine if a network connection is normal or abnormal.

For each SOM structure there are two phases of operation: SOM training phase and SOM classification phase. In the SOM training phase, SOM structure is trained with normal data and classifier parameters are calculated. Classifier parameters are 'SOM structure sMap' and the 'quantization vector'. In the SOM classification phase, a decision is made by using the classifier parameters. Fig. 3 displays the overall block diagram which explains the information flow in the training and classification phases of the anomaly detection based on the SOM structure.

2.1.3.1. SOM training phase. In order to train the SOM structure, first step is to enumerate and normalize input vectors. These tasks are accomplished in the Preprocessing Module. Input data set having input vectors of length 16 is obtained as explained in Section 2.1.2. This data set contains both normal traffic and the attacks traffic. In order to build up an anomaly detection module, normal data is extracted from the data set, preprocessed through Preprocessing Module and SOM structure is trained with the data set

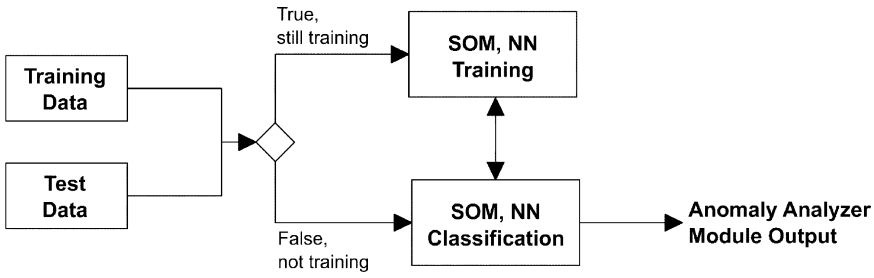

Fig. 3. Anomaly analyzer system architecture and data flow diagram.

containing only normal connections feature vectors. SOM training has two steps:

1. SOM initialization and
2. SOM training.

The flow diagram of the SOM Training Phase is shown in Fig. 4.

2.1.3.2. SOM initialization. First of all in the initialization phase, SOM structure is defined. Lattice or map size and shape have to be defined. A Rectangular lattice and a map size of [15,15] for TCP connections, [8,8] for UDP connections and [6,6] for ICMP connections are used in this work. Then, for all SOM structures, random initialization in which random values in the interval [0,1] are assigned to the codebook vectors is implemented. Random initialization is preferred since random initialization is much simpler as far as computational complexity is concerned and pre-simulation results show that PCA-initialization and random initialization give similar results.

2.1.3.3. SOM training. SOM training is done in two phases. The first phase is the ordering phase during which the reference vectors of the map units are ordered. The second phase is the fine tuning process. During the second phase, the reference vectors in each unit converge to its 'correct' values. The second phase is usually longer than the first one. Again, as in the ordering phase, training parameters are to be defined before training and they are different from the ordering phase. The learning rate is much smaller than that of the ordering phase. The neighborhood radius is also smaller. After these two phases of training, the map is ready to be used for classification.

2.1.3.4. SOM classification phase. SOM classification phase has two steps: SOM classifier parameters calculation step and SOM classifier step. In the SOM classifier parameters calculation step, a quantization vector is built by using training data. In the SOM classifier step, decision indicating if the test connection is an attack or not is made by using the quantization vector. The classification process is displayed in Fig. 5 in block diagram form.

2.1.3.5. SOM classifier parameters calculation. In order to classify if a network connection is normal or not, the best matching unit (bmu) and the quantization error (q_e) parameters are used. First, a fixed threshold is applied for classification. If the quantization error is greater than the threshold, it is classified as an anomaly. A variable thresholding algorithm is applied for the classification and the procedure is as follows:

