Detecting Anomalous Network Traffic with Self-Organizing Maps[*]

Manikantan Ramadas, Shawn Ostermann, and Brett Tjaden

Ohio University
[mramadas|ostermann]@cs.ohiou.edu

James Madison University
tjadenbc@jmu.edu

Abstract. Integrated Network-Based Ohio University Network Detective Service (INBOUNDS) is a network based intrusion detection system being developed at Ohio University. The Anomalous Network-Traffic Detection with Self Organizing Maps (ANDSOM) module for INBOUNDS detects anomalous network traffic based on the Self-Organizing Map algorithm. Each network connection is characterized by six parameters and specified as a six-dimensional vector. The ANDSOM module creates a Self-Organizing Map (SOM) having a two-dimensional lattice of neurons for each network service. During the training phase, normal network traffic is fed to the ANDSOM module, and the neurons in the SOM are trained to capture its characteristic patterns. During real-time operation, a network connection is fed to its respective SOM, and a "winner" is selected by finding the neuron that is closest in distance to it. The network connection is then classified as an intrusion if this distance is more than a pre-set threshold.

Keywords : Intrusion Detection, Anomaly Detection, Self-Organizing Maps

1 Introduction

We have seen an explosive growth of the Internet in the past two decades. As of January 2003, the Internet connected over 171 million hosts [12]. With this tremendous growth has come our dependence on the Internet for more and more activities of our lives. Hence, it has become critical to protect the integrity and availability of our computer resources connected to the Internet. We have to protect our computer resources from malicious users on the Internet who try to steal, corrupt, or otherwise abuse them. Towards this goal, intrusion detection systems are being actively developed and increasingly deployed.

Intrusion detection systems have commonly used two detection approaches, namely, misuse detection and anomaly detection. The misuse detection approach uses a database of "signature"s of well known intrusions and uses a pattern

[*] This work was funded by the National Science Foundation under grant ANI-0086642

matching scheme to detect intrusions in real-time. The anomaly detection approach, on the other hand, tries to quantify the normal operation of the host, or the network as a whole, with various parameters and looks for anomalous values for those parameters in real-time.

Integrated Network Based Ohio University Network Detective Service (INBOUNDS) is an intrusion detection system being developed at Ohio University. INBOUNDS uses the anomaly detection approach to detect intrusions, and is network-based i.e., it can be used to passively monitor the network as a whole. In this paper, we present the Self-Organizing Map based approach for detecting anomalous network behavior developed for INBOUNDS.

We organize this paper as follows. In Section 2, we give a brief description of Self-Organizing Maps and follow up in Section 3 with details of related work in the intrusion detection domain based on Self-Organizing Maps. In Section 4, we describe the INBOUNDS system and the design of the Self-Organizing Map based module for detecting intrusions. In Section 5, we present some of our experimental results, and in Section 6 we conclude and give some recommendations for future work.

2 Self-Organizing Maps

The concept, design, and implementation techniques of Self-Organizing Maps are described in detail in [25]. The Self-Organizing Map algorithm performs a nonlinear, ordered, smooth mapping of high-dimensional input data manifolds onto the elements of a regular, low-dimensional array [25]. The algorithm converts non-linear statistical relationships between data points in a high-dimensional space into geometrical relationships between points in a two-dimensional map, called the Self-Organizing Map (SOM). A SOM can then be used to visualize the abstractions (clustering) of data points in the input space.

The points in the SOM are called neurons, and are represented as multidimensional vectors. If the data points in the input space are characterized using k parameters and represented by k-dimensional vectors, the neurons in the SOM are also specified as k-dimensional vectors.

2.1 Learning

In the SOM Learning phase, the neurons in the SOM are trained to model the input space. This phase has the following two important characteristics:

- **Competitive** Each sample data point from the input data space is shown in parallel to all the neurons in the SOM, and the “winner” is chosen to be the neuron that responds best. The k-dimensional values of the winner are adjusted so that it responds even better to similar input.
- **Cooperative** A neighborhood is defined for the winner to include all neurons in its near vicinity in the SOM. The k-dimensional values of neurons in the neighborhood are also adjusted so that they too respond better to a similar input.

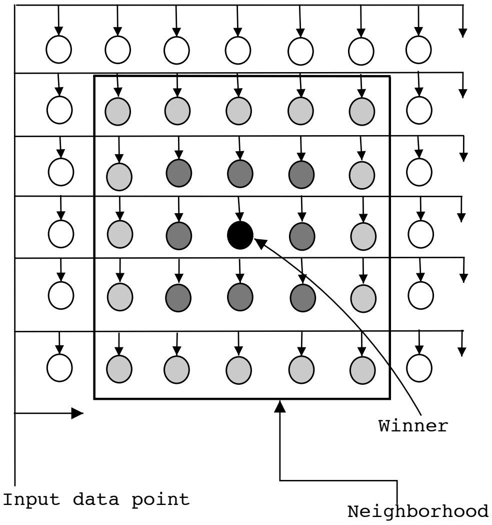


Fig. 1. SOM Learning

The SOM learning principle, illustrated in Figure 1, shows the SOM, with the circles representing neurons. The input data point is fed in parallel to all the neurons in the SOM. The winner neuron is colored black, and a square of length 5 centered around it represents the neighborhood.

During the learning phase, samples of data are collected from the input space and "shown" to the SOM. For this purpose, sample vectors representing input data covering the range of operational behavior are collected. The neurons in the SOM are initialized to values chosen from the range of sample data. The neurons can be assigned values linearly in the range (linear initialization), or assigned random values within the range (random initialization).

