DETECTING ANOMALOUS NETWORK TRAFFIC WITH SELF-ORGANIZING MAPS

A thesis presented to

the faculty of

the College of Engineering and Technology of Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

Manikantan Ramadas

March 2003

This thesis entitled

DETECTING ANOMALOUS NETWORK TRAFFIC WITH SELF-ORGANIZING MAPS

BY

MANIKANTAN RAMADAS

has been approved for

the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology by

Shawn D. Ostermann

Associate Professor of Electrical Engineering and Computer Science

Dennis Irwin

Dean, Russ College of Engineering and Technology

RAMADAS, MANIKANTAN. M.S. March 2003. Electrical Engineering and Computer Science

Detecting Anomalous Network Traffic With Self-Organizing Maps (78pp.)

Director of Thesis: Shawn D. Ostermann

Intrusion detection systems are aimed at distinguishing malicious network attacks from genuine network traffic. 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, characterized by six parameters, represents a vector in six-dimensional space. The ANDSOM module creates a two-dimensional lattice of neurons for each class of network traffic, with each neuron in the lattice specifying a six-dimensional vector. During the training phase, six-dimensional vectors of genuine network traffic are input into the ANDSOM module. The neurons in the lattice are trained to capture the characteristic patterns of genuine network traffic. During real-time operation, each network connection represented by a six-dimensional vector is input into the lattice, and a "winner" is selected by finding the neuron that is closest in distance in six-dimensional space. The network connection is then classified as an intrusion if this distance in six-dimensional space is more than a pre-set threshold.

Approved:

Shawn D. Ostermann

Associate Professor of Electrical Engineering and Computer Science

ACKNOWLEDGMENTS

I thank Dr. Ostermann, Dr. Tjaden, and all the INBOUNDS team members for their advice, feedback, and support.

I remain indebted to Dr. Ostermann, for taking me in his flock at IRG, for being an excellent teacher to learn Networking and Operating Systems from, and for being my mentor, motivator, and friend, all in one.

Finally, I would like to thank my parents, my uncle Mr. T.Prakash, my brother Kartik, and my best buddy Subramaniam, for all their love and support. Special thanks to my uncle, for always being there at the right time and right place for me; to my dad, for the constant words of encouragement; and to my mom, for the spirit to excel.

I dedicate this thesis to my mom.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

We have seen an explosive growth of the Internet in the past two decades. As of January 2002, the Internet connected over 147 million computers [15]. With this tremendous growth of the Internet, we have become more and more dependent on the it for various activities of our lives. The Internet has revolutionized the way we seek, process, and share information; the way we work; the way we do business; and has made geographical distances irrelevant for many purposes. Hence, it has become extremely important to secure the availability, integrity, and confidentiality of our computer network resources connected to the Internet.

Our computer network resources are more vulnerable when our network is part of the Internet than when our network is isolated, because our computers are potentially accessible to millions of users around the world when they are part of the Internet. Hence it is critical to protect our computer resources from compromise, to protect the integrity of stored data from malicious users who try to steal, corrupt, or otherwise abuse the data. Towards this goal, network intrusion detection systems are being developed and deployed.

Integrated Network Based Ohio University Network Detective Service (INBOUNDS), is an intrusion detection system being developed at Ohio University. This thesis details the development of a new intrusion detection approach for the INBOUNDS system.

We organize this thesis as follows. In this chapter, we give background information on intrusion detection systems, and discuss some of the current approaches in building them. We discuss the INBOUNDS system and the statistical approach for intrusion

detection used earlier in INBOUNDS. We then outline some of the limitations of this approach, and introduce Self-Organizing Maps, which form the basis for our new approach. In the second chapter, we describe the design of the Self-Organizing Map based module that we have developed, and how it fits in the INBOUNDS system as a whole. In the third chapter, we present our experimental results, detailing some of the intrusions detected successfully by our module. The final chapter gives conclusions and some recommendations for future work.

1.1 Intrusion Detection Systems

An "intrusion" of a resource is defined as "any set of actions that attempt to compromise the integrity, confidentiality, or availability of the resource" [39] and computer intrusion detection is defined as "the problem of identifying individuals who are using a computer system without authorization, and those who have legitimate access to the system but are abusing their privileges"[19]. Thus, the primary goal of a network intrusion detection system is to monitor a specific host computer or a network for intrusions, and to report detected intrusions.

An Intrusion Detection System (IDS) can be classified based on its scope of operation into host-based, multi-host-based, or network-based IDS. In a host-based IDS, the IDS runs on a host computer and monitors only the network activities of that particular host. A multi-host-based IDS has a hierarchical architecture, with host-based IDSs running on individual hosts and communicating the network activities they monitor to a master host. The multi-host-based IDS that runs on the master host corroborates the information provided by multiple hosts, and analyzes activity patterns of the network as a whole. In a network-based IDS, the IDS is run on a host that passively monitors the whole network.