1. Set a global threshold value (global_th), for q_e
2. Calculate [bmu,q_e] pairs for the network connection feature vectors labeled as normal (training data)
3. Calculate max q_e for each map unit (For example, TCP map size used in this work is [15,15], there are 225 neurons and 225 q_e values should be calculated)
4. Build up a quantization vector such that

a. If a map unit has no hit, then q_e value of the quantization vector is set to zero
b. If max q_e for a map unit is smaller than global_th, then q_e value of the quantization vector is set max q_e.
c. If max q_e for a map unit is greater than global_th, then q_e value of the quantization vector is set global_th.
d. A safety margin is added to the q_e values of the quantization vector

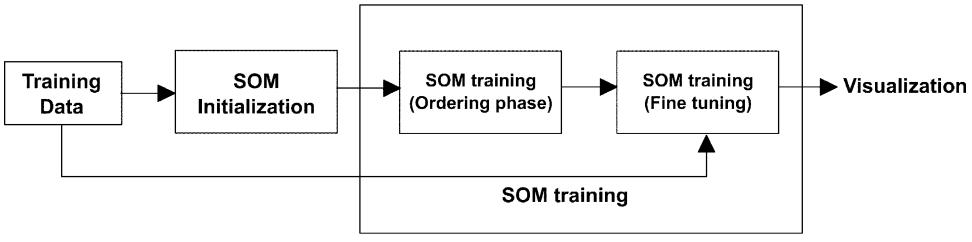

Fig. 4. The data flow diagram of the SOM training phase.

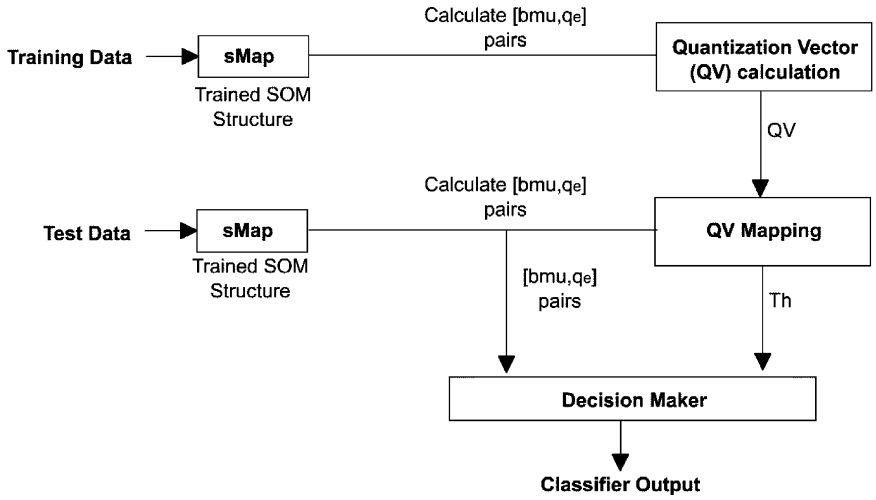

Fig. 5. The data flow diagram of the SOM classification phase.

For example, the size of the TCP Quantization Vector is [225,1], since TCP map size is [15,15]. A safety margin is added to be more realistic. Input vectors which are close to each other are mapped into closer locations on the SOM lattice. Therefore very close network connection vectors which are not in the training data set of the same class should hit the same neuron within the safety margin.

2.1.3.6. SOM classifier. Once the SOM structure sMap and quantization vector are obtained, the classification process is simple:

1. For each network connection feature vector calculate the [bmu,qe] pair by using sMap
2. By using Quantization Vector calculate the quantization threshold: Th = QV(bmu)
3. Classify the incoming network connection feature vector as:
   a. Classify connection as an attack if $q_e > Th$
   b. Classify connection as normal if $q_e < Th$
4. Generate SOM classifier output or Anomaly Analyzer Output

2.2. Misuse detection module

Misuse detection technique involves the comparison of a user's activities with the known behaviors of attackers attempting to penetrate a system (Kumar & Spafford, 1994, 1995). While anomaly detection typically utilizes threshold monitoring to indicate when a certain established metric has been reached, misuse detection techniques frequently utilize *a rule-based approach*. When applied to misuse detection, the rules become scenarios for network attacks. The intrusion detection mechanism identifies a potential attack if a user's activities are found to be consistent with the established rules. The use of comprehensive rules is critical in the application of expert systems for intrusion detection (Cannady, 1998).