Distance Measure For the purpose of locating the winner neuron given the data sample, a suitable measure of distance has to be defined. The commonly used distance measures are the Euclidean and the Dot-product measures. In the Euclidean measure, given two points X $(x_1, x_2, \ldots, x_k)$ and Y $(y_1, y_2, \ldots, y_k)$ in k-dimensional space, the Euclidean distance is given by

$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_k - y_k)^2}$$

If the Dot-product measure is to be used, the input data points and the neurons in the SOM have to be normalized. Normalization of a vector V $(v_1, v_2, \ldots, v_k)$ is a process of transforming its components into

$$\left(\frac{v_1}{\sqrt{{v_1}^2 + {v_2}^2 + \ldots + {v_k}^2}}, \frac{v_2}{\sqrt{{v_1}^2 + {v_2}^2 + \ldots + {v_k}^2}}, \ldots, \frac{v_k}{\sqrt{{v_1}^2 + {v_2}^2 + \ldots + {v_k}^2}}\right)$$

so that the modulus of the normalized vector is unity. The dot-product of the input data point is calculated individually with each of the neurons, where the dot-product of two normalized vectors X $(x_1, x_2, \ldots, x_k)$, and Y $(y_1, y_2, \ldots y_k)$ is defined to be

$$x_1.y_1 + x_2.y_2 + \ldots + x_k.y_k$$

The winner is selected to be the neuron that gives the maximum dot product.

Neighborhood function The neighborhood function determines the size and nature of the neighborhood around the winner neuron. The commonly used neighborhood functions are Bubble and Gaussian. In the Bubble function, the neighborhood radius is specified by a variable σ, and all neurons within the neighborhood are adjusted by the same factor α towards the winner. The parameter α, called the learning rate factor, and the neighborhood size σ, are generally chosen to be monotonically decreasing functions of time t, where t is a discrete time measure incremented with every iteration of the training process.

The Bubble neighborhood function $h_{ci}(t)$ is specified as :

$$h_{ci}(t) = \begin{cases} \alpha(t) & ||r_c, r_i|| < \sigma(t) \\ 0 & \text{otherwise} \end{cases}$$

where r_c and r_i represent the positions of the winner c and neuron i in the SOM, and $||r_c, r_i||$ is the distance between them.

The Gaussian neighborhood function adjusts the winner neuron the most towards the sample data, and adjusts the remaining neurons within the neighborhood lesser and lesser as their distance from the winner increases, based on a bell shaped Gaussian function. It is specified as :

$$h_{ci}(t) = \begin{cases} \alpha(t)exp\left(-\frac{||r_c,r_i||^2}{2\sigma^2(t)}\right) & ||r_c, r_i|| < \sigma(t) \\ 0 & \text{otherwise} \end{cases}$$

The Gaussian neighborhood function is illustrated in Figure 1. The winner shown in black is moved the most towards the sample data, while the neurons in the neighborhood are moved lesser and lesser as shown in the gray-shades.

Learning function After the winner is found and the neighborhood is determined, the k-dimensional values of the neurons are adjusted based on the learning function. The learning function is specified as :

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)]$$

where $m_i(t)$, $m_i(t+1)$ represent k-dimensional values of neuron i, at time t and $t+1$ respectively; $x(t)$ represents the k-dimensional values of the sample data.

To summarize, for every neuron in the SOM, the learning function calculates its distance from the sample data, and adjusts its k-dimensional values towards the sample data by a factor specified by the neighborhood function $h_{ci}(t)$.

2.2 Algorithm

The choice of the factors described in the previous section affect the nature of the SOM generated. Once these factors are decided upon, the following algorithm can be used to train the SOM.

After the SOM is initialized, the learning process is carried out in two phases. In the initial phase, a relatively large neighborhood radius is chosen. The learning rate factor $\alpha(t)$ is also chosen to have a high value, close to unity. This phase is carried out for relatively lesser number of iterations. Most of the map organization happens in this phase. In the final fine-tuning phase, a smaller neighborhood radius and smaller learning rate factor are chosen. This phase is carried out for relatively larger number of iterations. The adjustment done to the neurons are much smaller in this phase.

2.3 Operation

In a k-dimensional space, the sample data and the SOM neurons appear as points. During the course of the learning algorithm described above, the neurons "move" in the k-dimensional space to characterize the sample data as closely as possible. While clusters of neurons would form at spaces where the sample data points are concentrated, fewer neurons would represent the space where sample data occur sparsely.

During operation, a real-time sample can be fed to the SOM, and its winner located. It can be flagged normal if it is sufficiently closer to the winner, and flagged anomalous if its distance from the winner is more than a preset threshold.

3 Related Work

In the work cited in this paper, SOM-based profiles of various network services like web, email, etc., are built to perform anomaly-based intrusion detection. Using network-service profiles to perform intrusion detection is not new however. The paper [7] discusses the approach used to build statistical profiles of network services for the EMERALD [20] intrusion detection system. The paper [16] discusses a Neural network based approach to develop connection signatures of common network services.

Self-Organizing Maps have also been used in the past in the intrusion detection domain for various purposes. The paper [10] describes a Multi-level Perceptron/SOM prototype used to perform misuse-based intrusion detection. The paper [23] describes a system that uses SOMs as a monitor-stack to profile network data at various layers of the TCP/IP protocol stack. This system was used to detect buffer-overflow attacks by building profiles of application data based on percentage of bytes that were alphabetical, numerical, control, or binary. The paper [15] describes a system that uses Neural networks using the Resilient Propagation Algorithm (RPROP) to detect intrusions that uses SOMs for clustering and visualizing the data.

The paper [18] describes a host-based intrusion detection system that uses multiple SOMs to build profiles of user sessions, and uses them to detect abnormal user activities. The paper [17] describes an anomaly-based intrusion detection system that characterizes each connection based on the following features: Duration of the connection, Protocol type (TCP/UDP), Service type (HTTP/SMTP/etc.), Status of the connection, Source bytes (Bytes sent from Source to Destination), and Destination bytes (Bytes sent from Destination to Source). SOMs based on these features were used to classify network traffic into normal or anomalous. In our approach cited in this paper, we characterize each network connection based on a different set of features, and build SOMs for each individual network service of interest.

