# GROWING HIERARCHICAL SELF-ORGANISING MAPS FOR ONLINE ANOMALY DETECTION BY USING NETWORK LOGS

Mikhail Zolotukhin, Timo Hämäläinen and Antti Juvonen
*Department of Mathematical Information Technology, University of Jyväskylä, Jyväskylä, FI-40014, Finland*

Keywords: Intrusion Detection, Anomaly Detection, N-gram, Growing Hierarchical Self Organising Map, Data Mining.

Abstract: In modern networks HTTP clients request and send information using queries. Such queris are easy to manipulate to include malicious attacks which can allow attackers to corrupt a server or collect confidential information. In this study, the approach based on self-organizing maps is considered to detect such attacks. Feature matrices are obtained by applying n-gram model to extract features from HTTP requests contained in network logs. By learning on basis of these matrices, growing hierarchical self-organizing maps are constructed and by using these maps new requests received by the web-server are classified. The technique proposed allows to detect online HTTP attacks in the case of continuous updated web-applications. The algorithm proposed was tested using Logs, which were aquire acquired from a large real-life web-service and include normal and intrusive requests. As a result, almost all attacks from these logs have been detected, and the number of false alarms was very low at the same time.

## 1 INTRODUCTION

In modern society, 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 intrusions. Such intrusions are manually executed by a person or automatically with engineered software and can use legitimate system features as well as programming mistakes or system misconfigurations (Mukkamala and Sung, 2003). That is why the computer security becomes one of the most important issues when designing computer networks and systems.

One of the most popular attack targets are web-servers and web-based applications. Since web-servers are usually accessible through corporate firewalls, and web-based applications are often developed without following security rules, attacks which exploit web-servers or server extensions represent a significant portion of the total number of vulnerabilities. Usually, the users of web-servers and web-based applications request and send information using queries, which in HTTP traffic are strings containing set of parameters having some values. It is possible to manipulate these queries and create requests which can corrupt the server or collect confidential information (Nguyen-Tuong et al., 2005).

One means to ensure the security of web-servers and web-based applications is use of Intrusion Detection Systems (IDS). As a rule, IDS gathers data from the system under inspection, stores this data to logfiles, analyzes the logfiles to detect suspicious activities and determines suitable responses to these activities (Axelsson, 1998). There are a lot of diverse architectures of IDSs and they continue to evolve with time (Patcha and Park, 2007; Verwoerd and Hunt, 2002). IDSs can also differ in audit source location, detection method, behaviour on detection, usage frequency, etc.

There are two basic approaches for detecting intrusions from the network data: misuse detection and anomaly detection (Kemmerer and Vigna, 2002; Gollmann, 2006). In the case of 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 (Gollmann, 2006). However, misuse detection approach cannot detect attacks for which it has not been programmed, and, therefore it is prone 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 which form normal behaviour, and, by observing patterns that deviate from established norms (anomalies), detects when an intrusion has occurred. Thus, systems which use anomaly detection approach are mod-

Zolotukhin M., Hämäläinen T. and Juvonen A..
GROWING HIERARCHICAL SELF-ORGANISING MAPS FOR ONLINE ANOMALY DETECTION BY USING NETWORK LOGS.
DOI: 10.5220/0003936606330642
In *Proceedings of the 8th International Conference on Web Information Systems and Technologies* (WEBIST-2012), pages 633-642
ISBN: 978-989-8565-08-2

elled according to normal behaviour and, therefore, are able to detect zero-day attacks. However, the number of false alerts will probably be increased because not all anomalies are intrusions.

To solve the problem of anomaly detection different kinds of machine learning based techniques can be applied, for example, Decision Trees (DTs), Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), etc. As a rule, anomaly detection IDSs for web-servers are based on supervised learning: training the system by using a set of normal queries. On the contrary, unsupervised anomaly detection techniques do not need normal training data and therefore such techniques are the most usable.