In host-based and multi-host-based IDSs, an intrusion detection module has to be running on the monitored hosts, and the hosts are "aware" of the presence of such a module. In a network-based IDS however, intrusion detection modules are run

passively on a single-host that monitors the entire network; the hosts in the network may not be "aware" of the presence of an IDS in their network. An IDS can also be classified based on its intrusion detection approach into misuse detection or anomaly detection IDS.

1.1.1 Misuse Detection

An IDS based on misuse detection stores the network activity patterns of known intrusions as intrusion "signature"s in a database. The signature of an intrusion is the unique bit pattern that can be observed in the network when that intrusion occurs. A misuse detection IDS runs a pattern matching program that matches the network activity pattern of a live network connection with the signatures of all known intrusions. If a matching signature is found, the network connection is classified as an intrusion. Misuse detection IDSs are simple to configure and use, and are popular in the network security community.

The advantages of misuse detection IDSs are that they can detect intrusions with a fair amount of certainty, and can detect all intrusions whose signatures are available. However their disadvantage is that they cannot detect intrusions whose signatures are not in their database. Any new intrusion for which a signature is not available will go un-detected.

1.1.2 Anomaly Detection

An IDS based on anomaly detection identifies critical network activity parameters to monitor in a network. It then builds profiles storing normal operational values for the identified parameters. An intrusion is detected when a network activity is observed whose parameter values are sufficiently anomalous from the stored profile.

The advantage of anomaly detection IDSs is that they can detect never before seen intrusions. The disadvantages are that it could be difficult to come up with the correct set of network activity parameters to build profiles on, and that the probability of false-positives is relatively high.

1.2 INBOUNDS

Integrated Network Based Ohio University Network Detective Service (INBOUNDS), is a network based anomaly detection IDS being developed at Ohio University. The goals of INBOUNDS [17] are: run continually, be fault tolerant, be able to resist subversion, be scalable, operate with minimal overhead, be easily configurable, cope with changing system behavior, be difficult to fool with, and most importantly, detect never before seen attacks.

Prior to the work cited in this thesis, INBOUNDS performed anomaly detection using a statistical anomaly detection module [17]. We shall give a brief overview of its design and cite some of its limitations in the following sections.

1.2.1 Statistical Anomaly Detection Module

The statistical anomaly detection module identified five parameters to characterize individual network connections. These five parameters, also called as dimensions, were reported periodically; the time interval between such updates was a parameter that could be tuned.

- Interactivity - number of questions observed per second during the time interval.
- Average size of questions (ASOQ) - average size of questions observed during the time interval in bytes.
- Average size of answers (ASOA) - average size of answers observed during the time interval in bytes.
- Question-Answer idle time (QAIT) - the idle question-answer time observed per second during the time interval.
- Answer-Question idle time (AQIT) - the idle answer-question time observed per second during the time interval.

A sixth dimension, total Number of Connections (NOC), kept track of the total number of connections on a specific port, i.e., specific type of network traffic. Network

traffic was analyzed by *tcptrace* [34], a TCP/IP network traffic analyzer tool. A real-time module was added to *tcptrace* to report the opening, activity, and closing of network connections in the network. The real-time module for *tcptrace* reported the following messages :

- An 'O' message when a new network connection was started in the network.
- An 'I' message periodically during the life time of the connection. This period was tunable, and was typically set to 60 seconds. The 'I' message included the values observed for the five dimensions during the past period.
- A 'C' message when an active network connection was closed in the network.

These 'O', 'I', and 'C' messages generated in real-time were given as input to the statistical anomaly detection module described in the following section.

1.2.2 Anomaly Detection

The anomaly detection module called the Network Anomaly Intrusion Detection (NAID) module [17], had two different approaches to monitor network activity, namely,

- All Connections to a Single Host (ACSH)
- All Connections to All Hosts (ACAH)

In the ACSH approach, all classes of network traffic destined to a specific host could be monitored. For example, the NAID module could observe email, ssh, web, and other classes of network traffic destined to a specific host in this approach. In the ACAH approach, a specific type of network traffic could be observed across all the hosts in the network. For example, in this approach, the NAID module could monitor the email traffic destined for all the hosts in the network. The ACAH approach could be useful to detect an e-mail macro virus which might affect all the e-mail servers in the network simultaneously. The ACSH approach on the otherhand, could help

detect the case when a single host is facing anomalous network traffic - which could happen when it is the victim of an intrusion attempt.

The NAID module, in both the ACSH and ACAH approaches, had two methods to detect intrusions, namely:

- Abnormality Factor method
- Moving Average method

1.2.2.1 Abnormality Factor Method

In the Abnormality Factor method, a database called the historical data repository was created to store the average and standard deviations of each of the six dimensions of network connections. This database was accessible by a key that could be either an IP address or port or both. During real-time operation of the NAID module, the six dimension values of network connections were compared with the corresponding six dimensions entry found in the historical data repository. Two parameters : Standardization Factor (SF) and Threshold (THRESH) were set. For each of the six dimensions, the difference between the current value and the average value stored in the historical data repository for a corresponding entry, was found. For each of the dimensions, the difference was divided by the respective standard deviation found in the historical database, to measure its distance in units of standard deviations. The ceiling value of the difference between the distance and the SF, was added across all dimensions to give the net distance DIST (If the ceiling value of the difference for any of the dimensions was negative, it was set to 0). If the DIST value was found to be greater than THRESH, an intrusion alert was raised.

