Online Anomaly Detection by Using N-gram Model and Growing Hierarchical Self-Organizing Maps

Mikhail Zolotukhin, Timo Hämäläinen and Antti Juvonen
Department of Mathematical Information Technology, University of Jyväskylä,
Jyvväskylä, FI-40014, Finland,
Email: {mikhail.m.zolotukhin, timo.t.hamalainen, antti.k.a.juvonen}@jyu.fi

***Abstract*—In this research, online detection of anomalous HTTP requests is carried out with Growing Hierarchical Self-Organizing Maps (GHSOMs). By applying an n-gram model to HTTP requests from network logs, feature matrices are formed. GHSOMs are then used to analyze these matrices and detect anomalous requests among new requests received by the web-server. The system proposed is self-adaptive and allows detection of online malicious attacks in the case of continuously updated web-applications. The method is tested with network logs, which include normal and intrusive requests. Almost all anomalous requests from these logs are detected while keeping the false positive rate at a very low level.**

***Index Terms*—Data mining, intrusion detection, anomaly detection, n-gram, growing hierarchical self-organizing map.**

I. Introduction

The use of computer technologies, both for work and personal use, is growing with time. Unfortunately, computer networks and systems are often vulnerable to different forms of attacks [1]. Web-servers and web-based applications are one of the most popular attack targets. As a rule, users of web-servers and web-based applications request and send information in the form of queries, which in HTTP traffic are strings containing a set of parameters with some values. It is possible to manipulate these queries and create requests which can corrupt the server or collect confidential information [2].

Security of web-servers and web-based applications can be provided by means of Intrusion Detection Systems (IDS). IDS gathers data from the system under inspection, stores this data to logfiles, analyzes the logfiles to detect suspicious activities and determines a suitable response to these activities [3]. There are many kinds of IDS architectures, and they continue to evolve with time. IDSs can differ in audit source location, detection method, behavior on detection, usage frequency, etc [4], [5]. There are two basic approaches for detecting intrusions from the network data: misuse detection and anomaly detection [6], [7]. In the misuse detection approach, the IDS scans the computer system for predefined attack signatures. This approach is usually accurate which makes it successful in commercial intrusion detection [7]. However, the misuse detection approach cannot detect attacks for which it has not been programmed, and, therefore it is likely to ignore all new types of attack if the system is not kept up to date with the latest intrusions. The anomaly detection approach learns the features of event patterns with normal behavior, and by observing patterns that deviate from the established norms (anomalies), detects intrusions that have occurred. Systems which use anomaly detection approach are modeled according to normal behavior and, therefore, are able to detect zero-day attacks. However, the number of false alerts will probably increase because not all anomalies are intrusions.

Different machine-learning based techniques can be applied to solve the problem of anomaly detection and include, for example, Decision Trees (DTs), Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), etc. In this study, we use the approach based on Growing Hierarchical Self-Organizing Maps (GHSOMs). Simple Self-Organizing Map (SOM) is a neural network model based on unsupervised learning. It was proposed by Kohonen for analyzing and visualizing high dimensional data [8]. SOMs are able to discover knowledge in a data base, extract relevant information, detect inherent structures in high-dimensional data and map these data into a two-dimensional representation space [9]. Despite the fact that approach based on self-organizing maps has shown effectiveness at detecting intrusions [10], [11], it has two main drawbacks: its static architecture and the lack of representation of hierarchical relations. A Growing Hierarchical SOM (GHSOM) can solve these difficulties [12]. This neural network consists of several SOMs structured in layers, the number of which together with the number of neurons in each map and maps in each layer are determined during the unsupervised learning process. Thus, the structure of the GHSOM is automatically adapted to the structure of the data.