In this study, we consider the approach based on Self-Organizing Maps (SOMs). A SOM is a based on unsupervised learning neural network model proposed by Kohonen for analyzing and visualizing high dimensional data (Kohonen, 1982). 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 (Kohonen, 2001). Despite the fact that the approach based on self-organizing maps has shown effectiveness at detecting intrusions (Kayacik et al., 2007; Jiang et al., 2009), it has two main drawbacks: the static architecture and the lack of representation of hierarchical relations. A Growing Hierarchical SOM (GHSOM) can solve these difficulties (Rauber et al., 2002). This neural network consists of several SOMs structured in layers, whose number of neurons, maps and layers are determined during the unsupervised learning process. Thus, the structure of the GHSOM is automatic adapted according to the structure of the data.

The GHSOM approach looks promising for solving network intrusions detection problem. In the study (Palomo et al., 2008), 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 or anomalous connection records, and the type of attack if they are anomalies. An adaptive GHSOM based approach is proposed in (Ippoliti and Xiaobo, 2010). 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. The study (Shehab et al., 2008) investigates applying GHSOM for filtering intrusion detection alarms. GHSOM clusters these alarms in a way that helps network administrators to make decisions about true or false alarms.

In this research we aim to detect anomalous HTTP requests by applying approach based on adaptive growing hierarchical self-organizing maps. The remainder of this paper is organized as follows. Section 2 describes process of data acquisition and feature extraction from network logs. In Section 3 we present classic SOM and GHSOM models. Section 4 describes applying adaptive GHSOM for detecting anomalies. Experimental results are presented in Section 5. Section 6 concludes this paper.

## 2 DATA MODEL

Let us consider some network activity logs of a large web-service of some HTTP server. Such log-files can include information about the user's IP address, time and timezone, the HTTP request including used resource and parameters, server response code, amount of data sent to the user, the web-page which was requested and used by a browser software. Here is an example of a single line from some Apache server log file, this information is stored in combined log format (Apache 2.0 Documentation, 2011):