In misuse detection approach, an attack and the characteristics of the attack that distinguish this attack from normal data or traffic are defined. These characteristics are known as the *signature* of an attack, and the signature becomes part of a database of attack signatures. When the IDS detects one these signatures, it raises an alarm. Signature detection requires an attack to be studied before it can be recognized. Systems protected by such a system are vulnerable to new attacks until the updated database is available. Therefore, a hybrid mechanism which incorporates anomaly detection and misuse detection is expected to capture most attacks within a network under security threats.

In this work, a type of a decision tree algorithm is used as a misuse detection module. The rules which are generated from the decision tree are then used to classify the attacks.

2.2.1. Decision trees for supervised learning

Decision tree algorithms are supervised learning algorithms which recursively partition the data base on its attributes, until some stopping condition is reached. This recursive partitioning gives rise to a tree-like structure. The aim is that the final partitions (leaves of the tree) are homogeneous with respect to the classes, and the internal nodes of the tree are decisions about the attributes that were used to reach the leaves. The decisions are usually simple attribute tests, using one attribute at a time to discriminate the data. New data can be classified by following the conditions defined at the nodes down. J.R. Quinlan has popularized the decision tree approach. The latest public domain implementation of Quinlan's model is C4.5 (Quinlan, 1993). The Weka classifier package has its own version of C4.5 known as J48. WEKA is open source Java code created by researchers at the University of Waikato in New Zealand (Witten & Frank, 1999).

General decision tree approach can be specifically summarized as below:

1. Choose an attribute that best differentiates the output attribute values.
2. Create a separate tree branch for each value of the chosen attribute.
3. Divide the instances into subgroups so as to reflect the attribute values of the chosen node.
4. For each subgroup, terminate the attribute selection process if:

a. All members of a subgroup have the same value for the output attribute, terminate the attribute selection process for the current path and label the branch on the current path with the specified value.
b. The subgroup contains a single node or no further distinguishing attributes can be determined. As in (a), label the branch with the output value seen by the majority of remaining instances.

5. For each subgroup created in (3) that has not been labeled as terminal, repeat the above process.

The above algorithm is applied to the training data. The created decision tree is usually tested on a test data set, provided one is available. If test data is not available, J48 performs a cross-validation using the training data or split the whole data set into three parts, training data, validation data and test data (Witten & Frank, 1999).

2.2.2. Evaluation of machine learning and data mining

Any results from this machine learning procedure must be evaluated before we can have any confidence in its predictions. There are several standard methods for evaluation. In this work cross validation and bootstrap methods are used for evaluation.

Performance cannot be measured on the data set which is used to train the classifier. This would give an overly optimistic measure of its accuracy. The classifier may overfit the training data and therefore evaluation on an independent test data set is the most common way to obtain an estimate of the classification accuracy on future unseen data. If the amount of data available is large enough, then partitioning the data into training and independent test sets is the most common and fastest method of evaluation.

On smaller amounts of data, holding out a large enough independent test set may mean that not enough data is available for the training. In such cases, cross validation is preferred. The data is partitioned into a fixed number (N) of partitions or 'folds'. N is commonly chosen as 10, although 3 is also a popular choice in most applications. A typical example was displayed in Fig. 6 which shows the case for $N=3$. Each 'fold' is held out as test data in turn, while the other $(N-1)$ folds are used as the training data. Performance of the classifiers produced for each of the N folds is measured and then the N estimates are averaged to give the final result. 'Leave-one-out' cross validation scheme is standard cross validation taken to its extreme: instead of 10 folds, each individual data is held out in turn so there are as many folds as items in the data set. This increases the amount of data available for training each time, but it is a computationally expensive process to build so many classifiers (Clare, 2003).

