Towards Practical Anomaly-based Intrusion Detection by Outlier Mining on TCP Packets

Prajowal Manandhar and Zeyar Aung

Institute Center for Smart and Sustainable Systems (iSmart)
Masdar Institute of Science and Technology, Abu Dhabi, UAE
{pmanandhar,zaung}@masdar.ac.ae

Abstract. Intrusion detection System (IDS) is an important part of the security of large networks like the Internet. With increasing number of data being transmitted day by day from one subnetwork to another, the system needs to identify intrusion in such large datasets in an effectively and timely manner. So the application of knowledge discovery comes handy to identify unusual accesses or attacks. Improving an IDS's performance and accuracy is one of the major challenges network security research today. In this paper, we propose a practical anomaly-based IDS using outlier mining of the readily available basic Transmission Control Protocol (TCP) header information as well as other easily derivable attributes. We use a two-step approach of k-means clustering and one-class support vector machine (SVM) to model the normal sessions presented in MIT DARPA '99 dataset. We then feed the testing set to the resultant model to predict the attacks sessions.

Keywords: IDS, outlier mining, k-means clustering, one-class SVM, TCP

1 Introduction

Intrusion detection is a network security mechanism to protect the computer network system from invasions or attacks. As technologies evolve, it increases the risks of new threats evolving along with them. Thus, when a new type of attack emerges, an intrusion detection system (IDS) needs to be able to respond in an effectively and timely fashion in order to avoid hazardous effects. Intrusion detection has been one of the core areas of computer security whose objective is to identify these malicious activities in network traffic, and importantly, to protect the resources from threats. In today's context, the biggest challenge could be dealing with "big data", i.e., a huge volume of network traffic data which gets collected dynamically in the network's communications [1].

The common use of TCP means that it is likely to be exploited for misuse and various forms of attacks. Thus, there is a possibility of malicious behavior that can be executed through the TCP/IP protocols without being noticed/blocked by a firewall or a traditional signature-based IDS as they do not recognize new variations of attacks. So, there is a pressing need for a practical and easy-to-deploy IDS for TCP based on anomaly detection. Such an anomaly-based IDS

will be able to identify new attack variations and thus supplement the existing firewalls and signature-based IDSs. In fact a number of anomaly-based IDSs has been proposed by researchers throughout the years [2]. However, none of those become widely used in practise till now, and signature-based IDSs, both commercial and freeware, are still dominating the market [3].

One of the main reasons behind the underdeployment of anomaly-based IDSs is that they cannot be easily deployed. A majority of anomaly-based IDSs use data mining and machine learning algorithms, which take feature vectors containing some complicated features as their inputs. However, extracting such complicated features from raw TCP network traffic data is not a straight-forward task. The most notable example is the widely-used KDD '99 dataset [4] where, 13 out of 41 features are extracted based on domain knowledge. Unfortunately, to our best knowledge, there is no publicly available tools to automatically extract those features from the raw TCP data. As such, it is virtually impossible for an average network administrator to deploy an anomaly-based IDS which relies on such complicated feature vectors. In order to address this, instead of complicated feature vectors, we opt to use the raw TCP data, which can be easily obtained using the tcpdump tool. Many researchers have pointed out that different patterns of TCP flags associated with anomalies. In our approach, we inspect the TCP headers information from the TCP/IP packets which allows for high-speed network status detection.

Although a number anomaly-based IDS using raw TCP data has been proposed in the past, some of them like PHAD/ALAD [5] are packet-based. They mainly rely on the information of individual TCP packets in detecting anomalies and hence, it is not able to capture higher-level anomalous events that may occur in a "session" involving multiple packets, even though the individual packets in that particular session seem innocent. In order to address this issue, some session-based approach such as NATE [6], have been proposed. Our method in this paper also employs a session-based approach. We use MIT's DARPA '99 tcpdump dataset [7] in order to evaluate method's effectiveness. Although the DARPA dataset is known to have some weaknesses [7, 8], we use it for a proof of concept of our proposed IDS method, which can be readily applied to any tcpdump files. Our objective is to provide network security professionals with a practical anomaly-based IDS that can be easily deployed as a supplementary tool to the existing firewalls and and signature-based IDSs. To this end, we make the scripts/soure code and the running intrusions of our IDS program available in http://www.aungz.com/IDS/.