1.2.2.2 Moving Average Method

In the Moving Average method, a moving, or sliding time window was employed. The window size was set to a specific time value T. The average values for each of the six dimensions were initialized to zero. During real-time operation, a moving average was calculated for each of the six dimensions by taking the average for all

the values collected during the past time window. An intrusion alert was raised if the observed six dimension values were significantly different from the moving average values calculated in the past time period 'T', i.e., when the difference exceeded a pre-specified threshold. The moving window was then slided, dropping off any data more than T time units in the past, and took the current data into the calculation of the moving average.

1.2.3 Limitations

Though the statistical approach for intrusion detection used in the NAID module was successful in detecting attacks similar to the mailbomb attack reported in [17], it has some known limitations.

The Abnormality Factor method captures the average and standard-deviations of all the six dimensions for a key, where a key could be an IP address, a port number, or both. Consider the case where we are trying to store the characteristics of a class of network traffic, say email traffic. Email traffic based on the SMTP [37] protocol runs in the TCP well known port 25. We capture genuine network connections on port 25 to build the profile with the average and standard deviation values of the six dimensions. The limitation with this method is that we only have one reference data point for SMTP connections in six-dimensional space, i.e., the average values of all six dimensions of port 25 traffic. During operation, an SMTP connection would be classified as an intrusion if the sample is more than a specified number of standard deviations "away" from this stored reference point in the six-dimensional space.

The distribution of SMTP training data in six-dimensional space is assumed to occur as a single cluster. If however, the training data were to occur as multiple distinct clusters separated from one another in six-dimensional space, having reference points at the centers of the clusters would better characterize the training data than having a reference point at a point equidistant from all clusters; further, such a middle point may itself not fall in any of the clusters and could fall in a void area, outside all clusters. Thus, the Abnormality Factor method could generate a false-positive for

a network connection found to be just outside a cluster boundary, which could have been avoided by having a reference point for each cluster of data.

The Moving Average method has the limitation of giving rise to false-negatives, i.e., intrusions getting classified as genuine network traffic, when an intrusion gradually raises the six dimension values in every moving-window time. It could also give rise to a lot of false-poitives. For example, in a University setting, the monitored network could be idle with no real traffic until students come to the computer lab. Assuming that all students come at 10:00 AM to the lab, the network traffic would suddenly soar above the moving average. This might be classified as an intrusion, turning out to be a false-positive.

Our work is motivated by the need to design a more powerful method to build profiles of genuine network traffic and overcome some of the limitations cited. We found the Self-Organizing Map algorithm [27] to be a good mechanism for profiling genuine network traffic. The Self-Organizing Map algorithm uses a lattice of neurons to capture the characteristic patterns of genuine network traffic. This lets us store multiple reference data points for each class of network traffic, which helps reduce the amount of false-positives. The following section gives a description of Self-Organizing Maps.

1.3 Self Organizing Maps

The Self Organizing Map (SOM) can be described as a software tool for the visualization and abstraction of complex high-dimensional data. The SOM may be defined formally as a nonlinear, ordered, smooth mapping of high-dimensional input data manifolds onto the elements of a regular, low-dimensional array [27]. A SOM converts non-linear statistical relationships between data points in a high dimensional space into geometrical relationships between points in a two-dimensional map. Thus, a SOM compresses information from a high dimensional input space to a low dimensional output space and produces abstractions of data points from the input space.

The SOM algorithm has been known to model the way various topographical response areas are formed in the human brain. It has been found to model maps of acoustical frequencies, phonemes of speech, maps of elementary optical features [26] etc. The SOMs are also being successfully used in practical applications such as automatic speech recognition, image analysis, industrial process control [24, 21].

The SOM algorithm described in this section is used to form a self-organizing map from a two dimensional array of elements. Each element of the map called a "Neuron", after the biological neurons in the human brain on which they are modeled, is specified by a multi-dimensional vector.

The goal of the SOM algorithm is to model data points from a complex input signal space into a two dimensional lattice of neurons in the self-organizing map. If each individual input signal is characterized by k parameters, we can represent it as a point in k-dimensional space specified by a vector of k dimensions. The neurons in the SOM, chosen to be vectors of k-dimensions, also appear as points in this k-dimensional space.

1.3.1 Learning

In the learning phase, neurons are trained to model the input data points in the k-dimensional space. This learning phase has the following two important properties:

- **Competitive** Each data point is fed in parallel to all the neurons in the map. The neuron that responds best is selected as the "Winner", and its k dimensional values are adjusted so that it responds even better for a similar input data point in the future.