The GHSOM approach looks promising for solving network intrusion detection problems. For example, in [13], a GHSOM model with a metric which combines both numerical and symbolic data is proposed for detecting network intrusions. The IDS based on this model detects anomalies by classifying IP connections into normal and anomalous connection records, and into a type of attack if in case of anomalies. Another adaptive GHSOM based approach is proposed in [14]. The suggested GHSOM adapts online to changes in the input data over time by using the following enhancements: enhanced threshold-based training, dynamic input normalization, feedback-based quantization error threshold adaptation, and prediction confidence filtering and forwarding. A study [15] has investigated how to apply GHSOM for filtering intrusion

978-1-4577-1379-8/12/$26.00

detection alarms. GHSOM clusters these alarms in a way that helps network administrators in making decisions about true and false alarms.

In this study, our aim is to detect intrusive HTTP requests by applying an approach that is based on adaptive growing hierarchical self-organizing maps. The remainder of this paper is organized as follows. Section II describes process of data acquisition and feature extraction from network logs. Section III describes how to apply adaptive GHSOM for detecting anomalies. Experimental results are presented in Section IV. Section V concludes the paper.

II. System model

By analyzing HTTP requests, we focus here on detection of intrusions. Such requests can involve some parameter changes which creates opportunities to introduce malicious attacks. We do not focus on parameterless static requests, since it is not possible to inject code via static requests unless there are major deficiencies in the HTTP server itself. Dynamic requests, which are handled by the Web applications of the service, are of most interest. We concentrate on URIs of only such requests. A URI of a typical dynamic request can be expressed as a composition of the path to a desired resource and a query string (identified by a leading '?' character), which is used to pass parameters to the referenced resource:

```
/resource?param1=value1&param2=value2
```

Let us assume that most of the requests, coming to the HTTP server are normal, i.e. use legitimate features of the service, but that some of requests obtained are intrusions.

An n-gram model can be applied to extract features from each request URI. An n-gram is a sub-sequence of n overlapping items (characters, letters, words, etc) from a given sequence. N-gram models are widely used in statistical natural language processing [16] and speech recognition [17].

The n-gram character model is applied to transform each HTTP request to a sequence of n-characters. After that, ASCII codes of characters are used to represent a sequence of n-characters as a sequence of arrays, each of which contains n decimal ASCII codes. Then the frequency vector is built by counting the number of occurrences of n-character corresponding to each array in the analyzed request. The length of the frequency vector is 256^n, because every byte can be represented by an ASCII value between 0 and 255. In the previous URI example, the following sequence of decimal ASCII pairs can be obtained: $[47, 114]$, $[114, 101]$, $[101, 115]$, $[115, 111]$, $\ldots$, $[101, 50]$. The corresponding 256^2 vector is built by counting the number of occurrences of each of such pair. For example, the entry in location $(256 \times 61 + 118)$ in this vector contains the value 2 since the pair $[61, 118]$, which corresponds to pair '=v', can be seen twice. Thus, each request is transformed to 256^n numeric vector, and the matrix consisting of these vectors can be analyzed to find anomalies.

III. Method

Here an approach based on growing hierarchical self-organizing maps is used to find anomalies in feature matrices. Self-organizing map is an unsupervised, competitive learning algorithm that reduces the dimensions of data by mapping these data onto a set of units set up in a much lower-dimensional space [8], [9]. SOM is formed from a regular grid of neurones, each of which is fully connected to the input layer. The neurons are connected to adjacent neurons by a neighborhood relation dictating the structure of the map. Associated with it, each neuron of the SOM has a prototype (weight) vector dimension, which is equal to the dimension of the input vectors, and therefore each neuron has two positions: one in the input space (the prototype vector) and another in the output space (on the map grid). Thus, SOM is a vector projection method defining a nonlinear projection from the input space to a lower-dimensional output space. On the other hand, during the training the prototype vectors are modified in such way that they follow the probability density of the input data. Hence, the algorithm allows not only to compress high-dimensional data, but also to create a network that stores information in such a way that any topological relationships within the data set are maintained,

SOMs are widely applied for visualizing low-dimensional views of high-dimensional data. However, there are some drawbacks to the traditional SOM approach, caused by the static nature of the model architecture. The size and dimensionality of the SOM model is fixed prior to the training process and there is no systematic method for identifying an optimal SOM configuration. In addition, traditional SOM can not represent hierarchical relations that might be present in the data.