2 Background

2.1 Transmission Control Protocol (TCP)

The TCP protocol is a connection-oriented protocol that enables the reliable and ordered transmission of traffic in Internet application. The TCP header information is important for the establishment of a trusted connection. The TCP sequence and acknowledgement numbers provide unique identifiers of the

connection to the client and server, and also provide confirmation that data has been received. TCP flags are used to control the state of the TCP connection. The client and server must complete a three-way handshake before any data exchange takes place. However, either party can initiate termination of the connection by resetting with a TCP RST packets. Thus, we have focused on areas where, intruders look to exploit this weakness in the protocol and can come up with various attacks.

2.2 Intrusion Detection Systems

An intrusion detection system (IDS) is a device or tool that monitors network activities for malicious activities and generates alarms and logs. IDS can be categorized into two types based on how it works:

1. **Signature-based IDS:** A rule-based approach, where pattern matching on known signatures leads to high accuracy for detecting threats. But it cannot detect novel attacks as novel attack signatures are not available for pattern matching. The limitation with this IDS is their downfall to detect new attacks and also neglect minor variations of known patterns as well as an overhead costs in order to maintain signature databases.
2. **Anomaly-based IDS:** A statistical approach which analyzes the deviation from the normal activities by mining information from available data. It can detect novel attacks by comparing suspicious ones with normal traffics but has high false alarm rate due to difficulty of generating practical normal behavior profiles for protected systems [9]. More recently, different statistical techniques have been applied for IDSs, especially for anomaly-based ones. These techniques are used when there is no structured knowledge about the pattern of data. Many data mining/machine learning algorithms are considered good for attack detection, in which decision tree is considered one of the most powerful and effective ways of detecting attacks in anomaly detection [10, 3]. In addition, k-nearest neighbor (KNN), Naive Bayes, Artificial neural network (ANN), support vector machine (SVM), random forest, and fuzzy logic are also known to be used for learning and classifying the network traffic data as normals or attacks [10]. Ensemble learning, which is a combination of multiple machine learning approaches, also is being used to improve the result of intrusion detection as compared to the result of a single classifier.

2.3 Anomaly-based IDS using Outlier Mining

An outlier can be regarded as an observation which varies from the other observations as to arouse suspicions that it was generated by a different mechanism. It was also observed that researchers have tried two approaches either to model only normal data or both normal and abnormal data for finding intruders. Modeling both the normal and abnormal data allows the system to be tight on false positive and false negatives as both the normal and abnormal data needs to be

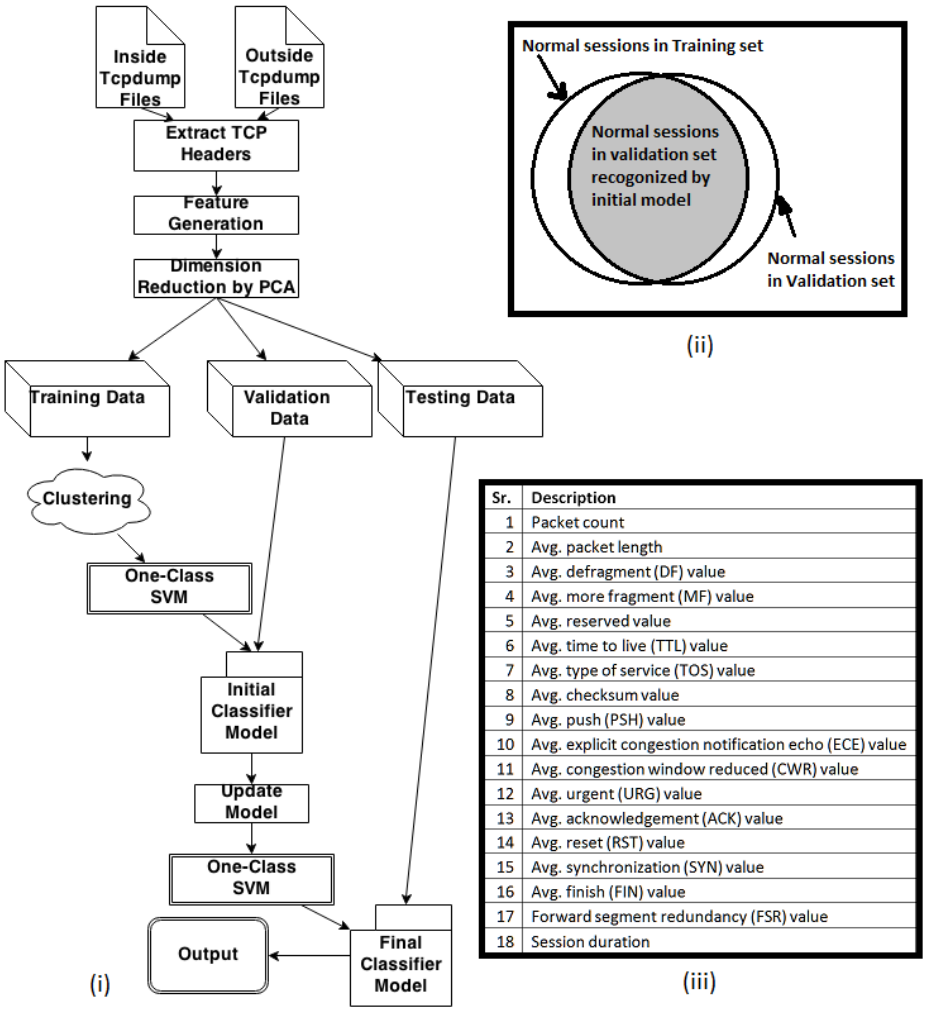
modeled; but it puts the limitation in modeling the abnormal patterns. Similarly, using only normal patterns allows the system to model the boundaries of normal data but due to unknown nature of the boundaries of abnormal data, it gives a possibility of false positive as well if it comes to overlap with the normal data. Thus, a proper tuning needs to be done for defining the threshold of the system to balance the number of true positives and false positives. Outlier mining has been a popular technique for finding intrusions [6, 11].