- **Cooperative** A neighborhood is defined for the winner neuron, to include all the neurons in its near vicinity. The k dimensional values of the neurons in the winner's neighborhood are also adjusted, so that they too respond better for a similar input data point in the future.

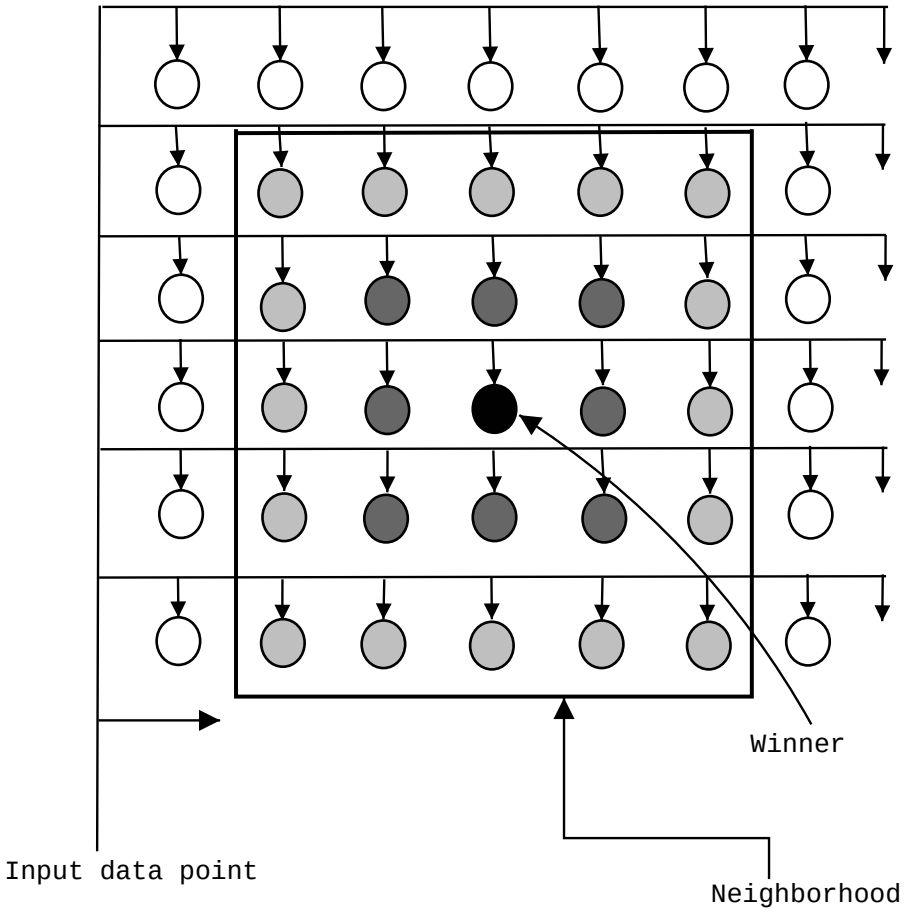

Figure 1.1. SOM Learning Scheme

In Figure 1.1, we have a lattice of neurons with each neuron being represented by a circle. Each input data point is fed in parallel to all the neurons in the lattice. The neuron that responds best to the input data point is identified as the winner. The winner is shown as the circle shaded black. The neighborhood of the winner, defined in this case to be of radius 2, is shown as a square of size 5, with the winner at the center. The winner neuron is adjusted the most towards the input data point, while other neurons in the neighborhood are adjusted too, with the adjustment factor getting lower and lower as the distance from the winner increase. All neurons outside

the neighborhood are left untouched. This adjustment factor is depicted in gray-scale shades, with the winner in black, having the maximum adjustment.

During the learning process, samples of data from the input space are "shown" to the neurons in the SOM. For this purpose, samples of data are collected from the input signal space encompassing various ranges of operational behavior. The neurons in the lattice have to be initialized within the range of operational behavior. However, the values used to initialize the neurons can be chosen linearly covering the range, called linear initialization; or values can be chosen randomly within the range and assigned, called random initialization. The SOM algorithm is known to converge to its final values in either case, though it converges faster in the latter case. The SOM lattice could be chosen to be hexagonal or rectangular in nature. The hexagonal lattice type however, is known to have better visualization properties.

1.3.1.1 Distance Measure

For the purposes of locating the winner for input data samples, a suitable distance measure has to be defined. The most common distance measure is the euclidean distance measure. Viewing the input data point and all neurons as points in the k dimensional space, the winner is identified by locating the neuron that is closest in euclidean distance to the input data point. For 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

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

Another commonly used distance measure is the dot-product measure. The input data points and the neurons in the lattice have to be normalized before using this measure. Normalization of a vector V $(v_1, v_2, \ldots, v_k)$ is a process of transforming its components into $(\frac{v_1}{\sqrt{{v_1}^2+{v_2}^2+\ldots+{v_k}^2}}, \frac{v_2}{\sqrt{{v_1}^2+{v_2}^2+\ldots+{v_k}^2}}, \ldots, \frac{v_k}{\sqrt{{v_1}^2+{v_2}^2+\ldots+{v_k}^2}})$ 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 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.

