Growing Hierarchical Self-organizing Maps and Statistical Distribution Models for Online Detection of Web Attacks

Mikhail Zolotukhin, Timo Hämäläinen, and Antti Juvonen

Department of Mathematical Information Technology,
University of Jyväskylä, Jyväskylä, FI-40014, Finland
{mikhail.m.zolotukhin,timo.t.hamalainen,antti.k.a.juvonen}@jyu.fi
https://www.jyu.fi/it/laitokset/mit/en/

Abstract. In modern networks, HTTP clients communicate with web servers using request messages. By manipulating these messages attackers can collect confidential information from servers or even corrupt them. In this study, the approach based on anomaly detection is considered to find such attacks. For HTTP queries, feature matrices are obtained by applying an n-gram model, and, by learning on the basis of these matrices, growing hierarchical self-organizing maps are constructed. For HTTP headers, we employ statistical distribution models based on the lengths of header values and relative frequency of symbols. New requests received by the web-server are classified by using the maps and models obtained in the training stage. The technique proposed allows detecting online HTTP attacks in the case of continuous updated web-applications. The algorithm proposed is tested using logs, which were acquired from a large real-life web service and included normal and intrusive requests. As a result, almost all attacks from these logs are detected, and the number of false alarms remains very low.

Keywords: Intrusion detection, Anomaly detection, n-Gram, Growing hierarchical self-organizing map, Single-linkage clustering.

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 various forms of intrusions. Such intrusions are executed manually by a person or automatically with engineered software and can use legitimate system features as well as programming mistakes or system misconfigurations [1]. That is why computer security becomes one of the most important issues when designing computer networks and systems.

Some 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 give rise to 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 a set

J. Cordeiro and K.-H. Krempels (Eds.): WEBIST 2012, LNBIP 140, pp. 281–295, 2013.

of parameters having some values. It is possible to manipulate these queries to create requests which can corrupt the server or collect confidential information [2]. In addition, a HTTP request message contains header fields, which define the operating parameters of an HTTP transaction. Such fields usually contain information about user agent, preferred response languages, connection type, referer, etc. The attacker can inject malicious code to these fields to construct various kinds of attacks based on HTTP response splitting or malicious redirecting [25].

One way to ensure the security of web-servers and web-based applications is to use 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 [3]. There are many different IDS architectures, which continue to evolve with time [4,5]. 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 [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 which form normal behaviour, and, by observing patterns that deviate from the established norms (anomalies), detects when an intrusion has occurred. Thus, systems which use the anomaly detection approach are modelled 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). As a rule, anomaly detection IDSs for web-servers are based on supervised learning, which trains the system by using a set of normal queries. On the other hand, unsupervised anomaly detection techniques do not need normal training data, and therefore such techniques are the most usable.

To find code injections in HTTP headers, we apply statistical distribution models based on the length of header values and relative frequency of non-alphanumeric symbols, whereas, to detect intrusive HTTP queries, the approach based on Growing Hierarchical Self-Organizing Maps (GHSOMs) is employed. For analyzing and visualizing high dimensional data, a regular Self-Organizing Map (SOM) based on the unsupervised learning neural network model proposed by Kohonen can be used [9]. 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 [8]. Despite the fact that the approach based on self-organizing maps has shown effectiveness at detecting intrusions [10,11], 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 [12]. This neural network consists of several SOMs structured in layers, the number of neurons, maps and layers

being determined during the unsupervised learning process. Thus, the structure of the GHSOM is automatically adapted according to the structure of the data.