Network Analysis of Anomalous Traffic Events (NATE) [6] is a good example of an anomaly-based IDS using outlier mining. It is a low cost network IDS, which measures only TCP packets' headers information. It generates several concise descriptions using clustering to model normal network traffic behaviors. The measure of detection is the deviations of new TCP header data from the existing clusters, which allows for high speed network traffic monitoring once the normal behavior base is built. They performed a cluster analysis and found 7 such clusters that represent their normal TCP sessions. And Chebyshev's inequality was used to perform the boundary analysis. After that, Mahalanobis distance or Euclidean distance was used to detect intrusions in their test sessions. They have identified that false positives could be a potential problem with this approach, and hence, they have used significance level as a tuning parameter in response to decrease the false positives.

3 Proposed Method

Our proposed method's objective is to model the normal network data and then predict the abnormality based on the deviation from the normal patterns. Normal data may be of various types and may show various types of behaviors. Therefore, it is important to understand and analyze the behavior for each of these normal patterns and try to distinguish one another from abnormal data. In this end, we try to explore the TCP protocol in the network data and traverse through the TCP headers to understand the normal patterns. Exploring the TCP headers allows for high speed network traffic monitoring because it can be performed in real time and once the normal behavior base is built, it greatly reduces analysis complexity. Therefore, our design objective is to identify suitable tools to model the normal network data, and then implement the same to achieve an improved output in identifying the anomalous (attack) data. Our proposed method is partially inspired by NATE in defining TCP sessions, outlier mining based on TCP header information, and sample clustering. However, we have added substantial improvements like clustering quality analysis, one-class SVM, and model validation and updating. As a result, our method offers better results than those by NATE (as discussed later in Section 4). Our approach is to use basic TCP headers which can be used to detect the anomaly intrusions in real time. We tried modeling using a different model like clustering and one-class classifier and tried to vary over a range of different model parameters in order to achieve a stable and accurate model. Finally we came up with a model that has a high accuracy and a perfect recall along with a high degree of consistency.

The proposed architecture of the system is depicted in Figure 1(i). We take the tcpdump data files as input, analyze and process the TCP headers, and then finally predict the intrusions.The four steps involving in our method, namely (i) preprocessing, (ii) initial model creation, (iii) model validation and updating, and (iv) anomaly prediction are briefly descried below.



Sr.	Description
1	Packet count
2	Avg. packet length
3	Avg. defragment (DF) value
4	Avg. more fragment (MF) value
5	Avg. reserved value
6	Avg. time to live (TTL) value
7	Avg. type of service (TOS) value
8	Avg. checksum value
9	Avg. push (PSH) value
10	Avg. explicit congestion notification echo (ECE) value
11	Avg. congestion window reduced (CWR) value
12	Avg. urgent (URG) value
13	Avg. acknowledgement (ACK) value
14	Avg. reset (RST) value
15	Avg. synchronization (SYN) value
16	Avg. finish (FIN) value
17	Forward segment redundancy (FSR) value
18	Session duration

Fig. 1. (i) Overall work flow of the proposed intrusion detection system using outlier mining. (ii) Modeling normal sessions using both training and validation sets. (iii) Attributes used to represent a TCP session.