1.3.1.2 Learning Function

During the learning process, the k-dimensional values of the neurons in the lattice are adjusted as specified by the learning function :

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

where t is a discrete time measure that gets incremented by 1 during every iteration of the training process.

x(t) represents the input data point chosen during the iteration t of the learning process.

$m_i(t)$ and $m_i(t+1)$ specify the vector measures of neurons at distance i from the Winner, during iterations t and t+1 respectively.

$h_{ci}(t)$ is the neighborhood function such that, $h_{ci}(t) = \mathrm{h}(||r_c, r_i||, \mathrm{t})$, is a function of $||r_c, r_i||$, and t. Here, r_c, r_i are the locations of the winner and neuron i in the lattice, and $||r_c, r_i||$ is the distance between them.

Two neighborhood functions are commonly used, namely bubble and gaussian. In the bubble function, a variable N_c defines the neighborhood radius for the winner. For example, if N_c were set to 3, all neurons within a circle of radius 3 from the winner r_c would be considered to be in the neighborhood. The bubble function could also be specified as $N_c(t)$ so that the neighborhood radius can shrink over time during the training process. The bubble function is specified as $h_{ci}(t) = \alpha(t)$ for all neurons within $N_c(t)$ of the winner r_c, where $\alpha(t)$ is called as the learning rate factor that affects how much the neurons are adjusted towards the winner. Both $N_c(t)$ and $\alpha(t)$ are typically chosen to be monotonically decreasing functions over time.

In the bubble function, all neurons in the neighborhood are adjusted by the same amount. The gaussian function on the other hand, lets the adjustment factor vary as a bell shaped gaussian function, with the maximum adjustment for the winner and

progressively lesser and lesser adjustment as the distance from the winner increases. The gaussian function is specified as:

$$h_{ci}(t) = \alpha(t) \exp\left(- \frac{||r_c, r_i||^2}{2\sigma^2(t)} \right)$$

where $\sigma(t)$ specifies the neighborhood radius.

1.3.1.3 Learning Process

The map dimensions, neighborhood radius, learning rate, and the number of iterations used in training, are all factors that are crucial to the formation of good maps during the learning process. Another important requirement is to capture a sample data set. A sample data set is a collection of snapshots of operation of the system being modeled. It is important that this data set encompasses all possible range of operations of the system.

Learning process typically involves three steps :

- Initialization: The lattice of neurons can be initialized either randomly or linearly. The initial values assigned to the neurons chosen randomly or linearly, should be in the range of values seen in the sample data set.

- Initial Learning Phase: In this phase a large initial neighborhood radius is kept and a large learning rate factor $\alpha(t)$ - typically 0.9 is set. The input data points are fed to all the neurons in the lattice, a winner is selected, and the winner and its neighborhood neurons are adjusted. This phase is carried out typically for a few thousand iterations. Most of the learning happens in this stage.

- Final Learning phase: This phase is for the fine adjustment of the map. Low values are set for the learning rate factor $\alpha(t)$, typically 0.05, and a relatively low neighborhood radius is chosen. However the number of iterations of training is chosen to be relatively high, typically in the order of hundred thousand iterations. A rule of thumb is to carry out this phase for 500 times the number of neurons in the lattice [27].

1.3.2 Operation

Once a SOM is trained according to the learning process cited above, it could be used for real-time operations. How a SOM is used could vary from application to application. For example, SOMs can be used to capture phonetic signal information for the automated speech processing applications. For such a SOM, during real-time operation, a vector representing an uttered sound could be fed to the SOM and it could locate the winner in the lattice. The phonetic corresponding to the uttered sound could be associated with the phonetic the winner responds best to.

To summarise, the basic idea behind the operation phase is to feed the input data point to the lattice, determine the winner, and associate the input data point to that input value the winner responds best to.

1.3.3 Software Packages

Software packages implementing the SOM algorithm are available in the public domain. We used the SOM_PAK [10] and the SOMTOOLBOX [11] software packages for our experiments.

Both SOM_PAK and SOMTOOLBOX are from the Laboratory of Computer Sciences, Helsinki University of Technology, Finland. SOM_PAK consists of a package of C programs with a UNIX style command line interface for initializing, training and testing self-organizing maps. These programs have options to choose values for parameters in the SOM learning process like the learning rate factor, neighborhood radius, and the number of iterations, during the training phase.

SOMTOOLBOX is a toolbox developed for the MATLAB package. The C programs that were part of the SOM_PAK package have been implemented in the MATLAB programming language in SOMTOOLBOX. It also includes powerful graphical visualization programs to aid in the visualization of the SOM algorithm.

2. ANDSOM Module For INBOUNDS

In this chapter we detail the design and working of the Anomalous Network traffic Detection with Self-Organizing Maps (ANDSOM) module that we have developed for INBOUNDS. We first describe the current architecture of the INBOUNDS system, and then describe how the ANDSOM module relates with the rest of the modules in the INBOUNDS system.