To resolve these limitations, growing hierarchical self-organizing maps was introduced. GHSOM is a multi-layered hierarchical architecture which adapts its structure to the input data. The GHSOM architecture starts with a 2×2 map at the first layer trained according to a SOM training algorithm [8], [9]. This map can grow in size until it achieves an improvement in the quality of data representation. The growth of the map in the first layer and the maps in the next layers are controlled by the use of quantization errors:

$$e_i = \sum_{x_j \in C_i} ||w_i - x_j|| \,, \tag{1}$$

where e_i is the quantization error for the i-th node, C_i is the set of input vectors x_j projected to the i-th node and w_i is the weight vector of the i-th node. Quantization error E_m of map m is defined as

$$E_m = \frac{1}{|U_m|} \sum_{i \in U_m} e_i, \tag{2}$$

where U_m is the subset of the m-th map nodes onto which data is mapped, and $|U_m|$ is the number of these nodes of the m-th map.

If E_m reaches certain fraction α_1 of the quantization error e_p of the corresponding parent unit p in the upper layer, the growing process is stopped. Quantization error e_0 of the parent

node for the SOM in the first layer can be found as follows:

$$e_0 = \sum_{x_j \in C_0} ||w_0 - x_j||\,, \tag{3}$$

where C_0 is the set of all input vectors mapped to the GHSOM and w_0 denotes the mean of the input data. Parameter α_1 controls the breadth of the maps, and its value ranges from 0 to 1.

If $E_m > \alpha_1 e_u$, the map should be extended. First, the e-th node with the maximal value of quantization error is found, and the most dissimilar neighboring node s is selected according to

$$s = \max_j(||w_e - w_j||), \text{ for } w_j \in N_e, \tag{4}$$

where w_j is the weight vector of the e-th node, N_e is the set of neighboring nodes of the e-th node, and w_i is the weight vector of the neighboring node in set N_e. A new row or column of nodes is placed in between nodes e and s. The weight vectors of newly added nodes are initialized with the mean of their corresponding neighbors.

Once the SOM's growth process is completed, every node of this SOM has to be checked for fulfillment of the global stopping criterion [12]:

$$e_i < \alpha_2 e_0, \tag{5}$$

where $\alpha_2 \in (0, 1)$ is a parameter which controls the hierarchical growth of GHSOM. Nodes not satisfying this criterion (5) and therefore representing a set of too diverse input vectors, are expanded to form a new map at a subsequent layer of the hierarchy. Similar to the creation of the first layer SOM, a new map of initially 2×2 nodes is created. This maps weight vectors are initialized to mirror the orientation of the neighboring units of its parent. For this reason, we can choose to set four new nodes to the means of the parent and its neighbors in the respective directions [18]. The newly added map is trained by using the input vectors, which are mapped onto the node just expanded, i.e., the subset of the data space mapped onto its parent. This new map will again continue to grow and the whole process is repeated for the subsequent layers until the global stopping criterion given in (5) is met by all nodes. Thus, an ideal topology of the GHSOM is formed unsupervised, based on the input data, and the hierarchal relationships in the data are discovered.

The method proposed here requires training the GHSOM before the detection of intrusions starts. Server logfiles, containing several thousands of HTTP requests, which are gathered from different web resources during several days or weeks, are used as a training set. These logs may include unknown anomalies and real attacks. The only condition is that the quantity of normal requests in the logs used must be significantly greater than the number of real intrusions and anomalous requests. HTTP requests from these logs are transformed to a feature matrix by applying the n-gram model. After that, GHSOM should be constructed and trained, based on this matrix.