3.1 Preprocessing

From the input tcpdump data, we extract TCP headers of individual packets. Then, following the assumptions by Taylor and Alves-Foss [6], we define 'session', a group of packets having a unique combination of source IP and port number to destination IP and port number. This definition varies from the standard one practised in network traffic, which is one complete TCP connection beginning with SYN flags from source, response of SYN/ACK flags from destination and ending along with FIN flag from source, FIN/ACK flags from destination. The reason for defining our session is due to the fact that it is very difficult to extract such independent connections as many connection do not complete until more than one FIN flags are sent out. Since it is technically difficult and time consuming to capture such activities, instead, we collect and analyze the aggregated traffic between the unique pairs of senders and receivers. This allows for a quick and real-time analysis of the network traffic. The important attributes in an individual TCP packet's header include its packet length, DF, MF, and reserved flags, time-to-live (TTL), type of service (TOS), checksum, and all 8 TCP flags. Once a session is defined, the number of packets in it is counted and the average values of the above attributes from the session's packets are computed. In addition, the attribute FSR (forward segment redundancy) is derived as below. FSR shows the percentage of connection control flags in total packets of the session and observed to have high values for abnormal traffics.

$$\text{FSR} = (\sum \text{FIN} + \sum \text{SYN} + \sum \text{RST}) \ / \ \text{packet count} \tag{1}$$

The resultant feature vector containing 18 attributes, shown in Figure 1(iii), represents a session's behavior. Since the values of attributes vary significantly from one attribute to another, we use a natural logarithmic transformation as a form of normalization. In order to avoid negative infinite values caused by logarithm on zeros, we modify transformation function as: $normalize(x) = \log(1+x)$. After that, we apply principal component analysis (PCA) technique to reduce the dimensionality of our feature vectors. The first c number of principal components are taken. (c is an empirically determined integer.)

3.2 Initial Model Creation

We split the whole dataset into 3 portions as follows:

1. **Training set:** contains normal sessions only; used for initial model creation.
2. **Validation set:** contains both normal and attack sessions; used for model update after removing attack zone from the validation set.
3. **Testing set:** containing both normal and attack sessions; for prediction.

The normal sessions, represented by c-dimensional feature vectors, in the training set are divided into clusters by means of k-means clustering in order to model the overall behaviors of the normal sessions. Different values of k are tried, and the one that gives the highest average silhouette value is taken. Since the

k-means algorithm is affected by first initialized center values, prior to deciding with k, we make sure we get the reliable value of k, by executing the clustering process multiple times for each value of k. Then, Euclidean distance is used to measure the distances from k cluster centers to each training data point. From this, new k-dimensional feature vectors, each containing k distance values, are constructed. Then, these new feature vectors in the training set are fed to one-class SVM, resulting in the "initial classifier model".

3.3 Model Validation and Updating

Now, a similar k-dimensional feature vector is also constructed for each of the normal sessions in the validation set by calculating its Euclidean distances to the k cluster centers obtained before. Then, in order to validate the initial model, each validation set's feature vector is sent to the initial classifier model for classification. Those which are misclassified (as attacks) are removed from the data pool, as they do not agree with the initial model defined by the normal sessions in the training data. Finally, the original c-dimensional feature vectors of the correctly classified normal session in the validation set (depicted as the gray area in Figure 1(ii)) used to build another one-class SVM model. This results in the "final classifier model", which reflects both the training and validation sets.

3.4 Anomaly Prediction

When a new unknown TCP session is to be classified, we first need to construct its c-dimensional feature vector. Then, it is sent to the one-class SVM of the final classifier model in order to predict whether it is a normal or an attack session.