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700]
"GET /resource?parameter1=value1&parameter2=
value2 HTTP/1.0"
200 2326 "http://www.example.com/start.html"
"Mozilla/4.08 [en] (Win98; I ;Nav)"
```

In this study, we focus on HTTP requests analysis. Such requests can contain some parameters changing which creates a possibility to include malicious attacks. We do not focus on static requests which do not contain any parameters because 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 more interesting in this study. Let us assume that most requests, which are coming to the HTTP server, are normal, i.e. use legitimate features of the service, but some obtained requests are intrusions.

All dynamic HTTP requests are analyzed to detect anomalous ones. The input to the detection process consists of an ordered set of HTTP requests. A request can be expressed as the composition of the path to the desired resource and a query string which is used to pass parameters to the referenced resource and identified by a leading '?' character.

To extract features from each request n-gram model is applied. N-gram models are widely used in statistical natural language processing (Suen, 1979) and speech recognition (Hirsimaki et al., 2009). A n-gram is a sub-sequence of *n* overlapping items (characters, letters, words, etc) from a given sequence. For example, 2-gram character model for the string '/resource?parameter1=value1&parameter2=value2' is '/r', 're', 'es', 'so', 'ou', 'ur', ..., 'lu', 'ue', 'e2'.

N-gram character model is applied to transform each HTTP request to the sequence of n-characters. Such sequences are used to construct a n-gram frequency vector, which expresses the frequency of every n-characters in the analyzed request. To obtain this vector, ASCII codes of characters are used to represent sequence of n-characters as sequence of arrays each of which contains *n* decimal ascii codes, and the frequency vector is built by counting the number of occurences of each such 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. For example, in the previous example the following sequence of decimal ASCII pairs can be obtained: $[47,114]$, $[114,101]$, $[101,115]$, $[115,111]$, $[111,117]$, $[117,114]$, ..., $[108,117]$, $[117,101]$, $[101,50]$. The corresponding $256^2$ vector is built by counting the number of occurences of each of such pair. For example, the entry in location $(256\times 61+118)$ in this vector contains the value equal to 2 since the pair $[61,118]$, which corresponds pair '=v', can be seen twice. Thus, each request is transformed to $256^n$ numeric vector. The matrix consisting of these vectors is called feature matrix and it can be analyzed to find anomalies.

## 3 BACKGROUND ON SOM AND GHSOM

In this study, adaptive growing hierarchical self-organizing maps are used to find anomalies in feature matrix. Traditional SOM model and growing hierarchical SOM model are briefly described in this section.

### 3.1 Self-organizing Maps

The 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 much lower dimensional space. This 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. Due to this fact SOMs are widely applied for visualizing the low-dimensional views of high-dimensional data.

SOM is formed form 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. The *i*-th neuron of the SOM has an associated with it *d*-dimensional prototype (weight) vector $w_i = [w_{i1}, w_{i2}, \ldots, w_{id}]$, where $d$ is equal to the dimension of the input vectors. 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 move so that they follow the probability density of the input data.

SOMs learn to classify data without supervision. At the beginning of learning the number of neurons, dimensions of the map grid, map lattice and shape should be determined. Before the training, initial values are given to the prototype vectors. The SOM is very robust with respect to the initialization, but properly accomplished initialization allows the algorithm to converge faster to a good solution. At each training step $t$, one sample vector $x(t)$ from the input data set is chosen randomly and a similarity measure (distance) is calculated between it and all the weight vectors $w_i(t)$ of the map. The unit having the shortest distance to the input vector is identified to be the best matching unit (BMU) for input $x(t)$. The index $c(t)$ of this best matching unit is identified. Next, the input is mapped to the location of the best matching unit and the prototype vectors of the SOM are updated so that the vector of the BMU and its topological neighbors are moved closer to the input vector in the input space:

$$w_i(t+1) = w_i(t) + \delta(t) N_{i,c(t)}(r(t))\,(x(t) - w_i(t))\,, \tag{1}$$

where $\delta(t)$ is the learning rate function and $N_{i,c(t)}(r(t))$ is the neighborhood kernel around the winner unit, which depends on neighborhood radius $r(t)$ and the distance between BMU having index $c(t)$ and *i*-th neuron.