The GHSOM approach looks promising for solving the problem of detecting network intrusions. In [13], a GHSOM model with a metric which combines both numerical and symbolic data is proposed for detecting network intrusions. An IDS based on this model detects anomalies by classifying IP connections into normal or anomalous connection records, and, if they are anomalies, into the type of attack. An 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. The study in [15] 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 request messages by applying an approach that is based on adaptive growing hierarchical self-organizing maps and statistical distribution models. The remainder of this paper is organized as follows: Section 2 describes the process of data acquisition and feature extraction from network logs; in Section 3 we show how to apply adaptive GHSOM and statistical distribution models for detecting anomalies; experimental results are presented in Section 4; and Section 5 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 time zone, the HTTP request, which includes the resource and parameters used, the server's response code, the amount of data sent to the user, and the web-page which was requested and used by a browser software. Here is an example of a single line from an Apache server log file. This information is stored in a combined log format [24]:

```
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)"
```

Here the focus is on analysis of HTTP header fields and HTTP queries, which are strings containing a set of attributes having some values. We do not focus on static HTTP queries because they do not contain any parameters. It is not possible to inject code via static requests unless there are major deficiencies in the HTTP server itself. Dynamic queries, which are handled by the web applications of the service, are more interesting for this study, because all static queries are normal. Let us assume that most request messages which are coming to the HTTP server are normal, i.e. they use legitimate features of the service, but some obtained requests are intrusions. All HTTP requests are analyzed to detect the anomalous ones.

A HTTP query can be expressed as a composition of the path to the desired web resource and a string which is used to pass parameters to the referenced resource and identified by a leading '?' character. To extract features from each query, an n-gram model is applied. N-gram models are widely used in statistical natural language processing [16] and speech recognition [17]. An n-gram is a sub-sequence of n overlapping items (characters, letters, words, etc) from a given sequence. For example, a 2-gram character model for the string '/resource?parameter1=value1&parameter2=value2' is '/r', 're', 'es', 'so', 'ou', 'ur', ..., 'lu', 'ue', 'e2'.

An n-gram character model is applied to transform each HTTP query to a sequence of n characters. Such sequences are used to construct an n-gram frequency vector, which expresses the frequency of every n-character in the analyzed request. To obtain this vector, ASCII codes of characters are used to represent the sequence of n-characters as a sequence of arrays, each of which contains n decimal ASCII codes, and the frequency vector is built by counting the number of occurrences 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 occurrences of each such pair. For example, the entry in location $(256 \times 61 + 118)$ in this vector contains a value equal to 2 since the pair $[61, 118]$, which corresponds to pair '=v' can be seen twice. Thus, each HTTP query is transformed into a 256^n numeric vector. The matrix consisting of these vectors is called the feature matrix and it can be analyzed to find anomalies.

To extract features from HTTP headers, the lengths of header values and all non alphanumeric symbols used are counted and stored separately for different HTTP header types. These vectors of length and sets of non-alphanumeric symbols can be used to train the system and find code injections in the header fields of HTTP request messages.

3 Method

The algorithm proposed can be considered as a set of two classifiers. The first of these is based on transforming the query strings into numeric vectors by applying an n-gram model, and constructing and training GHSOMs using the feature matrices obtained. The second one analyzes HTTP headers and searches for code injections using statistical distribution models based on the length of header values and relative frequency of non-alphanumeric symbols. If a request is defined as anomalous at least by one of these classifiers then this request is classified as an intrusion.

3.1 Detecting Anomalous HTTP Query Strings

In this study, adaptive growing hierarchical self-organizing maps are used to find anomalous HTTP queries. A 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. 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, SOMs are widely applied for visualizing low-dimensional views of high-dimensional data.

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 neighbourhood relation dictating the structure of the map. Associated with the i-th neuron of the SOM is a 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 the other one 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. 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, the dimensions of the map grid, the map lattice and the shape should be determined. Before the training, initial values are given to the prototype vectors. A 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 as 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 neighbours are moved closer to the input vector in the input space:

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

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

The most important feature of the Kohonen learning algorithm is that the area of the neighbourhood shrinks over time. In addition, the effect of learning is proportional to the distance of the node from the BMU. As a rule, the amount of learning fades over distance, and, at the edges of the BMUs neighbourhood, the learning process has barely any effect.

The SOM has shown to be successful for the analysis of high-dimensional data in data mining applications such as those used for 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 relations that might be present in the data.