4 SOM-based Anomaly Detection

In this section, we describe the Anomalous Network-traffic Detection using Self-Organizing Maps (ANDSOM) module used by INBOUNDS for detecting intrusions. We briefly describe the design of the INBOUNDS system as a whole, and then describe the ANDSOM module in detail.

4.1 INBOUNDS Architecture

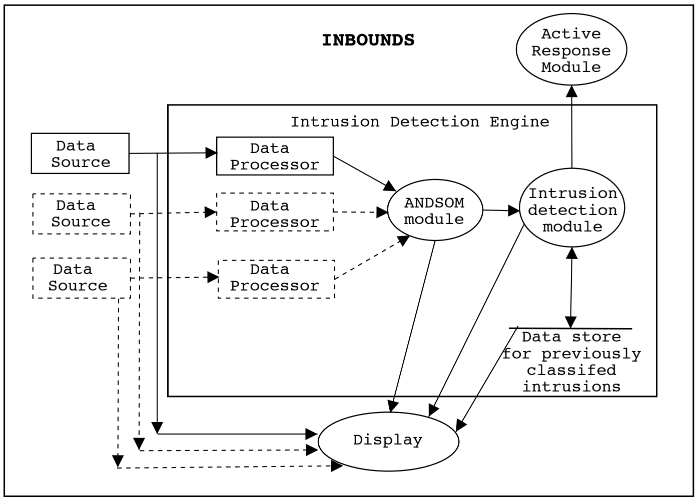


Fig. 2. INBOUNDS Architecture Diagram

The INBOUNDS Architecture diagram is shown in Figure 2. Some of the modules of the INBOUNDS system are currently under development. The goal of this section is only to present a high-level view of the INBOUNDS system, so as to give proper context to describe the ANDSOM module.

The Intrusion Detection Engine is the heart of the INBOUNDS system. Multiple Data Source modules feed real-time network data into the engine. This engine makes a decision on whether a network connection looks normal or anomalous. The Display module shall display real-time network traffic seen in the network on which INBOUNDS is being run, with a GUI front-end. The Active Response module takes response actions against intrusions. The response actions include being able to add a rule in the network firewall (if available) to block the traffic, to rate-limit traffic, to close the TCP connection by sending a RST packet to the sender etc. The Intrusion Detection module is present as a placeholder for a module that can make the final decision on whether the connection being analyzed is intrusive or not. The goal of this module is to incorporate the results of other modules, beside the ANDSOM module, and come up with a final decision. The modules that could be added in future include signature-based intrusion detection systems like SNORT [3] and modules based on other paradigms like the Bayesian module under development.

Data Source Module The Data Source module feeds live network data packets to the Intrusion Detection Engine. The program *tcpurify* [9] runs as the Data Source module, captures network packets off the wire, removes the application data from the packet and reports only the first 64 bytes of each packet, covering the IP and TCP/UDP protocol headers. The *tcpurify* program can also obfuscate the sender and receiver IP addresses and provide anonymity to the two hosts involved in the network connection during traffic analysis. This module is explained in detail in [22].

Data Processor Module This module receives the raw packets from the Data Source modules as input and runs the *tcptrace* [19] program with the INBOUNDS module. The INBOUNDS module for *tcptrace* reports the following messages for every connection seen in the network.

Open messages are reported upon seeing a new connection being opened in the network. The Open (O) message is of the format :

```
O TimeStamp Protocol <src host:port> <dst host:port> Status
```

The TimeStamp field reports the time the connection was opened. The Protocol field keeps track of if the protocol was TCP or UDP. Since UDP traffic doesn't have an implicit notion of connection unlike TCP, the Data Processor module groups UDP traffic between unique

$$< SourceIP, SourcePort, DestinationIP, DestinationPort >$$

4-tuples as connections, and uses a time-out mechanism to expire old connections. The $< srchost : port >$, and $< dsthost : port >$ fields together identify the end-points involved in the connection. The Status field is reported as 0 if both

SYN packets opening the connection were seen, and reported as 1 otherwise, for TCP connections. For UDP connections, the Status field is always reported as 0.

Update messages are reported periodically during the lifetime of the connection. The Update (U) message is of the following format :

```
U TimeStamp Protocol <src host:port> <dst host:port> INTER ASOQ ASOA QAIT AQIT
```

The period with which successive Update (U) messages are reported is tunable and defaults to 60 seconds. The INTER field reports the interactivity of the connection, defined as the number of questions per second seen during the past period. A sequence of data packets seen from the sender to the receiver constitute a single question until the receiver sends a packet carrying some data (pure TCP acknowledgments do not count), which would mark the end of the question. The answers are similarly defined for the receiver to sender direction. The ASOQ field reports the Average Size of Questions seen during the past period and the ASOA field reports the Average Size of Answers seen during the past period. The QAIT (Question-Answer Idle Time) field reports the idle time seen between seeing a question and an answer during the past period, averaged per second. The AQIT (Answer-Question Idle Time) similarly identifies the idle time between seeing an answer and a question, averaged per second, in the past period.

Close messages are reported when a previously open connection is closed in the network. The Close (C) message is of the following format :

```
C TimeStamp Protocol <src host:port> <dst host:port> Status
```

For TCP connections, the Status field is reported as 0 if both the FIN packets were seen during the connection close. If the connection was closed with a RST packet, the Status is reported as 1. UDP connections are reported as closed if they are found inactive for an expire-interval. The expire-interval is tunable, and defaults to 120 seconds. The Status field is always reported as 0 for UDP connections.