The most important feature of the Kohonen learning algorithm is that the area of the neighborhood shrinks over time. In addition, the effect of learning is proportional to the distance a node is from the BMU. As a rule, the amount of learning is fading over distance and at the edges of the BMUs neighborhood, the learning process does not have barely any effect.

The SOM have shown to be successful for the analysis of high-dimensional data on data mining applications such as network security. However, the effectiveness of using traditional SOM models is limited 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 configuration. Another disadvantage of the fixed grid in SOM, is that traditional SOM can not represent hierarchical relation that might be present in the data.

### 3.2 Growing Hierarchical Self-organizing Maps

The limitations mentioned above can be resolved by applying growing hierarchical self-organizing maps. GHSOM has been developed as a multi-layered hierarchical architecture which adapts its structure according to the input data. It is initialized with one SOM and grows in size until it achieves an improvement in the quality of representing data. In addition, each node in this map can dynamically be expanded down the hierarchy by adding a new map at a lower layer providing a further detailed representation of data. The procedure of growth can be repeated in these new maps. Thus, the GHSOM architecture is adaptive and can represent data clearly by allocating extra space as well as uncover the hierarchical structure in the data.

The GHSOM architecture starts with the main node at zero layer and a $2 \times 2$ map at the first layer trained according to SOM training algorithm. The main node represents the complete data set $X$ and its weight vector $w_0$ is calculated as mean value of all data inputs. This node controls the growth of the SOM at the first layer and the hierarchical growth of whole GHSOM. The growth of the map at the first layer and maps at the next layers is controlled by using the quantization error. This error for the $i$-th node is calculated as follows

$$e_i = \sum_{x_j \in C_i} \left|\left|w_i - x_j\right|\right|, \qquad (2)$$

where $C_i$ 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. The quantization error $E_m$ of map $m$ is defined as

$$E_m = \frac{1}{|U_m|} \sum_{i \in U_m} e_i, \qquad (3)$$

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 $m$-th map.

When $E_m$ reaches certain fraction $\alpha_1$ of the $e_u$ of the corresponding parent unit $u$ in the upper layer, the growing process is stopped. The parent node of the SOM at the first layer is the main node. The parameter $\alpha_1$ controls the breadth of maps and its value ranges from 0 to 1. After that, the most dissimilar neighboring node $s$ is selected according to

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

where $w_j$ is the weight vector of the error node, $N_e$ is the set of neighboring nodes of $e$-th node, and $w_i$ is weight vector of 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.

After the growth process of an SOM is completed, every node of this SOM has to be checked for fulfillment of the global stopping criterium (Rauber et al., 2002):

$$e_i < \alpha_2 e_0, \qquad (5)$$

where $\alpha_2 \in (0,1)$ is parameter which controls the hierarchical growth of GHSOM, and $e_0$ is the quantization error of the main node, which can be found as follows:

$$e_0 = \sum_{x_j \in X} \left|\left|w_0 - x_j\right|\right|. \qquad (6)$$

Nodes not satisfying this criterium (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 $2 \times 2$ nodes is created. This maps weight vectors are initialized so that to mirror the orientation of neighboring the units of its parent. For this reason, we can choose to set new four nodes to the means of the parent and its neighbors in the respective directions (Chan and Pampalk, 2002). The newly added map is trained by using the input vectors which are mapped onto the node which has just been 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 a GHSOM is formed unsupervised based on the input data as well as hierarchal relationships in the data are discovered.

## 4 METHOD

The anomaly detection algorithm which is proposed in this study is based on the usage of GHSOM. The algorithm consists of three main stages: training, detecting and updating.

## 4.1 Training

In the training phase, server logs are used to obtain training set. The logs can contain several thousands of HTTP requests which are gathered from different web-resources during several days or weeks. In addition, these logs can 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 number of real intrusions and anomalous requests. HTTP requests from these logs are transformed to feature matrix by applying n-gram model.