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 to the input data. It is initialized with one SOM and

grows in size until it achieves an improvement in the quality its representation of data. In addition, each node in this map can be dynamically expanded down the hierarchy by adding a new map at a lower layer for 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 the zero layer and a 2×2 map at the first layer trained according to the SOM training algorithm. The main node represents a complete data set X, and its weight vector w_0 is calculated as the mean value of all data inputs. This node controls the growth of the SOM at the first layer and the hierarchical growth of the whole GHSOM. The growth of the map at the first layer and the maps at the next layers are controlled with the help of quantization error. This error for the i-th node is calculated as follows

$$e_i = \sum_{x_j \in C_i} ||w_i - x_j|| \,, \tag{2}$$

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

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

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

When E_m reaches certain fraction α_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 α_1 controls the breadth of maps, and its value ranges from 0 to 1. After that, the most dissimilar neighbouring 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 error node, N_e is the set of neighbouring nodes of the e-th node, and w_i is the weight vector of the neighbouring node in set N_e. A new row or column of nodes is placed in between the nodes e and s. The weight vectors of the newly added nodes are initialized with the mean of their corresponding neighbours.

After the growth process of the SOM is completed, every node of this SOM has to be checked for satisfying of the global stopping criterion [12]:

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

where $\alpha_2 \in (0, 1)$ is the 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:

$$e_0 = \sum_{x_j \in X} ||w_0 - x_j|| \,. \tag{6}$$

The nodes not satisfying this criterion (5), and therefore representing a set of too diverse input vectors, are expanded to form a new map at the subsequent layer of the hierarchy.

Similarly 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 neighbouring units of its parent. For this reason, we can choose to set four new nodes to the means of the parent and its neighbours 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 a GHSOM is formed unsupervised and based on the input data, and hierarchal relationships in the data are discovered.

The anomaly detection algorithm which is proposed in this study is based on the use of GHSOM. The algorithm consists of three main stages: training, detecting and updating. In the training phase, server logs are used to obtain a training set. The logs can contain several thousands of HTTP requests, which are gathered from various web-resources during several days or weeks. 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 the number of real intrusions and anomalous requests. HTTP queries from these logs are transformed to a feature matrix by applying an n-gram model.

When the feature matrix is obtained, a 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 the 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, thus providing a further detailed representation of data. For each parent node on the zero layer, the quantization error which controls the growing process of the maps on the first layer is calculated and the GHSOM is hierarchically grown.

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. Each new query is transformed to a frequency vector by applying the n-gram model. After that, this vector goes to one of the parent node according to its resource and is mapped to one of the nodes on the corresponding map by calculating the best matching unit for this query. To determine whether the new request is an attack or not, the following two criteria are used:

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

The threshold for the first criterion is calculated based on the distances between the weight vector of the node, which is the BMU for the new query, and other queries from the server logs already mapped to this node at the training stage. Assume that the new query is mapped to the node which already contains l other queries mapped to this node

during the training phase. Denote the distances between the node and these l queries as $e_1, e_2, \ldots, e_l$. Let us assume that the values of these distances are distributed more or less uniformly. In this case, we can estimate maximum τ of continuous uniformly distributed variable as follows [19]:

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

Obtained value τ can be used as the threshold value for the node considered, and a new request message is classified as an intrusion if the distance between its query and the node is greater than τ.

To find "anomalous" nodes, a U^*-matrix [20] 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 queries mapped to a SOM [21]. The value of the i-th element of an U-matrix is the average distance of the i-th node weight vector w_i to the weight vectors of its immediate neighbours. Thus, the i-th element of the 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 neighbourhood 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 [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 node:

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

where X is the set of queries 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]:

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

where χ_d^2 is the Chi-square cumulative distribution function for d degrees of freedom and $p_u = 20.13\%$ of the number of requests contained 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 the principal component analysis, standardization and other transformations.