2.1 INBOUNDS Architecture

Figure 2.1 shows the current INBOUNDS architecture diagram. The INBOUNDS project is currently under development with some of the modules in their early stages of development. The goal of this section is to present a high-level view of the INBOUNDS system so as to give proper context for the description of the ANDSOM module.

The heart of the INBOUNDS system is the Intrusion Detection Engine. This engine makes the decision on whether the network connection being analyzed looks normal or if it is anomalous and should be classified as an intrusion. The following sections describe various modules part of the INBOUNDS system.

2.1.1 Data Source Module

The Data Source module provides network data packets as input to the Intrusion Detection Engine. Each network monitored by INBOUNDS requires a data source module.

The program *tcpurify* runs in the Data Source module. Since INBOUNDS focuses on anomaly detection, and not on misuse detection, application data in the packets are irrelevant for our purposes. Hence, *tcpurify* captures network packets from the

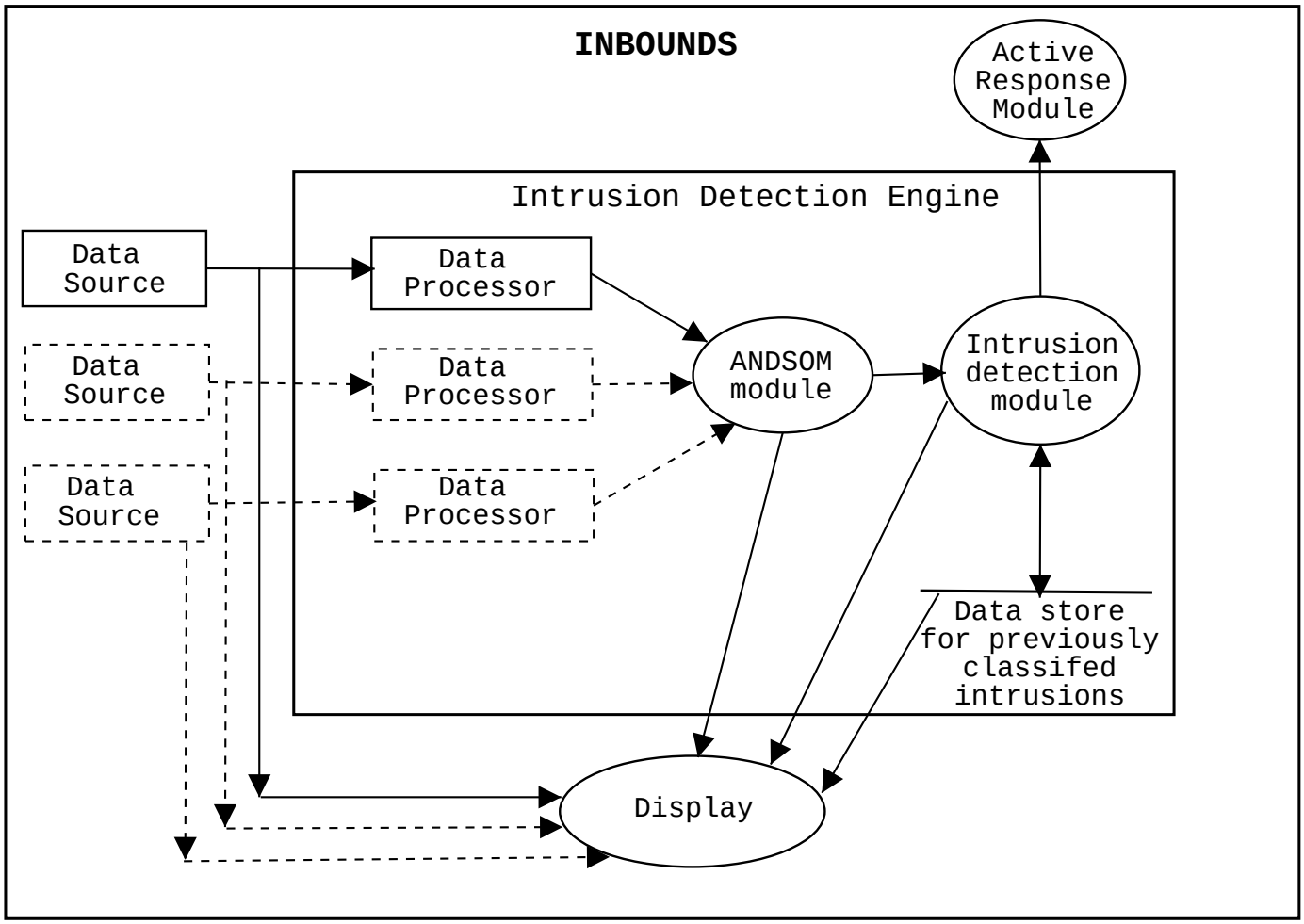

Figure 2.1. INBOUNDS Architecture Diagram

wire, wipes off the application data from the packet and reports only the first 64 bytes of each packet, typically covering the IP and TCP/UDP protocol headers. Any space remaining in the 64 bytes beyond the IP and TCP/UDP headers is set to zero.