4.2 ANDSOM module - Training

The ANDSOM module implements the SOM-based approach for intrusion detection. In the training phase, SOMs are built to model different network services like web, email, telnet etc. For example, if we are trying to model web traffic in our network, a training data set is first collected by capturing dumpfiles from the network having a large number of sample web connections. It is important to make sure that intrusions themselves do not get into the training data set, since such intrusions may be perceived as normal by the SOM being built. For this, the signature-based intrusion detection system SNORT is run on the dumpfile, and connections reported by SNORT as intrusive are removed. However, it is still possible that we could have intrusions missed by SNORT if it had no rules to detect them. To make our system robust against this possibility, we could use multiple training data sets to generate multiple SOMs for each network service and run training data sets against other maps to prune out anomalies from getting into our model, as discussed in [23].

The submodules that make up the ANDSOM module are explained below.

TRC2INP submodule This submodule receives the 'O', 'U', and 'C' messages generated by the Data Processor module as input and generates six-dimensional vectors characterizing network connections. The parameters constituting the six dimensions are INTER, ASOQ, ASOA, L_QAIT, L_AQIT, and DOC. The INTER, ASOQ, and ASOA dimensions are calculated by averaging the INTER, ASOQ, and ASOA values from the 'U' messages received during the lifetime of the connection. The dimensions L_QAIT, and L_AQIT represent log base 10 values of the average QAIT and AQIT parameters calculated from the 'U' messages received during the course of the connection. The DOC dimension reports the Duration of Connection, and is the difference between the TimeStamps reported in the 'O' and 'C' messages of the connection.

The rationale behind using log base 10 values of QAIT and AQIT is to be more robust to false-positives. Since the QAIT and AQIT values tend to be relatively low in magnitude compared to other dimensions, if the QAIT value of a connection was reported as 0.0008 for example, and if the mean value found in the training data set was 0.0002, the connection was perceived to be four times the mean and had a high probability of being found anomalous. However, this might turn out to be a false-alarm, and what we might actually need is the order of QAIT and AQIT, whether they are in milli-seconds or micro-seconds etc., than the actual values themselves. We observed false-positives similar to the above example during our experiments, and decided to use the log value of the dimensions to mitigate the problem.

Normalizer submodule Using the six-dimensional vectors reported by the TRC2INP module to build SOMs directly tends to be biased to certain dimensions, as different dimensional values tend to be in different units. Normalizing all dimensions to values from 0 to 1, for example, could help, but still the dimension with the most variance would tend to dominate the SOM formation. Hence we use the following variance normalization procedure in this submodule to normalize the six-dimensional vectors.

The goal of variance normalization is to normalize the six-dimensional vectors of the training data set so that, in the set of normalized vectors, each dimension has a variance of unity. This normalization process is done in two steps. In the first step, the mean (μ) and standard deviation (σ) values are calculated for each of the six dimensions in the training data set. In the second step, a six-dimensional vector $< d_1, d_2, d_3, d_4, d_5, d_6 >$ is normalized to $< n_1, n_2, n_3, n_4, n_5, n_6 >$, by

$$n_i = \frac{d_i - \mu_i}{\sigma_i}$$

where μ_i and σ_i are the mean and standard deviations of dimension i.

At the end of the normalization process, the mean and standard deviation values found are stored in a data file for use in real-time operation.

SOM Training We used the public domain software packages SOM_PAK [5], and SOMTOOLBOX [4], during this phase. The SOM_PAK is a set of C programs that implement various stages of the SOM algorithm. The SOMTOOLBOX is a Toolbox for the MATLAB package, which we used for its graphical visualization functions.

SOM Initialization The SOMs were initialized with the *som_lininit* function from the SOMTOOLBOX. This function uses Principal Component Analysis [14] to arrive at the SOM dimensions by calculating the eigen values and eigen vectors of the auto-correlation matrix of the training data set. The orientation corresponding to the two largest eigen values, which are the directions in which the data set exhibits the most variance, is found. The ratio between the SOM dimensions are chosen based on the ratio of the two largest eigen values. The actual dimensions are chosen depending on this ratio and on the number of vectors in the training data set, and is explained in further detail in [22].
The SOM was chosen to be of hexagonal topology, and an Euclidean distance measure was used. After choosing the dimensions, the *som_lininit* function initializes the neurons in the SOM to values linearly chosen from the range of values in the training data set.

Initial Training Phase The *vsom* program from the SOM_PAK package was used to train the neurons in the SOM. The number of iterations of training in this phase were chosen to be low, in the order of a few thousands. In our experiments, for each network service, we typically had a few thousand samples, and this phase was done so that each sample was shown at most once to the SOM being built. The Gaussian neighborhood function was used with an initial neighborhood radius as the lower of the SOM dimensions. The neighborhood radius decreased linearly to 1 at the end of the training. The learning rate factor $\alpha(t)$ was chosen to be 0.9 and reduced linearly to 0 at the end of training.

Final Training Phase The *vsom* program was used again in the final training phase, and the number of iterations of training was chosen to be high, in the range of 100,000s. The number of iterations was set to be 500 times the product of the lattice dimensions (based on the heuristic recommended in [25]). The Gaussian neighborhood function with a low initial neighborhood radius of 5, and a low learning rate factor of 0.05 were set, and the map was fine-tuned in the Final Training Phase.

SOM Validation We evaluate the SOM at the end of the training phase, by feeding back the training data set to the SOM, and calculating the distance to winner for each of the samples. We validate the SOM if at least 95.44% of sample vectors in the training data set had a winner within 2 units of distance. This heuristic assumes the training data set to follow Gaussian distribution. If the data in the training data set were to follow Gaussian distribution strictly, 95.44% of the samples must fall within 2 units of standard deviation from the mean, according to properties of Gaussian distribution. If the 95.44% heuristic is not met at the end of training, the samples that do not have a winner within 2 units are shown more often to the SOM, and the training is repeated.