When the feature matrix is obtained, new GHSOM is constructed and trained based on this matrix. The zero layer of this GHSOM is formed by several independent nodes the number of which corresponds to the number of different resources of the web-server. For each such node a SOM is created and initialized with four nodes. Requests to one web-resource are mapped to the corresponding parent node on the zero layer and used for training corresponding SOM. These SOMs form the first layer and each of these maps 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 as it is explained in Section 3. For each parent node on the zero layer the quantization error is calculated which controls the growing process of maps on the first layer and hierarchical growth of the GHSOM constructed.

## 4.2 Detection Method

The aim is not to find intrusions in the logs which were used as the training set, but to detect attacks among new requests received by the web-server. The new request is transformed to the frequency vector by applying n-gram model. After that, it goes to one of the parent node according to its resource and mapped to one of the nodes on the corresponding map by calculating the best matching unit for this request. To determine whether the new request is attack or not, two following criteria are used:

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

The threshold for the first criterium is calculated based on the distances between the weight vector of the node which is the BMU for the new request and other requests from the server logs already mapped to this node at the training stage. Assume that new request is mapped to the node which already contains $l$ other requests which are mapped to this node during the training phase. Denote the distances between the node and these $l$ requests as $e_1, e_2, \ldots, e_l$. Let us assume that values of these distances are distributed more or less uniformly. In this case, we can estimate maximum $\tau$ of continuous uniformly distributed variable as follows (Johnson, 1994):

$$\tau = \frac{l+1}{l} \max_{l}\{e_1, e_2, \ldots, e_l\}. \tag{7}$$

Obtained value $\tau$ can be used as the threshold value for the node considered, and new request is classified as an intrusion if distance between this request and the node is greater than $\tau$.

To find ”anomalous” nodes the $U^*$-matrix (Ultschk, 2005) is calculated for each SOM. $U^*$-matrix presents a combined visualization of the 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 a SOM (Ultsch and Siemon, 1990). The value of $i$-th element of U-matrix is the average distance of $i$-th node weight vector $w_i$ to the weight vectors of its immediate neighbors. Thus, $i$-th element of U-matrix $U(i)$ is calculated as follows:

$$U(i) = \frac{1}{n_i} \sum_{j \in N_i} D(w_i, w_j), \tag{8}$$

where $n_i = |N_i|$ is the number of nodes in the neighborhood $N_i$ of the $i$-th node, and $D$ is a distance function which for example can be Euclidean distance. A single element of U-matrix shows the local distance structure. If a global view of a U-matrix is considered then the overall structure of densities can be analyzed.

P-matrix allows a visualization of density structures of the high dimensional data space (Ultsch, 2003a). 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 node:

$$P(i) = |\{x \in X | D(x, w_i) < r\}|, \tag{9}$$

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 (Ultsch, 2003b):

$$r = \frac{1}{2}\chi_d^2(p_u), \tag{10}$$

where $\chi_d^2$ is the Chi-square cumulative distribution function for $d$ degrees of freedom and $p_u = 20.13\%$ of requests number containing in the data set $X$. The only condition is that all points in $X$ must follow a multivariate mutual 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 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:

$$U^*(i) = U(i)\frac{|p \in P|p > P(i)|}{|p \in P|}, \tag{11}$$

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 regions of clusters). We can also adjust the multiplication factor such that $U^*(i) = 0$ for the $p_{high}$ percent of P-matrix elements which have greatest values.

Since we assumed that most of requests are normal, intrusions can not form big clusters but will be mapped to nodes which are located on cluster borders. Thus, ”anomalous” nodes are those ones which correspond to high values of $U^*$-matrix elements. In this research, the following criterium for finding anomalous nodes is used: if 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 node of GHSOM is classified as ”normal” but has child SOM, then all nodes of this child SOM should be also checked whether they are ”normal” or ”anomalous” by calculating new $U^*$-matrix for this SOM.

### 4.3 Updating

Web-applications are highly dynamic and change on regular basis, which can cause noticeable changes in the HTTP requests which are sent to the web-server. It can lead to the situation when all new allowable requests will be classified as intrusions. For this reason, the GHSOM should be retrained after a certain period of time $T$ to be capable of classifing new requests.