Large web services can include several different web resources. These resources are accessed by requests, which can greatly differ by length, number of attributes, type of attributes, etc. In this case, constructing only one GHSOM for numerous web resources is most likely to fail considering that requests to the least popular web-resources can be classified as anomalous. The most intuitive way to solve this problem is to construct a GHSOM for each web-resource. Thus, requests to different web resources will be mapped to different GHSOMs, the number of GHSOMs corresponds to the number of different resources of the web server. Each GHSOM is created and initialized with one layer consisting of four nodes. One by one is trained using the corresponding request to that specific web-resource. Every GHSOM can grow in size by adding new rows and columns or by adding a new map of four nodes at a lower layer providing a further detailed representation of data.

When the training is completed, the system is able to detect intrusions among HTTP requests sent to the server. The new request is first transformed to a frequency vector by applying the n-gram model. After that it goes to the corresponding GHSOM, and in the first layer of the GHSOM this new request is mapped to a neuron for which the distance between this neuron's weight vector and new request is minimal. Such neuron is called the Best Matching Unit (BMU) for the request. If this best matching neuron has a child SOM in the next layer, then the BMU for the request should also be found among the neurons of this SOM.

Let us assume that this new request is mapped to a neuron. If this neuron already contains l other requests which are mapped to this neuron during the training phase, we can denote the distances between the neuron's weight vector and these l requests as $e_1, e_2, \ldots, e_l$. Let us assume that values of these distances are distributed uniformly. In this case, we can estimate maximum τ of continuous uniformly distributed variable as follows [19]:

$$\tau = \frac{l+1}{l} \max_l\{e_1, e_2, \ldots, e_l\}. \tag{6}$$

Obtained value τ can be used as the threshold value for the neuron considered, and the new request is classified as an intrusion if distance between this request and the neuron's weight vector is greater than τ.

We will also calculate U^*-matrices [20] for each individual SOM in GHSOMs constructed. U^*-matrix presents a combined visualization of distance relationships and density structures of a high-dimensional data space. This matrix has the same size as the grid of the corresponding SOM and can be calculated based on U-matrix and P-matrix.

U-matrix represents distance relationships of requests mapped to an SOM [21]. The value of the i-th element of U-matrix is the average distance of the i-th neuron weight vector w_i to the weight vectors of its immediate neighbors. Thus, the i-th element of U-matrix $U(i)$ is calculated as follows:

$$U(i) = \frac{1}{n_i} \sum_{j \in N_i} D(w_i, w_j), \tag{7}$$

where $n_i = |N_i|$ is the number of neurons in the neighborhood N_i of the i-th neuron, and D is a distance function which for example can be Euclidean distance.

P-matrix allows to obtain a visualization of density structures of the high-dimensional data space [22]. The i-th element of P-matrix is a measure of the density of data points in the vicinity of the weight vector of the i-th neuron:

$$P(i) = |\{x \in X | D(x, w_i) < r\}|, \tag{8}$$

where X is the set of requests mapped to the SOM considered and radius r is some positive real number. A display of all P-matrix elements on top of the SOM grid is called a P-matrix. In fact, the value of $P(i)$ is the number of data points within a hypersphere of radius r. The radius r should be chosen such that $P(i)$ approximates the probability density function of the data points. This radius can be found as the Pareto radius [23]:

$$r = \frac{1}{2}\chi^2_d(p_u), \tag{9}$$

where χ^2_d is the Chi-square cumulative distribution function for d degrees of freedom and $p_u = 20.13\%$ of requests' number in the data set X. The only condition is that all points in X must follow a multivariate mutually independent Gaussian standard normal density distribution (MMI). It can be enforced by different preprocessing methods such as principal component analysis, standardization and other transformations.

The U^*-matrix which is a combination of a U-matrix and a P-matrix presents a combination of distance relationships and density relationships and can give an appropriate clustering. The i-th element of U^*-matrix is equal to the $U(i)$ multiplied with the probability that the local density, which is measured by $P(i)$, is low. Thus $U^*(i)$ can be calculated as follows:

$$U^*(i) = U(i)\frac{|p \in P | p > P(i)|}{|p \in P|}, \tag{10}$$

i.e. if the local data density is low $U^*(i) \approx U(i)$ (this happens at the presumed border of clusters) and if the data density is high, then $U^*(i) \approx 0$ (this is in the central region of clusters).