Although this heuristic lets the SOM model capture the behavior exhibited by the bulk of traffic, it does not capture the out-lier behavior exhibited by fewer connections. These outliers may give rise to false-positives with our model. If we were to raise the heuristic and try to capture more outliers into our model, we could get rid of some false-positives, but would be subjecting ourselves to more false-negatives. Note that to include an out-lier in our model, we need a SOM neuron within 2 units of standard deviation from it in six-dimensional space, as all connections with winner more than 2 units distant will be classified as anomalous. When we have such a neuron, all connection samples within the six-dimensional hyper-sphere of radius 2 units from the neuron will be classified as normal traffic. If an attack were to fall in such a hyper-sphere surrounding such a neuron, it would give rise to a false-negative. Thus, as we increase our heuristic value, we would add a lot of such hyper-spheres for the outliers into our model and the SOM would tend to get more and more general in nature, losing its specificity of modeling only the network service of interest.

We believe that our 95.44% Gaussian heuristic is a reasonable value to capture the characteristics exhibited by the bulk of traffic. However, our experiments need to be repeated with various threshold percentages for the heuristic as a future work, to study the trade-off more thoroughly.

4.3 ANDSOM module - Operation

During real-time operation phase, the ANDSOM module receives the 'O', 'U', and 'C' messages of connections from the Data Processor module. These messages are converted to six-dimensional vectors by the TRC2INP module. If a SOM was built for that network service, it is normalized based on the mean (μ_i) and standard deviation (σ_i) values found from the training data used to build the SOM. The normalized vector is then fed to the SOM, and the winner is found. The network connection is classified as anomalous, if the distance to the winner was more than 2 units.

5 Experimental Results

In this section, we describe our experiments with the SOM models for Domain Name System (DNS) and web (Hyper-Text Transfer Protocol) traffic, and analyze the performance of the models built.

5.1 DNS

DNS [21] traffic runs on top of both TCP and UDP. Although some DNS connections are observed on top of TCP, typically when two name servers transfer bulk domain information, the bulk of DNS traffic is found to be on top of UDP, and involve simple query-response of domain information. We collected dumpfiles from our network yielding 8857 sample DNS connections, and a SOM of dimensions 19x25 was built and linearly initialized. The mean and standard deviation values of this data set found by the Normalizer submodule shown in Table 1 illustrate the traits of DNS traffic found in our network.

Table 1. DNS Training Data Statistics

Dimensions	Mean	Standard Deviation
INTER	0.653	0.701
ASOQ	29.082	19.831
ASOA	112.352	94.651
L_QAIT	-1.142	1.376
L_AQIT	-0.016	0.186
DOC	2.033	1.056

The mean INTER value of 0.65 and the DOC value of 2 seconds, indicate that a DNS connection has 1.3 questions asked during the course of a connection. This is expected since the bulk of DNS connections involve a single query-response. The relative mean values of ASOQ and ASOA indicate that the answers tend to be much bigger than the questions, which is expected since DNS responses tend to be much bigger than DNS queries in general. The mean L_QAIT indicates that the QAIT value tends to be in hundredths of second per second. The L_AQIT value close to 0 corresponds to an AQIT value close to a second per second and is expected because single query-response traffic has no request (question) following a response (answer). The AQIT value is hence reported as the maximum value of 1 second per second for these connections, causing the mean L_AQIT value to be close to 0.

Table 2. DNS Exploit Vector

INTER	ASOQ	ASOA	L_QAIT	L_AQIT	DOC
1.989	493.000	626.000	-2.847	-2.375	1.006

For testing the SOM model, we generated anomalous DNS traffic with an attack based on an exploit of the BIND [11] DNS server. BIND server version

Table 3. DNS Normalized Exploit Vector

INTER	ASOQ	ASOA	L_QAIT	L_AQIT	DOC
1.906	23.393	5.427	-1.239	-12.664	-0.973

Table 4. DNS Winner Neuron

INTER	ASOQ	ASOA	L_QAIT	L_AQIT	DOC	Distance
0.708	6.072	-0.799	0.150	-0.212	-0.128	22.314

8.2.x was run as a vulnerable server in our test-bed. The exploit [1], available in the public domain, is based on the buffer-overflow vulnerability [6] in processing Transaction Signatures found in 8.2.x versions.

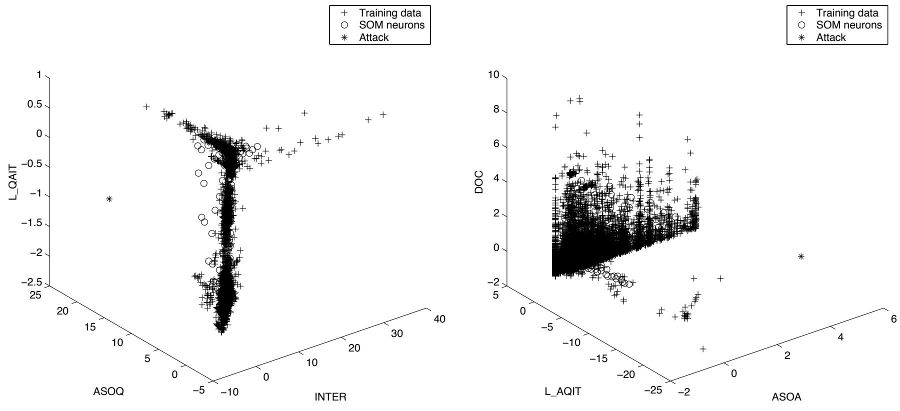
Table 2 shows the six-dimensional values of the DNS exploit vector, and Table 3 shows the normalized values of the DNS exploit vector, based on the mean and standard deviation values found in the training data set. We can observe from these tables that the ASOQ value of 493 bytes is highly anomalous with a distance of 23.393 standard deviations, since the training data set had a mean ASOQ value of 29 bytes. The ASOA value of 626 bytes is also anomalous from the training data set with 5.427 standard deviations away from the mean ASOA of 112.35 bytes. Further, the L_AQIT value of the -2.375 indicates that the actual AQIT was in the order of milli seconds per second. This happens to be highly anomalous with a normalized value of -12.664 standard deviations because the mean L_AQIT was in the order of -0.016, corresponding to an AQIT value of close to one second per second.