Let us assume that the number of requests sent to the web-server for this period $T$ is much less than number of requests in the training set. We update the training set by replacing first requests from this set by requests obtained during the period $T$. After that, the GHSOM is retrained by using the resulting training set. During the update phase the structure of the GHSOM can be modified. The update of the GHSOM structure starts from the current structure. Parameters $\tau$ and matrices $U$, $P$ and $U^*$ should be recalculated. The update phase can occur independently from the detecting anomalies. During retraining, requests obtained are classified using old GHSOM, and when 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 webserver with a large number of intrusions. It 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. It is also possible to restrict the globally allowed replacements per time slot independent of the IP addresses, in order to address the threat of botnets.

## 5 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 anomalies and actual intrusions. The logfiles are acquired from several Apache servers and stored in combined log format. The logs contain requests from multiple webresources. Since it is not possible to inject code via static requests unless there are major deficiencies in the HTTP server, we focus on finding anomalies from dynamic requests because these requests are used by the web-applications, which are run behind the HTTP server. Thus, the requests without parameters are ignored.

We run two simulations. In the first simulation, requests to the most popular web-resourse are chosen from the logs. In the second simulation, thirty-eight most popular resources are analyzed. In both cases, the training set is created at the beginning. It contains 10 000 and 25 000 requests in the first and second tests respectively. After GHSOMs are trained, new requests are chosen from logfiles and classified by GHSOMs one by one to test the technique proposed. The number of testing requests is equal 20 000 and 50 000 for the first and second simulations respectively. During the testing, the GHSOMs are updated every 2 000 and 5 000 requests in the first and second cases respectively.

To evaluate the performance of proposed technique, the following characteristics are calculated in both tests:

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

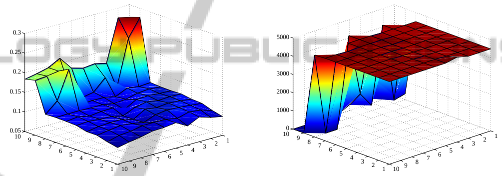
In the first test, the requests in training and testing sets are related to one web-resource. This resource allows users to search a project by choosing the appropriate category of the projects or initial symbols of the project name. Thus, all requests have one of the two different attributes which can be used by attackers to inject malignant code. Settings of the first simulation are presented in Table 1. When the GHSOM training

Table 1: The first simulation settings.

| Number of web-resources | Training set size | Testing set size | Update period |
|---|---|---|---|
| 1 | 10 000 | 20 000 | 2 000 |

is completed, $U$-matrix, $P$-matrix and $U^*$-matrix are constructed. In Figure 1, $U$-matrix and $P$-matrix are shown. As one can see, some nodes on one of the map edges are distant from all others (Figure 1 (a)), and at the same time density of data inputs in these nodes is very low (Figure 1 (b)). These facts make these nodes candidates to ”anomalous” ones. $U^*$-matrix is plotted in Figure 2. We can notice that there are two big clusters corresponding to the requests in which different methods of searching required project are used: by specifying the project category or initial symbols of project name. Nodes on one of the map edges are classified as ”anomalous”. The technique proposed can 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, buffer overflow attacks and directory traversal attacks, as shown in Figure 2.

After constructing $U^*$-matrix, detection process is started. New requests are mapped to the GHSOM one by one and classified as intrusions if the distance between new request and its BMU weight vector is greater than threshold value or if the node which is the BMU for this new request is anomalous. During the detection phase, the GHSOM is retrained periodically when a certain number of requests are processed. After the GHSOM update, threshold values for all nodes are modified and $U^*$-matrix is also recalculated. Figure 3 shows the $U^*$ matrix after the training phase and the first and fifth updates. Requests which are used



(a) $U$-matrix. (b) $P$-matrix.

Figure 1: $U$-matrix and $P$-matrix after the training stage in the first simulation.

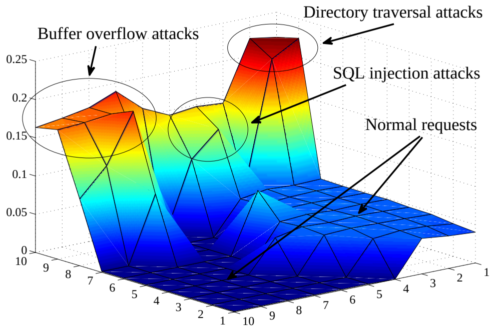



Figure 2: $U^*$-matrix for detecting anomalies after the training stage in the first simulation.

Table 2: The second simulation settings.

| Number of web-resources | Training set size | Testing set size | Update period |