Since we assumed that most of the requests are normal, intrusions can not form big clusters but will be mapped to neurons which are located on cluster borders. Thus, a neuron which has a high value of the U^*-matrix element contains anomalies. In this paper, the following criterion for finding such neurons is used: if the difference between the $U^*(i)$ and $U^*_{average}(i)$ (average value of all elements of U^*-matrix) is greater than difference between the $U^*_{average}(i)$ and minimal value of U^*-matrix, then the i-th neuron is classified as "anomalous". Otherwise this neuron is classified as "normal". If a neuron of GHSOM has a child SOM, then all the neurons of this child SOM should be checked also whether they are "normal" or "anomalous", by calculating a new U^*-matrix for this SOM.

In this manner, the following criteria are used to determine whether the new request is allowable or intrusive:

- If the distance between a new request and its BMU weight vector is greater than the threshold value, then this request is intrusion, otherwise it is classified as normal;
- If the neuron which is the BMU for the new request is classified as "anomalous", then this request is intrusion, otherwise it is classified as normal.

The approach based on constructing a GHSOM for each individual web resource can be used only if the number of HTTP requests to each web-resource is large enough to analyze the normal behavior of users. Sometimes, attackers try to access the data stored on servers or to harm the system by using holes in security of less popular web resources, for which it is difficult to define which requests are "normal". In addition, constructing a GHSOM for each individual web resource is disadvantageous in terms of time and memory. One approach which can solve this problem is to combine HTTP requests having a similar structure to one group, i.e. for requests which have a similar length, number of parameters, type of parameters, etc. only one GHSOM is constructed regardless of whether these requests belong to one resource or not. This allows to reduce the number of GHSOMs constructed and increase the number of requests mapped to each GHSOM.

Furthermore, web applications are highly dynamic and change on a regular basis, which can cause noticeable changes in the HTTP requests which are sent to the web server. This can lead to a situation where all new allowable requests will be classified as intrusions. For this reason, GHSOMs should be retrained after a certain period of time to be capable to classify new requests.

Let us assume that the number of requests sent to the web-server for this particular period is much less than the number of requests in the training set. We update the training set by replacing the first requests from this set by requests obtained during the period. After that, GHSOMs are retrained by using the resulting training set. During the update phase the structure of the GHSOMs can be modified. The update of the GHSOMs structure starts from the current structure. After updating GHSOMs, parameters τ^i and matrices U, P and U^* should be recalculated. The update phase can occur independently from anomaly detection. During retraining, the requests obtained are classified using the old GHSOM. Once the GHSOM retraining is completed, the classification of new requests continues with the updated GHSOM.

Countermeasures are necessary against attackers who try to affect the training set by flooding the web-server with a large number of intrusions. This can be enforced for example by allowing a client (one IP address) to replace a configurable number of HTTP requests in the training set per time slot. In order to address the threat of botnets it is also possible to restrict the globally allowed replacements per time slot independent of the IP addresses.

IV. Simulation Results

The proposed method is tested using logs acquired from a large real-life web-service. These logs contain mostly normal traffic, but they also include intrusive HTTP requests. The logfiles are acquired from several Apache servers and stored in a combined log format [24]. The logs contain requests from multiple web-resources. As mentioned already, we focus

on finding anomalies from dynamic requests because these requests are used by web applications, which are run behind the HTTP server. Thus, the requests without parameters are ignored.