The winner neuron for the DNS exploit, and its distance to the winner in six dimensional space, are shown in Table 4. We can see that the winner neuron was at a distance of 22.314 standard deviations in the six-dimensional space, resulting in the DNS exploit being successfully classified with our intrusion threshold of 2 units.

To aid in the visualization of the six-dimensional space, we split the space into two three-dimensional views. The dimensions that take the X, Y, and Z axes of the two views were chosen arbitrarily with the goal of showing the attack point from the training data points clearly. The two three-dimensional views are shown in Figure 3.

5.2 HTTP

We built a SOM to model web traffic based on the HTTP [8] [13] protocol. A training dataset of 7194 HTTP connections collected from our network was used, and a SOM of dimensions 16x27 was built and linearly initialized. The mean



(a) View 1

(b) View 2

Fig. 3. DNS Exploit View (Units : Standard-Deviations from Mean)

and standard deviation values of the training data set found by the Normalizer submodule are shown in Table 5.

The mean interactivity of a HTTP connection is close to 0.8 questions per second. The mean size of questions is an order of magnitude smaller than the size of the answers, implying that more data seems to come from web-servers to web-clients than in the other direction. However ASOQ and ASOA tend to be highly variant as indicated by their high standard deviations. The QAIT seems to be in hundredth-s of a second per second, which could correspond to the fact that web-servers tend to be across the Internet causing the delay between the questions and answers. The AQIT value is in the order of ten-thousandths of a second per second and seems to indicate the fact that it takes very less time for a web-client to generate the next question, once the answer to a previous question is received. The mean duration of a HTTP connection is 9 seconds, although this duration is found to be highly variant with a standard deviation of 27.

We used the HTTP Tunnel [2] program to generate anomalous HTTP traffic in the network. The HTTP Tunnel program creates application-layer HTTP tunnels between two hosts, and lets any type of traffic to be run on top of HTTP. The HTTP tunnel program can be used by attackers inside an organization to break firewall rules. For example, assuming an organization firewall allows traffic to a host A on HTTP port 80, a malicious user inside the organization could setup an HTTP tunnel server on host A, and could let a user outside the organization on host B establish a telnet session to A by encapsulating all data as HTTP. The HTTP tunnel program uses the HTTP POST and GET methods to establish a duplex connection between two hosts; the POST method is used by the tunnel

Table 5. HTTP Training Data Statistics

Dimensions	Mean	Standard Deviation
INTER	0.829	0.773
ASOQ	589.120	743.973
ASOA	6802.338	59463.781
L_QAIT	-1.383	0.874
L_AQIT	-3.714	3.324
DOC	9.463	27.244

client on B to send data to A, and the GET method is used by the client on B to fetch data from A.

We used this program for our experiment by running a tunnel server in our test-bed, and establishing a telnet session to it from a tunnel client across the Internet. Although this traffic shows up as HTTP, we expect our model to classify it as anomalous since its connection characteristics might be different from normal HTTP traffic. The telnet connection was run on the HTTP tunnel for approximately 10 minutes, during which 13 connections (3 POST connections and 10 GET connections) were opened. We present the GET and POST connections that turned out to be highly anomalous amongst the 13 connections. The six dimensional vectors of those GET and POST connections are shown in Table 6 and the normalized values of these vectors are shown in Table 7.

Table 6. HTTP Tunnel Traffic

	INTER	ASOQ	ASOA	L_QAIT	L_AQIT	DOC
GET	0.004	17.200	22860.200	-5.699	-5.854	247.687
POST	0.023	491.667	0.000	-5.523	-10.000	307.706

Table 7. HTTP Normalized Tunnel Traffic

	INTER	ASOQ	ASOA	L_QAIT	L_AQIT	DOC
GET	-1.068	-0.769	0.270	-4.937	-0.644	8.744
POST	-1.044	-0.131	-0.114	-4.735	-1.891	10.947

In the HTTP GET connection, a single query is made at the beginning of the connection by the client, and all replies from the server form a single answer.

Hence, the QAIT value is calculated only once, when the first data packet is seen on the tunnel from the server after the GET request is made. Such a QAIT value, calculated and normalized to a 60 second update interval, turns out to be very low, in the order of micro-seconds, which results in the L_QAIT value of -5.699, which is considered to be highly anomalous, being approximately -4.94 standard deviations from the mean. The duration of the connection (DOC) happens to be 247 seconds and is found to be highly anomalous with a distance of 8.74 standard deviations.

Similarly, since the POST connection lasts for 307 seconds approximately, the DOC dimension is considered highly anomalous. The ASOA value is found to be 0 bytes in Table 6 because all data in the POST connection flows from the tunnel client to the tunnel server, with only pure TCP ACKs arriving from the tunnel server. The L_AQIT value is also calculated to be -10.000 since no sample was available to calculate AQIT as there were no answers. The AQIT is found to be its initial value of 0 at the end. Since log base 10 of 0 is negative infinity, a low value of -10.000 is reported by the TRC2INP submodule. The L_QAIT is found to be anomalous with the value of -5.523, which corresponds to a QAIT value in microseconds. This again is due to the fact that no data flowed in the opposite direction, causing all data from tunnel client to server to be perceived as one question. The QAIT value was calculated when the FIN packet was seen on the connection. This happens to be low, as the first FIN packet seen is also sent from the tunnel client.

To summarize, both the GET and POST connections are found to be anomalous because the packet flow in both directions is found to be almost completely uni-directional, which is unusual for HTTP traffic, and because of the fact that the connections last a much longer time compared to the normal HTTP traffic used in training. The same winner neuron was found for both the GET and POST connection traffic, which is presented in Table 8.