The *tcpurify* program can also be used to obfuscate the sender and receiver IP addresses and provide anonymity to the two hosts involved in the network connection during traffic analysis. For obfuscating the IP addresses, *tcpurify* uses symmetric key encryption based on the RC5 [16] algorithm, using a 32-bit block size. Using just 32-bits for encryption weakens the quality of encryption; but this limit is from the size of IP addresses, and is necessary to be able to decrypt the encryption performed.

Two keys are used: a root key and a session key. A new session key is generated for each session of packet capture. The session key itself is encrypted with the root key, stored in a dummy ethernet packet, and is reported as the first packet seen. The source and destination IP addresses in the captured packets are changed to the values obtained by encrypting them with the session key. Then the TCP / UDP and IP checksums are re-calculated and changed in the packets. With the knowledge of the root key, and the session key (stored in the first packet), the encrypted IP addresses can be decrypted later, if necessary.

2.1.2 Data Processor Module

The Data Processor module receives the network packets provided by the Data Source modules as input, and runs the *tcptrace* program with the real-time inbounds module. The output of this module consists of messages reporting the opening, activity, and closing of live network connections. The inbounds module for *tcptrace* is based on the module specified in Section 1.2.1.

The Data processor module reports three types of messages:

- An 'O' message is of the format :

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

  The 'O' (open) message is reported when a new connection is opened in the

network. The TimeStamp field reports the time when the connection opened. The Protocol field indicates the protocol type TCP or UDP. The source and destination IP addresses and their respective ports are reported in the next fields. A status field indicates how the connection was opened. In case of TCP connection, a status value of 0 indicates that the open is proper, i.e., SYN packets were seen for the opening of this connection. The status value is reported as 1, if we didn't see the SYN packets opening the connection. For UDP connections, an 'O' message reporting the opening of a new connection is output when a packet is seen from one host to the other for the first time. The status field is always reported as 0 for UDP traffic.

- A 'U' message is of the format :

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

  The 'U' (update) message is reported periodically during the life time of the connection. This period is tunable and defaults to 60 seconds. The fields Inter (Interactivity), ASOQ (Average Size of Questions), ASOA (Average Size of Answers), QAIT (Question Answer Idle Time), and AQIT (Answer Question Idle Time) are the five dimensions reported. These fields have the same semantics as described in Section 1.2.1.

- A 'C' message is of the format :

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

  The 'C' (close) message is reported when an active connection is closed in the network. For TCP connections, the status field has a value 0 if the close was proper, i.e., two FIN packets were seen during the closure of the connection. If the connection was closed with a RST packet, the status field has the value 1. UDP connections get "closed" when they expire due to inactivity for a

specific period of time called expire interval. This expire interval is tunable, and defaults to 120 seconds. The status field is always reported as 0 for all UDP connections. TCP connections are also considered closed, if no activity is seen on the connection for a period of 8 hours.

2.1.3 ANDSOM Module

The Anomalous Network-traffic Detection with Self-Organizing Maps (ANDSOM) module uses the Self-Organizing Map algorithm described in Section 1.3 to build profiles of normal network traffic. The profiles built are later used to make a decision on whether a network connection is normal or anomalous.

The steps in the training of ANDSOM module are illustrated in Figure 2.2.

The Data Source and Data Processor modules are run on offline network dumpfiles containing the class of network traffic for which the SOM model is being built. For e.g., if we are building a SOM model for telnet traffic, multiple dumpfiles containing telnet traffic are collected during various times from the network. To make sure that well known intrusions themselves do not get into the model, the dumpfiles are processed with SNORT [9], a public domain intrusion detection system that does misuse detection. Any network connections reported as suspicious by SNORT are pruned from the dumpfiles.

The ANDSOM module receives the 'O', 'U', and 'C' messages from the Data Processor module which is run on the dumpfiles. A submodule, TRC2INP processes these messages and generates six dimension values used to characterize network connections. The six dimension values are then normalized by the Normalizer submodule and sent to the SOM Training submodule. At the end of training, a validation check is made. If successful, the ANDSOM module finishes the training; on failure at the validation phase, the training is repeated with certain changes to the training process, as outlined in the following sections.