|---|---|---|---|
| 38 | 20 000 | 50 000 | 5 000 |

as the testing set in our first simulation, contain also other types of intrusions except those ones which are

Table 3: The first simulation results.

| True positive rate | Flase positive rate | True negative rate | False negative rate | Accuracy | Precision |
|---|---|---|---|---|---|
| 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |

Table 4: The second simulation results for different web-resources.

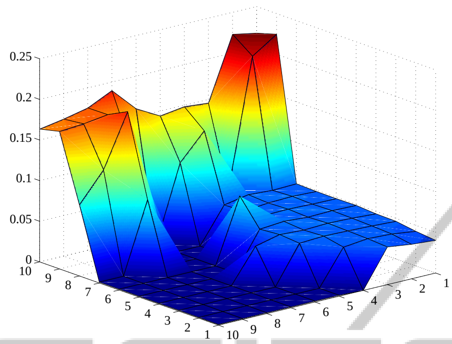
| Resource number | True positive rate | Flase positive rate | True negative rate | False negative rate | Accuracy | Precision |
|---|---|---|---|---|---|---|
| 1 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 2 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 3 | 98.08 % | 0 % | 100 % | 1.92 % | 99.90 % | 100 % |
| 4 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 5 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 6 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 7 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 8 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 9 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 10 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 11 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 12 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 13 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 14 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 15 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 16 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 17 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 18 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 19 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 20 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 21 | 95.65 % | 0 % | 100 % | 4.35 % | 99.74 % | 100 % |
| 22 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 23 | 98.57 % | 0 % | 100 % | 1.43 % | 99.93 % | 100 % |
| 24 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 25 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 26 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 27 | 100 % | 0.10 % | 99.90 % | 0 % | 99.90 % | 98.00 % |
| 28 | 100 % | 0.07 % | 99.93 % | 0 % | 99.93 % | 98.63 % |
| 29 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 30 | 100 % | 0.20 % | 99.80 % | 0 % | 99.81 % | 95.35 % |
| 31 | 97.50 % | 0 % | 100 % | 2.50 % | 99.87 % | 100 % |
| 32 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 33 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 34 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 35 | 98.51 % | 0 % | 100 % | 1.49 % | 99.93 % | 100 % |
| 36 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 37 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| 38 | 100 % | 0 % | 100 % | 0 % | 100 % | 100 % |
| Average | 99.69 % | 0.01 % | 99.99 % | 0.31 % | 99.97 % | 99.79 % |

contained in the training set. However, all intrusions are detected and false alarms are absent. The summary of the first simulation results is presented in Table 3.

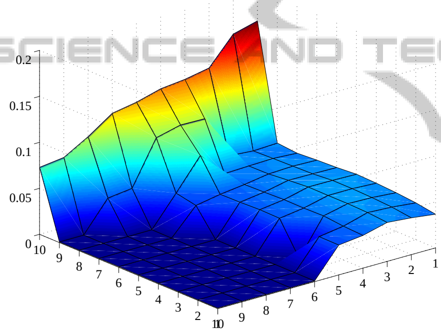
In the second simulation, thirty eight most popular web-resources are chosen from web-server logs. Since the logs contain a few different types of intrusions, we generate other types of intrusions and add them to the testing set. The basic settings of the second simulation are presented in Table 2.

Results of the detection phase are shown in Table 4. As one can see, almost all real attacks are classified correctly as intrusions by using proposed technique. At the same time, 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.

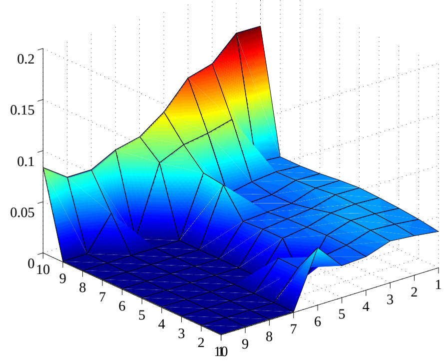
In the second simulation, the training set contains fourteen different types of attack. The results for these attack types are presented in Table 5. We can see that our algorithm found 99.63% of all attacks. Thus, almost all intrusions are detected despite the fact that some of them are not contained in the training set.




(a) After training phase.




(b) After the first update.




(c) After the fifth update.

Figure 3: $U^*$-matrix in the first simulation.

Table 5: The second simulation results for different types of attacks.

| Attack type | Total number of attacks | Number of detected attacks | Proportion of detected attacks |
|---|---|---|---|
| SQL injection | 229 | 229 | 100 % |
| Directory traversal | 276 | 276 | 100 % |
| Buffer overflow | 243 | 243 | 100 % |
| Cross-site scripting | 230 | 230 | 100 % |
| Double encoding | 239 | 239 | 100 % |
| Common gateway interface scripting | 189 | 182 | 96.30 % |
| Shell scripting | 44 | 43 | 97.73 % |
| XPath injection | 251 | 251 | 100 % |
| HTTP response splitting | 253 | 253 | 100 % |
| Cache poisoning | 44 | 44 | 100 % |
| Eval injection | 186 | 186 | 100 % |
| Web-parameter tampering | 81 | 80 | 98.77 % |
| String formatting | 79 | 79 | 100 % |