The U^*-matrix which is the combination of a U-matrix and a P-matrix combines distance relationships with density relationships and can give an appropriate clustering. The i-th element of the U^*-matrix is equal to $U(i)$ 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 the P-matrix elements which have greatest values.

Since we assumed that most of the 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 which correspond to high values of U^*-matrix elements. In this research, the following criterion for finding anomalous nodes is used: if the difference between $U^*(i)$ 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 a GHSOM is classified as "normal" but has a child SOM, then all the nodes of this child SOM should also be also checked by calculating new U^*-matrix for this SOM to find out whether they are "normal" or "anomalous".

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, the GHSOM should be retrained after a certain period of time T to be capable of classifying 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 the 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 τ and matrices U, P and U^* should be recalculated. The update phase can occur independently from the anomaly detection. During retraining, requests obtained are classified using the old GHSOM, and, 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. 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. 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.

3.2 Detecting Anomalous HTTP Headers

Usually header fields have a finite set of possible values, therefore to solve the problem of finding anomalous headers it is reasonable to apply simple statistical distribution models. In this research, we analyze the lengths of header fields and non-alphanumeric symbols used in them [26]. All different header types are supposed to be analyzed separately. Similarly to the previous scheme, for the second classifier we define three stages: training, detecting and updating.

In the training stage, headers of the request messages which have been employed for constructing GHSOMs are used. For each header type, we construct the vector of its lengths $(l_1, l_2, \ldots, l_M)$. Since we assume that some HTTP requests from the training set can be attacks, some filtering can be applied to remove outliers and build the pattern of normal user behaviour. As proposed in study [26], we define the following distance function: $d(l_i, l_j) = p(l_i \text{ is normal}) - p(l_j \text{ is normal})$, where $p(x \text{ is normal})$ is the probability that length x is normal and can be found as follows:

$$p(x \text{ is normal}) = \begin{cases} \frac{\sigma^2}{(x-\mu)^2}, \text{ if } x \geq \mu + \sigma, \\ 1, \text{ if } x < \mu + \sigma, \end{cases} \tag{12}$$

where μ and σ^2 are the mean and the variance, respectively. In this case, the distance d between a normal pattern and an outlier pattern is expected to be higher than the distance d between two normal patterns or two outlier patterns. Thus, it is easy to divide all the lengths into two clusters, i.e. normal lengths and outliers, by using a simple clustering algorithm, e.g. a single-linkage clustering [27]. All outliers are removed from the model and all normal lengths are used for detecting anomalies. For header type k we denote the cluster of normal lengths as L_k^n and the number of entries contained in this cluster as $|L_k^n|$.

In addition, during training all non-alphanumeric symbols are counted for each header type. The distance function between the different symbols s_i and s_j is defined as $d(s_i, s_j) = p(s_i \text{ is normal}) - p(s_j \text{ is normal})$ where $p(x \text{ is normal})$ is the probability that symbol x is legitimate and can be found as the relative frequency of symbol x in the training set:

$$p(x \text{ is normal}) = \frac{N_x}{N}, \tag{13}$$

where N_x is the number of appearances of non-alphanumeric symbol x in the training set and N is the total number of non-alphanumeric symbols there. Similarly to the previous model, all outliers can be removed by applying a single-linkage clustering, where the distance between two symbols is defined as $d(s_i, s_j)$ and the number of clusters is two. All the remaining non-alphanumeric symbols are considered as legitimate to use. Let us denote the set of legitimate symbols for header type k as S_k^l.

In the detecting stage, the following criterion is used to find anomalous requests. Let a new request message received by the web server contain the following header fields $\{h_1, h_2, \ldots\}$. If for any header value h_k of this request at least one of the following conditions:

1. length of $h_k > \frac{|L_k^n|+1}{|L_k^n|} \cdot \max(L_k^n)$, where $\max(L_k^n)$ is maximal element of L_k^n,

2. $\exists$ non-alphanumeric symbol $s \in h_k : s \notin S_k^l$.

is satisfied, then this new request message is classified as an intrusion, otherwise it is legitimate.