4 Experimental Results

4.1 Dataset Used

We use the publicly available MIT DARPA '99 dataset [7]. It has been one of the well-known benchmarks for IDS for a long time. One drawback with DARPA dataset concerns with its simulated nature as pointed out by McHugh [8]. However, we chose to proceed with this dataset because our results would be comparable with others when we use a standard and recognized dataset. We use the three weeks of data starting from third, fourth and fifth. The third week of dataset comprises of only normal packets and hence is used for training purpose. While fourth and fifth week's datasets, containing both normal and attack packets, are used for validation and testing (evaluation) purposes respectively. The different types of TCP attacks included in DARPA '99 dataset are Portsweep, Satan, Neptune, Mailbomb, nmap, etc.

In our experiment, we look at the distributions of of the numbers of sessions over the three weeks' data. Based on the assumption presented by Taylor and Alves-Foss [6], we similarly define our session. The number of normal and attack sessions in training, validation, and testing sets are shown in Table 1.

Table 1. Statistics of training, validation, and testing sets.

Dataset	DARPA'99	#Files	Total Size (GB)	#Normal Sessions	#Attack Sessions
Training	week 3	16	6.46	1,318,656	0
Validation	week 5	10	4.85	1,232,356	159,929
Testing	week 4	9	2.80	639,326	2,304

4.2 Results and Discussions

Our first objective is to model the normal sessions, for which we begin with performing PCA on the features of all three sets in order to reduce the dimensionality of our feature set. Here, we choose to proceed with the first $c = 8$ principal components out of 18 because those 8 covers above 99.6% of variance of the data. Now, in order to model the normal sessions in the training set, we apply k-means clustering to the resultant 8-dimensional feature vector set. The best k is chosen based on the highest average silhouette value obtained, which is 0.805 when $k = 3$. Once clustering of the normal sessions in the training set is done, we construct its 3-dimensional feature vectors to build the initial model using one-class SVM. Then, we feed the similar 3-dimensional feature vectors from the validation set to the initial model in order to perform a validation. We would like to achieve a clear distinction between the normal sessions and the attack ones. Thus, we select the one-class SVM parameters that allows us to achieve the maximum number of true negatives (normal sessions). After that, we use only the true negative in the validation set in order to build the final model using one-class SVM. Figure 2 shows the distribution of the validation set containing normal and attack sessions.

Finally, we feed the testing set, to the final model for prediction. We tried building our final model using different kernel types in one-class SVM. Among different kernel used, radial basis kernel (RBF) gave us the best results as shown in Table 2, with a perfect recall of 100% and a high accuracy of 91.85%. A relatively high false positive rate incurred (8.18%) can be attributed to the underlying similarities between the normal and the attack sessions. The reason for RBF being stand out amongst other kernels is that, RBF kernel in support vector algorithm automatically determines centers, weights and threshold such as to minimize an upper bound on the expected test error.

Table 2. Results using different kernel types in one-class SVM. The best results, which are obtained with radial basis kernel, are highlighted.

	Kernel Type			
	Linear	Polynomial (degree=3)	Sigmoid	**RBF**
True Positive (TP)	2,205	2,166	2,285	**2,304**
True Negative (TN)	502,094	515,047	532,681	**587,033**
False Positive (FP)	137,232	124,279	106,645	**52,293**
False Negative (FN)	99	138	19	**0**

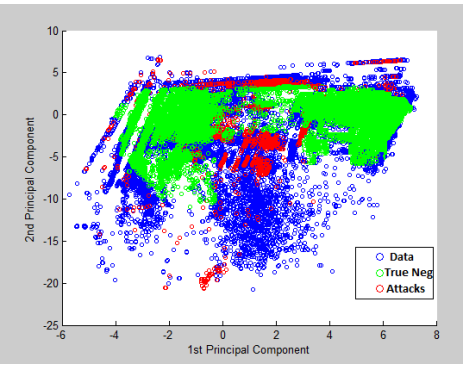


Fig. 2. Modeling true negatives detected in the validation set.

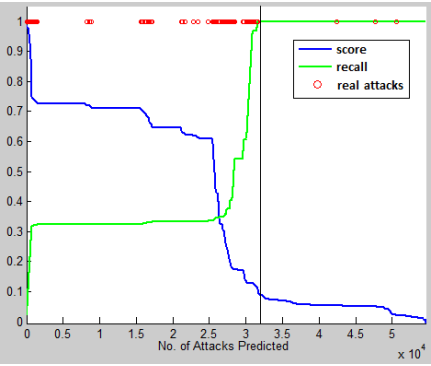


Fig. 3. Positions of 'real attacks' among all predicted attacks ordered by their 'prediction scores' in descending order with their corresponding 'recall' values.