In our simulations, requests to twenty most popular web-resources of the web-service are analyzed. The training set, which contains 20 000 requests, is created at the beginning. After training the GHSOMs, new requests are chosen from logfiles and classified by these GHSOMs one by one to test the technique proposed. The number of testing requests is equal to 200 000, and 9723 (4.86 %) of those requests are intrusive. During the testing, GHSOMs are updated every 4 000 requests. The performance of proposed technique is evaluated based on the following characteristics:

- True positive rate, ratio of the number of correctly detected intrusions to the total number of intrusions in the testing set;
- False positive rate, ratio of the number of normal requests detected as intrusionsIn the first simulation to the total number of normal requests in the testing set;
- True negative rate, ratio of the number of correctly classified normal requests to the total number of normal requests in the testing set;
- False negative rate, ratio of the number of intrusions detected as normal requests to the total number of intrusions in the testing set;
- Accuracy, ratio of the total number of correctly classified requests to the total number of requests in the testing set;
- Precision, ratio of the number of correctly detected intrusions to the number of requests classified as intrusions in the testing set.

In the first simulation, a GHSOM is constructed for each unique web resource. Since the requests are related to twenty different web-resources, twenty GHSOMs are constructed during the training phase. N-gram character models are used to transform requests from the training set to numeric matrices: 1-gram, 2-gram and 3-gram.

After the GHSOMs training is completed, U-matrices, P-matrices and U^*-matrices are calculated for each GHSOM. In Figure 1, the U^*-matrix is shown for the GHSOM corresponding to a web-resource which allows users to explore a project from the project list after 2-gram model applied. A request to this web-resource may contain the following parameters: project identification number and a page of the project description, or just a project identification number. Thus, all requests have one or two attributes which can be used by attackers to inject malignant code. As one can notice from Figure 1, there are two big clusters corresponding to the requests in which one or two attributes are used. The nodes on one of the map corner are classified as "anomalous". The technique proposed does not allow us to define intrusion types, but we can check manually the nodes which have been classified as "anomalous" and make sure that requests mapped to those nodes are real intrusions: SQL injections, cross-site scripting, double encoding attacks, etc., as shown in Figure 1.

After the construction of the U^*-matrix, detection process

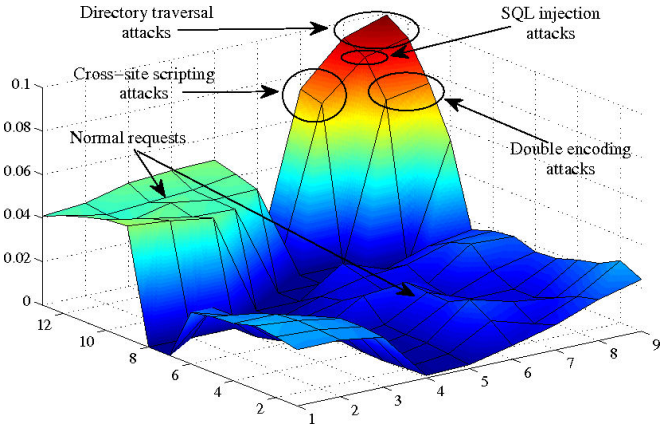

Fig. 1. U^*-matrix of the GHSOM after the training stage (17-th resource).

is started. New requests are mapped to the GHSOMs one by one and classified as intrusions if the distance between a new request and its BMU weight vector is greater than the threshold value or if the node, which is the BMU for this new request, is anomalous. During the detection phase, GHSOMs are retrained periodically when a certain number of requests are processed. After GHSOMs have been updated, the threshold values for all nodes are modified and U^*-matrices are also recalculated. Requests which are used as the testing set in our simulation contain also other types of intrusions but lack those which are contained in the training set. Since the logs contain only a few intrusion types, we generate intrusions of other types manually and add them to the testing set.

The summary of the simulation results is presented in Table I. As one can see 1-gram model does not allow to obtain good results. However, almost all real attacks are classified correctly as intrusions when 2-gram model is applied to transform HTTP requests to numeric vectors. When requests are processed by using 3-gram model, all intrusions are detected. At the same time, for 2-gram and 3-rgam models false positive rate is about 0.01% on average, which means that the number of false alarms is very low. The accuracy of the method is close to one hundred percent when applying 2-gram or 3-gram model. As one can notice, applying 3-gram model does not give much advantage in terms of accuracy, but on the other hand requires large amount of memory for saving numeric vectors obtained, in addition to slower processing of the data. Thus, the use of 2-gram model looks more beneficial since it allows to classify HTTP requests accurately without using huge amount of computing resources.