Similarly to the first part of the algorithm, all these criteria are supposed to be updated with time. We update the training set by replacing the first requests from this set by requests obtained during the period T. For the updated training set for each header type we find a new cluster of normal lengths, recalculate the threshold value, and update the set of legitimate non-alphanumeric symbols. Just as in the case for detecting anomalous HTTP queries, countermeasures against attackers trying to affect the training set by flooding the web server with a large number of intrusions are supposed to be applied here also.

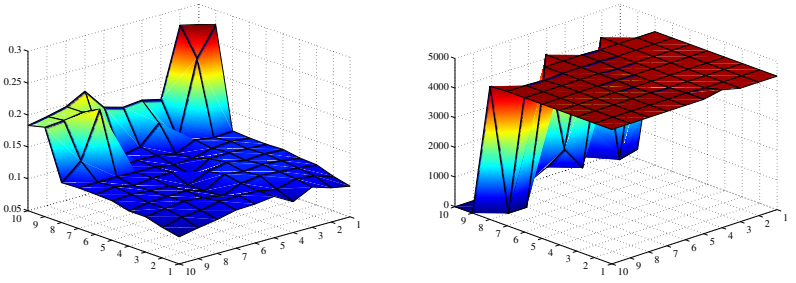
4 Simulation Results

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 a combined log format. The logs contain requests from multiple web-resources. Since it is not possible to inject code via static query strings unless there are major deficiencies in the HTTP server, HTTP query strings without parameters are considered as normal.

In our simulation, request messages to twenty-five most popular web resources of the server are analyzed. The training set is created at the beginning and it contains 20000 requests. By using this training set, twenty-five GHSOMs are trained (one for each web resource) based on dynamic query strings and for each header type the cluster of legitimate lengths as well as the set of legitimate non-alphanumeric symbols are formed. New requests are chosen from logfiles and classified one by one to test the technique proposed. The number of requests in the testing set is equal to 100000 and 9679 of them are attacks. During the testing stage, the system is updated after each processing of 5000 requests.

To evaluate the performance of the proposed technique, the following characteristics are calculated in our test:

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



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

Fig. 1. U-matrix and P-matrix after the training stage

Let us consider one of the web resources, which allows users to search a project by choosing the appropriate category of the projects or initial symbols of the project name. Thus, query strings of all those HTTP requests have one of the two different attributes which can be used by attackers to inject malignant code. When the GHSOM training 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 the density of data inputs in these nodes is very low (Figure 1 (b)). These facts make these nodes candidates to "anomalous" ones.

The U^*-matrix for this GHSOM is plotted in Figure 2. We can notice that there are two big clusters corresponding to the queries in which different methods of searching a required project are used: by specifying the project category or the initial symbols of project name. The nodes on one of the map edges are classified as "anomalous". The technique proposed does not allow us to define the 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 the U^*-matrix and building statistical distribution models for each header type, the detection process is started. Query strings of new requests are mapped to the GHSOM 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. In addition, the lengths of header fields and non-alphanumeric symbols used in them are checked according to the scheme proposed.

During the detection phase, the system is retrained periodically when a certain number of requests are processed. After the system update, all threshold values, GHSOMs and statistical distribution models are modified to allow detection of new request messages.

the results of the detection phase are shown in Table 1. As one can see, almost all real attacks are correctly classified as intrusions by using the proposed technique. At the same time, the false positive rate is about zero on average, which means that the number of false alarms is very low. The accuracy of the method is close to one hundred percent.