The bootstrap is a method that constructs a training set by sampling with replacement from the whole data set (Efron & Tibshirani, 1993). The test set comprises of the data that are not used in the training set. This means that the two sets are independent, but the training set can contain repeated items. This allows a reasonable sized training set to be chosen, while still keeping a test set. Kohavi compares the bootstrap and cross validation schemes, and show examples where each fails to give a good estimate of accuracy, and compares their results on standard data sets (Kohavi, 1995).

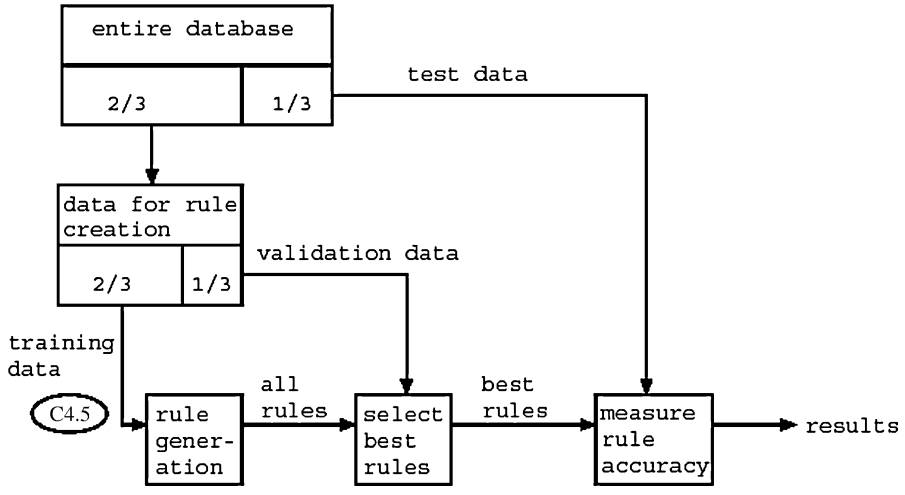

Fig. 6. The data was split into 3 parts, training data, validation data and test data. Training data was used for rule generation, validation data for selecting the best rules and test data for measuring rule accuracy. All three parts are independent [23].

2.2.3. Important parameters of the decision tree algorithm

The most important parameter of the decision tree algorithm J.48 is the confidence value. The confidence value is used when pruning the tree (removing branches that are deemed to provide little or no gain in statistical accuracy of the model). A default value of 25% works reasonably well in most cases, but it can be modified. However, if the actual error rate on the real data (or the error rate on cross-validation) is significantly higher than the error rate on the training data, the confidence factor can be decreased to cause more drastic pruning, and a more general model of the data. If a more specific modeling based on the training data is desired, the confidence factor can be increased, which will decrease the amount of pruning that occurs (Witten & Frank, 1999).

The other parameter determines the minimum number of instances that must be present in the training data for a new leaf to be created in the decision tree to handle those particular instances. This parameter is used to create a more generalized or specialized tree; a higher number will create a more generalized tree and a lower number will create a more specialized tree.

Another important parameter for J.48 is the number of folds for cross-validation. It determines how to construct and then test the model in the absence of test data. If the number of folds for cross-validation is x, then $(x-1)/x$ of the training data is used to construct the model and $1/x$ of the training data is used to test the model. This process is then repeated 'x' times so that all the training data is used exactly once in the test data. The 'x' different error estimates are then averaged to yield an overall error estimate. While extensive tests on numerous datasets have shown that ten-fold cross-validation is one of the best numbers for getting the most accurate error estimate, other values can be used. Decreasing the number of folds from the default value of 10 will likely decrease the amount of time it takes for the decision tree to be generated, and increasing the number of folds will likely increase the amount of time it takes. Of course, increasing the number of folds will create a larger dataset for the training data, which may increase accuracy of the decision tree; similarly, decreasing the number of folds will create a smaller dataset for the training data, which may decrease the accuracy of the decision tree (Witten & Frank, 1999).

If the attributes which are used to make the partition is symbolic, then there will be one branch per possible value of this attribute. If the attribute is continuous, the branch will usually be a two-way choice: comparing the value to see whether it is less than a fixed constant or not. This constant is determined by the range of values in the dataset (Witten & Frank, 1999).