| Cross-User defacement | 67 | 67 | 100 % |
| Total | 2411 | 2402 | 99.63 % |

## 6 CONCLUSIONS AND DISCUSSION

The main advantage of anomaly detection based IDSs is that they are able to detect zero-day attacks. In this research, adaptive growing hierarchical self-organizing maps are used to find anomalies in HTTP requests which are sent to the server. The technique proposed is self-adaptive and allows to detect HTTP attacks in online mode in the case of continuously updated web-applications. The method is tested using logs acquired from a large real-life web-service. These logs include normal and intrusive requests. As a result, almost all attacks from these logs are detected and at the same time the number of false alarms is very low. Thus, the accuracy of the method proposed is about one hundred percent. However, this method can be applied only if the number of HTTP requests to a web-resource is enough to analyze the normal behaviour of users. Sometimes, attackers try to access the data stored on servers or to harm the system by using holes in the security of not popular web-resources for which it is difficult to define which requests are "normal". In the future, we are planning to develop anomaly detection based system which can solve this problem.

## REFERENCES


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

Axelsson, S. (1998). Automatically hardening web applications using precise tainting. Technical report, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden.

Chan, A. and Pampalk, E. (2002). Growing hierarchical self organising map (ghsom) toolbox: visualisations and enhancements. In *9-th International Conference Neural Information Processing, ICONIP '02*, volume 5, pages 2537–2541.

Gollmann, D. (2006). *Computer Security*. Wiley, 2nd edition.

Hirsimaki, T., Pylkkonen, J., and Kurimo, M. (2009). Importance of high-order n-gram models in morph-based speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions*, 17:724–732.

Ippoliti, D. and Xiaobo, Z. (2010). An adaptive growing hierarchical self organizing map for network intrusion detection. In *19th IEEE International Conference Computer Communications and Networks (ICCCN)*, pages 1–7.

Jiang, D., Yang, Y., and Xia, M. (2009). Research on intrusion detection based on an improved som neural network. In *Fifth International Conference on Information Assurance and Security*, pages 400–403.

Johnson, R. W. (1994). Estimating the size of a population. *Teaching Statistics*, 16:50–52.

Kayacik, H. G., Nur, Z.-H., and Heywood, M. I. (2007). A hierarchical som-based intrusion detection system. *Engineering Applications of Artificial Intelligence*, 20:439–451.

Kemmerer, R. and Vigna, G. (2002). Intrusion detection: A brief history and overview. *Computer*, 35:27–30.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43:59–69.

Kohonen, T. (2001). *Self-organizing map*. Springer-Verlag, Berlin, 2rd edition.

Mukkamala, S. and Sung, A. (2003). A comparative study of techniques for intrusion detection. In *Tools with Artificial Intelligence, 15th IEEE International Conference*.

Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., and Evans, D. (2005). Automatically hardening web applications using precise tainting. In *20th IFIP International Information Security Conference*.

Palomo, E. J., Domínguez, E., Luque, R. M., and Muñoz, J. (2008). *A New GHSOM Model Applied to Network Security*, volume 5163 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany.

Patcha, A. and Park, J. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51.

Rauber, A., Merkl, D., and Dittenbach, M. (2002). The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data. *Neural Networks, IEEE Transactions*, 13:1331–1341.

Shehab, M., Mansour, N., and Faour, A. (2008). Growing hierarchical self-organizing map for filtering intrusion detection alarms. In *International Symposium Parallel Architectures, Algorithms, and Networks, I-SPAN*, pages 167–172.

Suen, C. Y. (1979). n-gram statistics for natural language understanding and text processing. *Pattern Analysis and Machine Intelligence, IEEE Transactions*, PAMI-1:162–172.

Ultsch, A. (2003a). Maps for the visualization of high-dimensional data spaces. In *Workshop on Self-Organizing Maps (WSOM 2003),*, pages 225–230.

Ultsch, A. (2003b). Pareto density estimation: A density estimation for knowledge discovery. In *Innovations in Classification, Data Science, and Information Systems - Proc. 27th Annual Conference of the German Classification Siciety*, pages 91–100.

Ultsch, A. and Siemon, H. P. (1990). Kohonen's self organizing feature maps for exploratory data analysis. In *Proc. Intern. Neural Networks, Kluwer Academic Press*, pages 305–308.

Ultschk, A. (2005). Clustering with som: U*c. In *U*C. Proc. Workshop on Self-Organizing Maps (WSOM 2005)*, pages 75–82.

Verwoerd, T. and Hunt, R. (2002). Intrusion detection techniques and approaches. *Computer Communications*, 25:1356–1365.