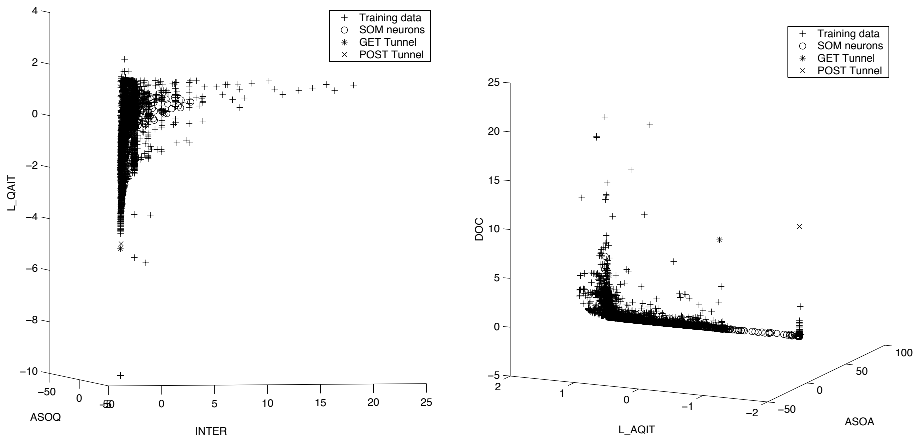
Table 8. HTTP Winner Neuron

	INTER	ASOQ	ASOA	L_QAIT	L_AQIT	DOC	Distance
GET	-0.953	-0.389	0.022	-1.460	1.131	5.895	4.855
POST	-0.953	-0.389	0.022	-1.460	1.131	5.895	6.743

Both the connections are classified as intrusions with the intrusion threshold of 2 units. The two three-dimensional views of six-dimensional space for HTTP traffic, are shown in Figure 4.

5.3 Performance Analysis

In this section, we present the Run-time and Modeling analyses of our system. Run-time analysis is aimed at estimating the feasibility of using the SOM-based



(a) View 1

(b) View 2

Fig. 4. HTTP Exploit View (Units : Standard-Deviations from Mean)

approach real-time, and the Modeling analysis is aimed at evaluating the efficiency with which the traffic characteristics of a network-service are modeled by its SOM.

Run-time Analysis We performed an off-line evaluation of the modules of the INBOUNDS system to estimate its run-time performance. Traffic from our department network was captured in dumpfiles of varying durations (15 min, 30 min, 45 min, 1 hour, 2 hours, and 3 hours), and the Data Processor module was run off-line on them to generate the 'O','U', and 'C' messages [Sec. 4.1]. These messages were fed to the TRC2INP module to generate six-dimensional vectors of network connections. The *locator* module normalized the vectors based on the HTTP training statistics (mean and standard deviation of the dimensions), fed it to the HTTP SOM, determined the winner neuron and the distance to it. The connection is classified anomalous if this distance was more than 2 units. Communication between the modules was through pipes, with each module reading its input from STDIN and writing out its output to STDOUT. Finally, as the goal was just to estimate the peak run-time performance, traffic from all network services found (not necessarily HTTP) was fed to the HTTP SOM.

Here, we present the results of the 1-hour, 2-hour, and 3-hour dumpfiles. The tests were performed on a GNU/Linux system running on a 800 MHz Pentium III processor with 256 MB RAM. The reader is referred to [22] for a more detailed analysis of the evaluation.

Table 9 illustrates the total bytes seen for various durations of capture, the Average Data Rate seen in that duration, total number of packets seen, the total

Table 9. Off-line Run-time Performance Analysis

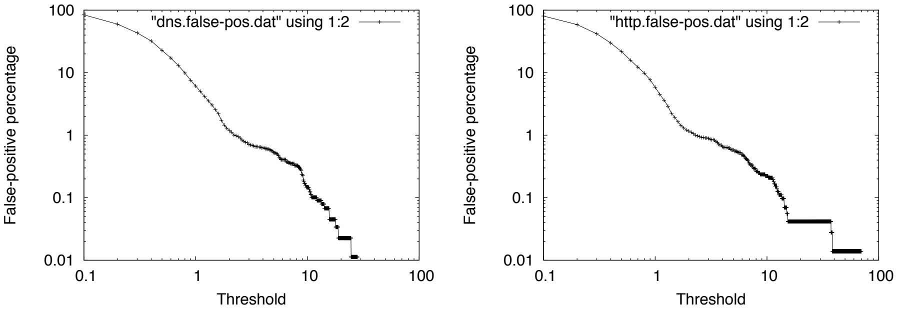
Duration (Hours)	Bytes	Avg. Data Rate (Mbps)	Packets	Conns.	Proc. Time (sec)	Proc. Rate (Mbps)
1	1,096,577,208	2.44	2,536,807	10,704	57.11	153.61
2	1,954,570,814	2.17	5,322,218	87,817	214.6	72.86
3	2,810,266,341	2.08	7,686,773	124,935	295.6	76.06

number of connections found, total Processing Time taken by all the modules together, and the corresponding Processing Rate of the modules. The net processing time depends on the size of the dumpfile and the number of packets and connections per unit time found in the network. Further, the bulk of the processing time is spent on the Data Processor module that needs to keep state of all active connections. For example, when the experiments were repeated to evaluate the time taken by individual modules for the 3-hour dumpfile, the Data Processor module took 254.9 seconds, the TRC2INP module took 21.9 seconds, while the *locator* module that actually implements the SOM algorithm took just 11.24 seconds. The SOM algorithm itself seems to be fairly light-weight in that once a trained SOM is available, it merely involves normalizing the vector based on the service statistics (mean and standard deviation of the dimensions) and computing the distance from the normalized vector to all the neurons in the SOM (taking linear time in the number of neurons in the SOM) and determining the winner.

Modeling Analysis To determine the modeling efficiency of the SOMs constructed, the amount of false-positives generated for the vectors from the training data set for different values of threshold (distance to the winner on which a connection is considered anomalous) was measured.