Evaluation Criteria: Using radial basis kernel (RBF), we achieve the best results with a perfect recall of 100% (because there are no false negatives or dismissals) and a high accuracy of 91.85%. A relatively high false positive rate incurred (8.18%) can be attributed to the underlying similarities between the normal and the attack sessions.

$$\text{recall} = \text{TP}/(\text{TP} + \text{FN}) \tag{2}$$

$$\text{face positive rate} = \text{FP}/(\text{TN} + \text{FP}) \tag{3}$$

$$\text{accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}) \tag{4}$$

Comparison with NATE: In the NATE method presented by Taylor and Alves-Foss [6], among the 4 selected types of attacks explored, they were able to detect the Portsweep, Satan, and Neptune attacks, but not Mailbomb (ref. Tables 4 and 5 in [6]). That means their recall is less than 100%. In our case, we are able to detect all attacks presented in the 4th week of DARPA '99 dataset with a perfect recall of 100%. Our model overcomes the problem of NATE, which requires absolutely normal packets for normal session modeling. Since we have used the validation set to cross-check our normal session modeling, our final model reflects both the normal session and the attack session (by means of using the negative space).

Reducing False Positives: Apart from taking the default labels given by the one-class SVM classifier implemented with LIBSVM, we can also take the probabilistic or decision values (prediction scores) as output. Figure 3 shows the positions of real attacks (true positives) among all predicted attacks (predicted positives) ordered by their prediction scores, normalized into the range of 0 to 1, in descending order. It can be observed that among the 54,597 predicted positives, after ∼31,700 highest scoring ones (the region after the black vertical

line), the recall is already 99.87%, with only 3 true positives remain undetected. That point corresponds to the prediction score of ∼0.1. Thus, by sacrificing a very small fraction in recall value, we can significantly reduce the false positive rate and substantially improve the accuracy. By setting the score threshold as 0.1 and taking only the sessions whose scores are ≥ 0.1, we can achieve: TP=2,301; TN=609,913; FP=29,413; FN=3; recall=99.87%; false positive rate=4.60%; and accuracy=95.42%.

5 Conclusion and Future Work

In this research work, we have been able to model the normal sessions and detect the attack sessions derived from '99 DARPA dataset. Our work provides a practical solution for IDS based on outlier mining method on TCP headers. In today's world of increased network-based security threats, it enables network security professionals to analyze the TCP header information and perform anomaly detection in fast incoming traffic in a real-time manner.

For future work, apart from using basic TCP header information, we look forward to using derived information from the connection analysis that might help us to further reduce the false positive rate. And we also look to compare our results with ALAD/PHAD's.

References

1. Suthaharan, S., Panchagnula, T.: Relevance feature selection with data cleaning for intrusion detection system. In: Proc. 2012 IEEE SECon. (2012) 1–6
2. Zhang, X., Jia, L., Shi, H., Tang, Z., Wang, X.: The application of machine learning methods to intrusion detection. In: Proc. 2012 S-CET. (2012) 1–4
3. Kumar, M., Hanumanthappa, M., Kumar, T.V.S.: Intrusion detection system using decision tree algorithm. In: Proc. 14th IEEE ICCT. (2012) 629–634
4. Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.A.: A detailed analysis of the KDD CUP 99 data set. In: Proc. 2nd IEEE CISDA. (2009) 53–58
5. Mahoney, M.V., Chan, P.K.: Learning nonstationary models of normal network traffic for detecting novel attacks. In: Proc. 8th ACM KDD. (2002) 376–385
6. Taylor, C., Alves-Foss, J.: NATE – network analysis of anomalous traffic events, a low-cost approach. In: Proc. 2001 NSPW. (2001) 89–96
7. Mahoney, M.V., Chan, P.K.: An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In: Proc. 6th RAID. (2003) 220–237
8. McHugh, J.: Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. ACM Transactions on Information System Security **3** (2000) 262–294
9. Gharibian, F., Ghorbani, A.: Comparative study of supervised machine learning techniques for intrusion detection. In: Proc. 5th CNSR. (2007) 350–358
10. Sarvari, H., Keikha, M.M.: Improving the accuracy of intrusion detection systems by using the combination of machine learning approaches. In: Proc. 2010 SoCPaR. (2010) 334–337
11. Hofmeyr, S.A., Forrest, S.A.: Architecture for an artificial immune system. Evolutionary Computation **8** (2000) 443–473