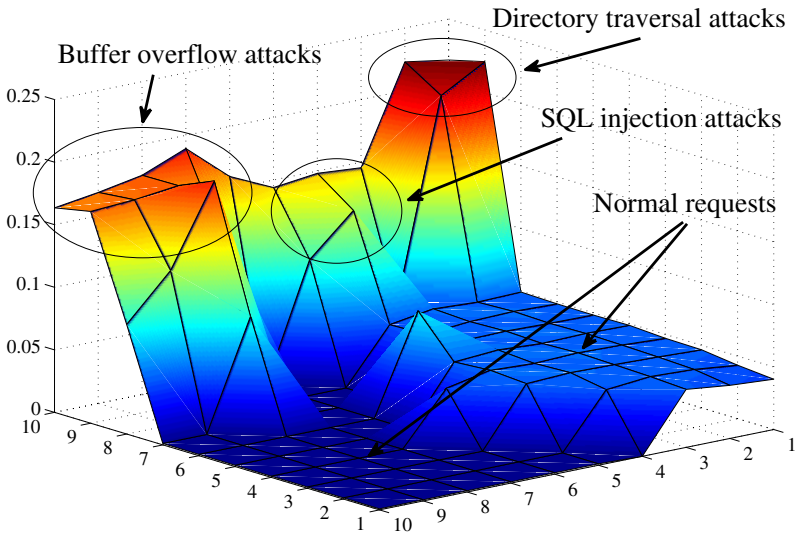


Fig. 2. U^*-matrix after the training stage

Table 1. Performance metrics values

True positive rate	False positive rate	True negative rate	False negative rate	Accuracy	Precision
99.56 %	0.00 %	100.00 %	0.44 %	99.96 %	100.00 %

Table 2. The simulation results for different types of attacks

Attack type	Total number of attacks	Number of detected attacks	Proportion of detected attacks
SQL injection	484	484	100 %
Directory traversal	488	469	96.11 %
Buffer overflow	488	486	99.59 %
Cross-site scripting	1011	1011	100 %
Double encoding	471	471	100 %
Common gateway interface scripting	401	392	97.76 %
Shell scripting	114	112	98.25 %
XPath injection	510	510	100 %
HTTP response splitting	2530	2529	99.96 %
Cache poisoning	117	117	100 %
Eval injection	377	369	97.88 %
String formatting	238	238	100 %
Cross-User defacement	246	246	100 %
Session fixation	2204	2204	100 %
Total	9679	9638	99.57 %

In our simulation, the testing set contains fourteen different types of attack. The results for these attack types are presented in Table 2. We can see that the proposed algorithm found 99.57% of all attacks. Thus, almost all intrusions are detected despite the fact that some of them are not contained in the training set.

5 Conclusions and Discussion

The main advantage of IDSs based on anomaly detection is that they are able to detect zero-day attacks. In this research, the approach based on anomaly detection is considered as suitable for finding intrusive HTTP request messages. The technique proposed is self-adaptive and allows detection of HTTP attacks in online mode in the case of continuously updated web-applications. The method was 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 large enough to allow the analysis of normal user behaviour. Sometimes, attackers try to access the data stored on servers or to harm the system by using holes in the security of less popular web-resources, for which it is difficult to define which requests are "normal". In the future, we are planning to develop an anomaly detection based system which can solve this problem.

References

1. Mukkamala, S., Sung, A.: A comparative study of techniques for intrusion detection. In: Proc. 15th IEEE International Conference on Tools with Artificial Intelligence, pp. 570–577 (November 2003)
2. Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., Evans, D.: Automatically Hardening Web Applications Using Precise Tainting. In: Sasaki, R., Qing, S., Okamoto, E., Yoshiura, H. (eds.) Security and Privacy in the Age of Ubiquitous Computing. IFIP AICT, vol. 181, pp. 295–307. Springer, Boston (2005)
3. Axelsson, S.: Research in intrusion-detection systems: a survey. Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, Technical Report, pp. 98–117 (December 1998)
4. Patcha, A., Park, J.M.: An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer Networks: The International Journal of Computer and Telecommunications Networking 51(12) (August 2007)
5. Verwoerd, T., Hunt, R.: Intrusion detection techniques and approaches. Computer Communications - COMCOM 25(15), 1356–1365 (2002)
6. Kemmerer, R.A., Vigna, G.: Intrusion Detection: A Brief History and Overview. Computer 35, 27–30 (2002)
7. Gollmann, D.: Computer Security, 2nd edn. Wiley (2006)
8. Kohonen, T.: Self-organizing map, 3rd edn. Springer, Berlin (2001)
9. Kohonen, T.: Self-organized formation of topologically correct feature maps. Biological Cybernetics 43(1), 59–69 (1982)
10. Kayacik, H.G., Nur, Z.-H., Heywood, M.I.: A hierarchical SOM-based intrusion detection system. Engineering Applications of Artificial Intelligence 20 (2007)