2.3. The decision support system

Decision Support System (DSS) is responsible for interpreting the results of both anomaly and misuse detection modules which are discussed in detail in Sections 2.1 and 2.2. DSS is the final module of the proposed architecture which reports the intrusion detection activity to the end user. Different types of DSS can be implemented to accomplish this task. Rule-based DSS used in this work is composed of simple user-defined rules to make a decision. It is based on the heuristic rules defined by the end user. Main advantage here is that, it is simple and fast. Several rule sets can be defined. Rules which are used in this module is given as follows,

2.3.1. Rules

(1) If anomaly module detects an attack and misuse module detects an attack then *it is an attack* and misuse module *classifies this attack*
(2) If anomaly module does not detect an attack and misuse module detects an attack then *it is an attack* and misuse module *classifies this attack* (simply trust misuse module)
(3) If anomaly module detects an attack and misuse module does not detect an attack then it is an attack and it is *defined as an unclassified attack*.

3. Simulation results

In the simulation tests, the KDD data set is used which is the common data set used in IDS research papers. 10 percent of the KDD 99 data set is used in this work. Normal connections are extracted from the data set and SOM structures are trained with 50% of the normal connections. For TCP, UDP and ICMP protocols, SOM structures utilized a rectangular map shape. Random initialization is used. For the TCP connections, a 15×15 map size, for the UDP connections an 8×8 map size and for ICMP connections a 6×6 map size is used. Training parameters which are used in the simulation tests are shown in Table 1.

The preprocessed data set is fed to the J.48 decision tree algorithm and all the attributes are selected to generate a J. 48 decision tree. A confidence value is used when pruning the tree (removing branches that are deemed to provide little or no gain in statistical accuracy of the model). Since a

Table 1
Training parameters of the SOM structures used in the simulation tests

Training parameters	Ordering phase	Fine tuning phase
Neighborgood function	Gaussian	Gaussian
Radius [initial final]	TCP: [15 1]	TCP: [1 1]
	UDP: [8 1]	UDP: [1 1]
	ICMP: [6 1]	ICMP: [1 1]
Learning type	Sequential	Sequential
Learning rate	0.6	0.05
Learning function	Linear	Linear
Epochs	1	1

```
wrong_fragment <= 0
|  num_compromised <= 0
|  |  count <= 236
|  |  |  dst_host_srv_diff_host_rate <= 0.24
|  |  |  |  dst_host_same_srv_rate <= 0.01
|  |  |  |  |  src_bytes <= 1
|  |  |  |  |  |  rerror_rate <= 0.98
|  |  |  |  |  |  |  serror_rate <= 0.32
|  |  |  |  |  |  |  |  count <= 2
|  |  |  |  |  |  |  |  |  protocol_type = tcp: portsweep
|  |  |  |  |  |  |  |  |  protocol_type = udp: satan
```

Fig. 7. A small part of a decision tree obtained for the KDD 99 Data Set, showing decisions at the nodes, and final classification at the leaves.

default value of 25% works reasonably well in most cases, this default value is set for the evaluation. Then, both the '10 fold' cross validation and bootstrap methods are used and the results for '10 fold' cross validation method is presented in this work. Fig. 7 shows an example of a small part of a J. 48 decision tree obtained for the KDD 99 dataset.

After the construction of a J.48 decision tree, the best rules are derived for each type of attack class. Some sample rules, for portsweep and satan attacks, derived from the J.48 decision tree are given as follows:

The rule for the portsweep attack:

IF wrong_fragment $<=0$ AND num_compromised $<=0$ AND count $<=2$ AND dst_host_srv_diff_host_rate $<=0.24$ AND dst_host_same_srv_rate $<=0.01$ AND src_bytes $<=1$ AND rerror_rate $<=0.98$ AND serror_rate $<=0.32$ AND protocol_type = tcp THEN attack = portsweep

The rule for the satan attack:

IF wrong_fragment $<=0$ AND num_compromised $<=0$ AND count $<=2$ AND dst_host_srv_diff_host_rate $<=0.24$ AND dst_host_same_srv_rate $<=0.01$ AND src_bytes $<=1$ AND rerror_rate $<=0.98$ AND serror_rate $<=0.32$ AND protocol_type = udp THEN attack = satan

The above two sample rules cover the attack types 'portsweep and satan'. The remaining attacks are similarly covered by the other rules generated from the rest of the J.48 decision tree. Then these rules which are generated from the decision tree are tested on the KDD 99 data for the accuracy of the rules and *detection rates*, *classification rates* and *other performance* criteria are computed. Simulations are performed for the hybrid IDS model and the results for each module and each type of attack is given in Table 2 and Table 3.

4. Conclusions and future work

Simulation results of both anomaly and misuse detection modules based on the KDD 99 data set are given in Table 2. We obtained a detection rate of % 98.96 and a false positive rate of % 1.01 for anomaly detection module and also a classification rate of % 99.61 and a very low false positive rate of % 0.20 are achieved for the misuse detection module.

As observed from these results, misuse module gives very low false positive rates; on the other hand anomaly detection module detects some type of attacks that misuse module misses such as the 'ftp write' attack. However, anomaly module also gives relatively higher false positive rate.

The proposed hybrid IDS takes the advantages of *both modules* and combines the outputs of these two modules based on a simple decision support mechanism. As a result, a detection rate of % 99.90, a classification rate of % 99.84 and a false positive rate of % 1.25 are achieved by the proposed hybrid approach and these results are demonstrated in Table 2 and Table 3. It is observed that the proposed hybrid approach gives better performance over individual approaches.

For future work, different implementations for anomaly and misuse modules could be explored. For example, temporal relations between successive connections can be used for intrusion detection. For anomaly detection, instead of analyzing the whole data set, the data set can be first classified based on different 'network services' and then the anomaly detection for each network service type could be performed. For misuse detection module, instead of analyzing and generating rules for the whole data set, the data set can again be classified into different services and rules for each service type can be generated with better accuracy and less number of attributes. The determination of the most dominant features for each type of attack is an open area for intrusion detection researchers.

Table 2
Intelligent IDS System detection results

Detection module	Total # of instances	Total # of attacks	Detected (detection rate)	Missed (missed rate)	False positives (FP rate)
Anomaly	199677	128452	127118 (98.96%)	1334 (1.04%)	716 (1.01%)
Misuse	199677	128452	127950 (99.61%)	502 (0.39%)	127 (0.20%)
Hybrid	199677	128452	128328 (99.90)	124 (0.1%)	797 (1.25%)