The percentage of false-positives generated for the DNS and HTTP SOMs shown in Figure 5 show the false-positive percentages as the threshold value is increased from 0.1 in steps of 0.1 until the threshold required to classify all the vectors in the training data set are classified as normal. For the threshold of 2 units used in our modules, which was sufficient to classify the attacks studied as anomalous, 1.18% and 1.16% of the DNS and HTTP training data set vectors give rise to false positives. The false-positive percentage drops exponentially, and increasing the threshold beyond 2 units seems to yield limited drops in the percentages of false-positives.

We also tested the HTTP SOM for streaming music and chat programs as these programs tend to run on HTTP port 80, and hence could be perceived as HTTP traffic. Streaming music connections (running on TCP) tend to be classified as anomalous, giving rise to false-positives. Such connections exhibit high ASOA values since unidirectional streams are treated as single huge Answers from the server to the client. Chat sessions are classified as anomalous too based



(a) DNS SOM

(b) HTTP SOM

Fig. 5. SOM Sensitivity

on the DOC dimension when they last for more than a couple of minutes, since the Mean and Standard-deviation values of DOC for the HTTP training data set were 9.5 seconds and 27.2 respectively. Although we can mitigate the false-positives rising out of such streaming music and chat sessions by adding multiple samples of them to the training data set and repeating the SOM training process, this could make the HTTP SOM to be more generic in nature, yielding false-negatives to attacks that resemble music streams/chat sessions.

6 Conclusions

The ability of the SOM based approach to correlate multiple aspects of a network connection (reported by the six parameters) to decide if it looks normal or abnormal, makes it a powerful technique for anomaly detection. The SOM model we built to characterize SMTP (email) traffic was also successful in detecting a Sendmail [24] buffer overflow attack, and is described in [22]. The SOM based approach seems to be particularly well suited to detect buffer-overflow attacks, as they tend to differ from the normal traffic behavior on the six dimensions.

However, the ANDSOM module may not detect attacks that resemble normal operational behavior. An intrusion massaged to resemble normal traffic might go un-noticed. Another limitation is that although the behavior exhibited by the bulk of traffic for a network service can be modeled, corner-case behavior occurring infrequently may be classified as intrusions, giving rise to false-positives.

For future work, it could be interesting to study the effects of modification to the SOM algorithm, including trying other neighborhood functions and different map topologies. It would also be interesting to construct and validate maps with various values of threshold with the Gaussian heuristic, and also by assuming other distributions of data besides Gaussian for the training data set.

References

1. Bind named 8.2.x Remote Exploit. http://downloads.securityfocus.com/vulnerabilities/exploits/tsig.c.
2. HTTP Tunnel. http://www.nocrew.org/software/httptunnel.html.
3. Snort - The Open Source Network Intrusion Detection System. http://www.snort.org.
4. SOM Toolbox for Matlab. http://www.cis.hut.fi/projects/somtoolbox.
5. SOM_PAK. http://www.cis.hut.fi/research/som_lvq_pak.shtml.
6. VU#196945 ISC BIND 8 Buffer Overflow in TSIG Handling Code. http://www.kb.cert.org/vuls/id/196945.
7. Phillip A.Porras and Alfonso Valdes. Live Traffic Analysis of TCP/IP Gateways. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security*, 1998.
8. T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0, May 1996. RFC 1945.
9. Ethan Blanton. TCPurify. http://irg.cs.ohiou.edu/~eblanton/tcpurify.
10. James Cannady and Jim Mahaffey. The Application of Artificial Intelligence to Misuse Detection. In *Proceedings of the First Recent Advances in Intrusion Detection (RAID) Conference*, 1998.
11. Internet Software Consortium. Bind. http://www.isc.org/products/BIND.
12. Internet Software Consortium. Internet Domain Survey, Jan 2003. http://www.isc.org/ds/WWW-200301/index.html.
13. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, June 1999. RFC 2616.
14. J.Hollmen. Principal Component Analysis. http://www.cis.hut.fi/~jhollmen/dippa/node29.html.
15. Chaivat Jirapummin, Naruemon Wattanapongsakorn, and Prasert Kanthamanon. Hybrid Neural Networks for Intrusion Detection System, 2002.
16. K.Tan and B.Collie. Detection and Classification of TCP/IP Network Services. In *Proceedings of the 13th Annual Computer Security Applications Conference*, 1997.
17. Peter Lichodzijewski, A.Nur Zincir-Heywood, and Malcolm I.Heywood. Dynamic Intrusion Detection Using Self-Organizing Maps. In *The 14th Annual Canadian Information Technology Security Symposium (CITSS)*, 2002.
18. Peter Lichodzijewski, A.Nur Zincir-Heywood, and Malcolm I.Heywood. Host-based Intrusion Detection Using Self-Organizing Maps. In *The IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks, IJCNN'02*, 2002.
19. Shawn Ostermann. Tcptrace - TCP Connection Analysis Tool. http://www.tcptrace.org.
20. P.A.Porras and P.G.Neumann. EMERALD : Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the National Information Security Conference*, pages 353–365, Oct 1997.
21. P.Mockapetris. Domain Names - Concepts and Facilities, November 1987. RFC 1034.
22. Manikantan Ramadas. Detecting Anomalous Network Traffic with Self-Organizing Maps. Master's thesis, Ohio University, Mar 2003. http://irg.cs.ohiou.edu/~mramadas/documents/MS-Thesis/thesis.pdf.
23. Brandon Craig Rhodes, James A.Mahaffey, and James D.Cannady. Multiple Self-Organizing Maps for Intrusion Detection. In *Proceedings of the 23rd National Information Systems Security Conference*, 2000.
24. The Sendmail Consortium. Sendmail. http://www.sendmail.org.
25. T.Kohonen. *Self Organizing Maps*. Springer, third edition, 2001.