11. Jiang, D., Yang, Y., Xia, M.: Research on Intrusion Detection Based on an Improved SOM Neural Network. In: Proc. of the Fifth Intl Conference on Information Assurance and Security (2009)
12. Rauber, A., Merkl, D., Dittenbach, M.: The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data. IEEE Transactions on Neural Networks 13(6), 1331–1341 (2002)
13. Palomo, E.J., Domínguez, E., Luque, R.M., Muñoz, J.: A New GHSOM Model Applied to Network Security. In: Kůrková, V., Neruda, R., Koutník, J. (eds.) ICANN 2008, Part I. LNCS, vol. 5163, pp. 680–689. Springer, Heidelberg (2008)
14. Ippoliti, D., Xiaobo, Z.: An Adaptive Growing Hierarchical Self Organizing Map for Network Intrusion Detection. In: Proc. 19th IEEE International Conference on Computer Communications and Networks (ICCCN), pp. 1–7 (August 2010)
15. Shehab, M., Mansour, N., Faour, A.: Growing Hierarchical Self-Organizing Map for Filtering Intrusion Detection Alarms. In: International Symposium on Parallel Architectures, Algorithms, and Networks, I-SPAN 2008, pp. 167–172 (May 2008)
16. Suen, C.Y.: n-Gram Statistics for Natural Language Understanding and Text Processing. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-1(2), 164–172 (1979)
17. Hirsimaki, T., Pylkkonen, J., Kurimo, M.: Importance of High-Order N-Gram Models in Morph-Based Speech Recognition. IEEE Transactions on Audio, Speech, and Language Processing 17(4), 724–732 (2009)
18. Chan, A., Pampalk, E.: Growing hierarchical self organising map (ghsom) toolbox: visualisations and enhancements. In: 9th Int'l Conference Neural Information Processing, ICONIP 2002, vol. 5, pp. 2537–2541 (2002)
19. Johnson, R.W.: Estimating the Size of a Population. Teaching Statistics 16(2), 50–52 (1994)
20. Ultsch, A.: Clustering with SOM: U*C. In: Proc. Workshop on Self-Organizing Maps (WSOM 2005), Paris, France, pp. 75–82 (2005)
21. Ultsch, A., Siemon, H.P.: Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis. In: Proc. Intern. Neural Networks, pp. 305–308. Kluwer Academic Press, Paris (1990)
22. Ultsch, A.: Maps for the Visualization of high-dimensional Data Spaces. In: Proc. WSOM, Kyushu, Japan, pp. 225–230 (2003)
23. Ultsch, A.: 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 Society (GfKL), pp. 91–100. Springer, Heidelberg (2003)
24. Apache 2.0 Documentation (2011), `http://www.apache.org/`
25. Klein, A.: Detecting and Preventing HTTP Response Splitting and HTTP Request Smuggling Attacks at the TCP Level. Tech. Note (August 2005), `http://www.securityfocus.com/archive/1/408135`
26. Corona, I., Giacinto, G.: Detection of Server-side Web Attacks. In: Proc of JMLR: Workshop on Applications of Pattern Analysis, pp. 160–166 (2010)
27. Jain, A., Murty, M., Flynn, P.: Data clustering: a review. ACM Computing Surveys 31(3), 264–323 (1999) ISSN 0360-0300