Table 3
Detection rates of each attack for the Intelligent IDS System

Attack name	Protocol type	Total # of attack instances	Detected (detection rate)	Classified (classification rate)	Missed (missed rate)
Back	TCP	2103	2103 (100%)	2102 (99.95%)	0 (0%)
Buffer overflow	TCP	12	11 (91.67%)	10 (83.33%)	1 (8.33%)
Ftp write	TCP	8	2 (25%)	0 (0%)	6 (75%)
Guess passwd	TCP	53	52 (98.11%)	51 (96.23%)	1 (1.89%)
Imap	TCP	50	50 (100%)	48 (96%)	0 (0%)
Ipsweep	TCP	94	94 (100%)	94 (100%)	0 (0%)
Land	TCP	17	17 (100%)	12 (70.59%)	0 (0%)
Load module	TCP	9	7 (77.78%)	5 (55.56%)	2 (22.22%)
Multihop	TCP	7	5 (71.43%)	5 (71.43%)	2 (28.57%)
Neptune	TCP	41084	40998 (99.79%)	40950 (99.67%)	86 (0.21%)
Nmap	TCP	103	103 (100%)	98 (95.16%)	0 (0%)
Perl	TCP	2	2 (100%)	2 (100%)	0 (0%)
Phf	TCP	3	3 (100%)	3 (100%)	0 (0%)
Portsweep	TCP	639	639 (100%)	634 (99.22%)	0 (0%)
Rootkit	TCP	7	4 (57.14%)	4 (57.14%)	3 (42.86%)
Satan	TCP	1416	1408 (99.36%)	1408 (99.36%)	8 (0.64%)
Spy	TCP	2	1 (50%)	0 (0%)	1 (50%)
Warezclient	TCP	1020	1010 (99.02%)	1006 (98.62%)	10 (0.98%)
Warezmaster	TCP	20	17 (85%)	17 (85%)	3 (15%)
Nmap	UDP	25	24 (96%)	24 (96%)	1 (4%)
Satan	UDP	170	170 (100%)	169 (99.41%)	0 (0%)
Teardrop	UDP	397	397 (100%)	397 (100%)	0 (0%)
Ipsweep	ICMP	768	768 (100%)	768 (100%)	0 (0%)
Nmap	ICMP	103	103 (100%)	98 (95.16%)	0 (0%)
Pod	ICMP	102	102 (100%)	102 (100%)	0 (0%)
Smurf	ICMP	80238	80238 (100%)	80238 (100%)	0 (0%)
Total	All	199677	128328 (99.90)	128245 (99.84%)	124 (0.1%)

References

Anderson, J. (1995). *An introduction to neural networks*. Cambridge: MIT Press.

Cannady J. (1998). Artificial neural networks for misuse detection, Proceedings of the 1998 National Information Systems Security Conference (NISSC'98), (pp. 443–456). Arlington, VA.

Clare A. (2003). Machine learning and data mining for yeast functional genomics, PhD Thesis, Department of Computer Science University of Wales, Aberystwyth.

DARPA Intrusion Detection Evaluation, MIT Lincoln Laboratory, (http://www.ll.mit.edu/IST/ideval).

Efron, B., & Tibshirani, R. J. (1993). *An introduction to the bootstrap*. New York, NY: Chapman and Hall.

Ingham K. (2003). Protecting network servers.Technical Report, Department of Computer Science, University of New Mexico.

Kayacik H.G. (2003). Hierarchical self organizing map based IDS on KDD benchmark, MS Thesis, Middle East Technical University.

Kemmerer, R. A., & Vigna, G. (2002). Intrusion detection: A brief history and overview. *IEEE Security and Privacy Magazine* .

Kohonen, T. (2001). *Self-organizing map* (3rd ed.). Berlin: Springer-Verlag.

Kumar, S., & Spafford, E. (1994). *A pattern matching model for misuse intrusion detection Proceedings of the 17th national computer security conference* pp. 11–21.

Kumar, S., & Spafford, E., (1995). A software architecture to support misuse intrusion detection, Technical report, Department of computer Sciences, Purdue University, (CSD-TR-95-009).

Lichodzijewski P. (2002). Network based anomaly detection using self organizing maps, Technical Report, Nova Scotia: Dalhousie University Halifax.

Lichodzijewski, P., Zincir-Heywood, A., & Heywood, M. (2002). *Host-based intrusion detection using self-organizing maps Proceedings of the 2002 IEEE IJCNN, Hawaii, USA*.

Quinlan, J. R. (1993). *C 4.5: programs for machine learning*. San Mateo: Morgan Kaufmann Publishers.

Rhodes, B., Mahaffey, J., & Cannady, J. (2000). *Multiple self-organizing maps for intrusion detection Proceedings of the 23rd national information systems security conference, Baltimore, MD*.

Stolfo SJ, et al., KDD cup 1999 data set, Irvine: University of California. KDD repository, http://kdd.ics.uci.edu.

Tiwari, P. (2002). *Intrusion detection Technical report, department of electrical engineering indian institute of technology, Delhi*.

Witten, I. H., & Frank, E. (1999). *Data mining: practical machine learning tools with Java implementations*. San Francisco: Morgan Kaufmann Publishers.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proceedings international joint conference on artificial intelligence* pp. 1137–1143.