TABLE I
THE SIMULATION RESULTS FOR DIFFERENT WEB-RESOURCES.

Model	True positive rate	False positive rate	True negative rate	False negative rate	Accuracy	Precision
1-gram	89.0 %	0.01 %	99.9 %	11.0 %	99.4 %	99.5 %
2-gram	99.9 %	0.01 %	99.9 %	0.01 %	99.9 %	99.9 %
3-gram	100 %	0.01 %	99.9 %	0 %	99.9 %	99.9 %

Despite the fact that the accuracy of the method proposed is very high, this method requires to construct one GHSOM for each unique web resource, which is disadvantageous in terms of time and memory. As mentioned in the previous section, HTTP requests having a similar structure can be combined to one group. In this study, we will form such groups selected by the following criteria: total number of attributes, number of attributes with text values, number of attributes with numeric values, number of attributes which contain special symbols and average length of attributes values (short, medium or long). As

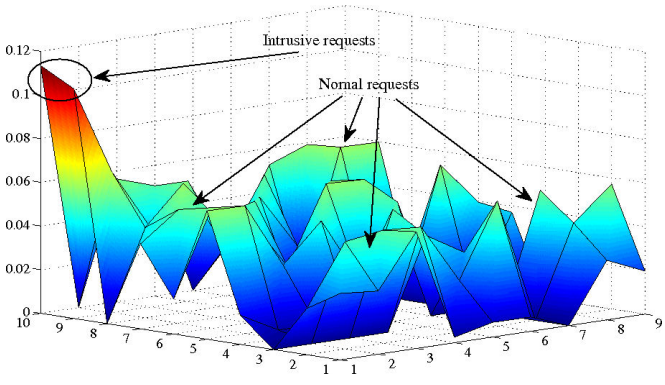

Fig. 2. U^*-matrix of the GHSOM after the training stage (2-nd group).

a result, we obtained seven different groups of HTTP requests in our example. For each group, one GHSOM is constructed during the training stage and 2-gram model is used to obtain numeric matrices from HTTP request sets.

In Figure 2, the U^*-matrix is shown for the GHSOM corresponding one such group. In this group, HTTP requests, which have one numeric attribute with a value of medium length, are combined. Anomalous requests are mapped to the peaks of the U^*-matrix and can be easily detected, as in the case of separating requests belonging to different web-resources. The summary of the simulation results is presented in Table II. As one can see, the accuracy of the method in this case remains very high.

TABLE II
THE SIMULATION RESULTS FOR DIFFERENT GROUPS OF HTTP REQUESTS.

Metric	True positive rate	False positive rate	True negative rate	False negative rate	Accuracy	Precision
Value	99.1 %	0.2 %	99.9 %	0.9 %	99.8 %	97.7 %

V. CONCLUSION

In this research, HTTP requests sent to the server are transformed to numeric vectors by applying an n-gram model, and adaptive growing hierarchical self-organizing maps are used to find anomalies among these vectors. The technique proposed is self-adaptive and allows to detect online HTTP attacks in the case of continuously updated web applications. The method is tested using logs which include normal and intrusive requests. As a result, almost all attacks from these logs are detected, and the number of false alarms remains very low.

REFERENCES

[1] S. Mukkamala, A. Sung. A comparative study of techniques for intrusion detection. Tools with Artificial Intelligence, 2003. Proc. 15th IEEE International Conference. pp. 570-577. Nov. 2003.

[2] A. Nguyen-Tuong, S. Guarnieri, D. Greene, J. Shirley, D. Evans. Automatically Hardening Web Applications Using Precise Tainting. 20th IFIP International Information Security Conference (SEC 2005) Chiba, Japan. 30 May - 1 Jun. 2005.

[3] S. Axelsson. Research in intrusion-detection systems: a survey. Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, Technical Report. pp. 98-117. Dec. 1998.

[4] A. Patcha, J.M. Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer Networks: The International Journal of Computer and Telecommunications Networking. Vol. 51, Issue 12. Aug. 2007.

[5] T. Verwoerd, R. HuntComputer. Intrusion detection techniques and approaches. Communications - COMCOM. Vol. 25, no. 15, pp. 1356-1365. 2002.

[6] R.A. Kemmerer, G. Vigna. Intrusion Detection: A Brief History and Overview. Computer, 35, pp. 27-30. Apr. 2002.

[7] D. Gollmann. Computer Security. Wiley, 2nd edition, 2006.

[8] T. Kohonen. Self-organizing map (3rd ed.). Berlin: Springer-Verlag. 2001.

[9] T. Kohonen. Self-organized formation of topologically correct feature maps. Biological cybernetics 43(1), pp.59-69. 1982.

[10] H. G. Kayacik, Z.-H. Nur, M. I. Heywood. A hierarchical SOM-based intrusion detection system. Engineering Applications of Artificial Intelligence, Vol. 20, 2007.

[11] D. Jiang, Y. Yang, M. Xia. Research on Intrusion Detection Based on an Improved SOM Neural Network. Proc. of the Fifth Intl Conference on Information Assurance and Security. 2009.

[12] A. Rauber, D. Merkl, M. Dittenbach. The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data. Neural Networks, IEEE Transactions. Vol. 13, Is. 6, pp. 1331-1341. Nov. 2002.

[13] E. J. Palomo, E. Domnguez, R. M. Luque, J. Muoz. A New GHSOM Model Applied to Network Security. Lecture Notes in Computer Science, Vol. 5163/2008, pp. 680-689, 2008.

[14] D. Ippoliti, Z. Xiaobo. An Adaptive Growing Hierarchical Self Organizing Map for Network Intrusion Detection. Computer Communications and Networks (ICCCN), Proc. 19th IEEE International Conference. pp. 1-7. Aug. 2010.

[15] M. Shehab, N. Mansour, A. Faour. Growing Hierarchical Self-Organizing Map for Filtering Intrusion Detection Alarms. Parallel Architectures, Algorithms, and Networks, I-SPAN 2008. International Symposium. pp. 167-172. May 2008.

[16] C. Y. Suen. n-Gram Statistics for Natural Language Understanding and Text Processing. Pattern Analysis and Machine Intelligence, IEEE Transactions. Vol. PAMI-1, Is. 2. pp. 164-172. 1979.

[17] T. Hirsimaki, J. Pylkkonen, M. Kurimo. Importance of High-Order N-Gram Models in Morph-Based Speech Recognition. Audio, Speech, and Language Processing, IEEE Transactions. Vol. 17, Is. 4. pp. 724-732. 2009.

[18] A. Chan, E. Pampalk. Growing hierarchical self organising map (ghsom) toolbox: visualisations and enhancements. In 9-th Int'l Conference Neural Information Processing, ICONIP 02, Vol. 5, pp. 2537-2541. 2002.

[19] R. W. Johnson. Estimating the Size of a Population. Teaching Statistics, Vol. 16, Is. 2, pp. 50-52. Jun. 1994.

[20] A. Ultsch. Clustering with SOM: U*C. Proc. Workshop on Self-Organizing Maps (WSOM 2005), Paris, France, pp. 75-82. 2005.

[21] A.Ultsch, H.P.Siemon. Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis. Proc. Intern. Neural Networks, Kluwer Academic Press, Paris, pp. 305-308. 1990.

[22] A.Ultsch. Maps for the Visualization of high-dimensional Data Spaces. Proc. WSOM, Kyushu, Japan, pp. 225-230. 2003.

[23] A. Ultsch. Pareto Density Estimation: A Density Estimation for Knowledge Discovery. Innovations in Classification, Data Science, and Information Systems - Proc. 27-th Annual Conference of the German Classification Society (GfKL) Berlin, Heidelberg, Springer. pp. 91-100. 2003.

[24] Apache 2.0 Documentation (2011). http://www.apache.org/.