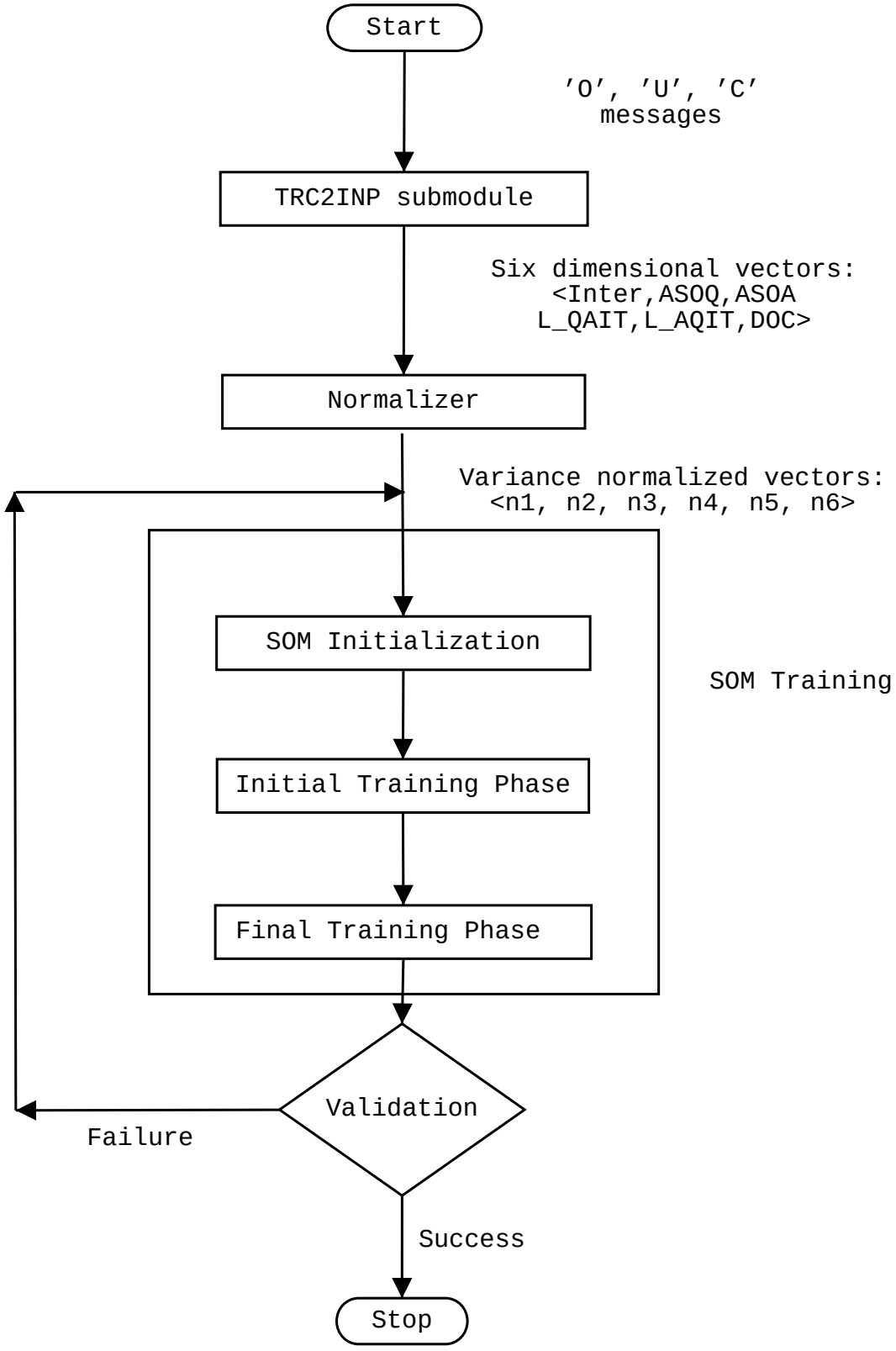

Figure 2.2. ANDSOM Training

2.1.3.1 TRC2INP Submodule

The TRC2INP submodule runs the *trc2inp* program. The *trc2inp* program reads the 'O', 'U', and 'C' messages produced by the INBOUNDS module as input, and for each connection, generates the six dimension values of Interactivity (INTER), Average Size of Questions (ASOQ), Average Size of Answers (ASOA), Log base 10 of Question Answer Idle Time (L_QAIT), Log base 10 of Answer Question Idle Time (L_AQIT) and Duration of Connection (DOC).

The *trc2inp* program maintains a hash table to store the state of active connections. Each entry in the hash table is indexed by the four tuple of < Source IP address, Source Port, Destination IP address, Destination Port > and can store the INTER, ASOQ, ASOA, QAIT, AQIT in fields dim1, dim2, dim3, dim4, and dim5. All the fields in a newly created entry are initialized to zero. Upon receiving an 'O' message, the hash value is calculated for the four tuple found from the message, and a new entry is added in the hash table. The timestamp found from the message is stored in the start_timestamp field of the entry.

Upon receiving an 'U' message, the four tuple of < Source IP address, Source Port, Destination IP, Destination port > in the message is used to locate the entry created for the connection. The INTER, ASOQ, ASOA, QAIT, and AQIT fields in the message are added to the dim1, dim2, dim3, dim4, and dim5 fields in the entry. The icnt field is incremented by 1, and a new average for dim1, dim2, dim3, dim4, and dim5 fields are obtained by dividing them by the icnt value. This procedure is repeated for every 'U' message received, storing the current average values of INTER, ASOQ, ASOA, QAIT, AQIT in the dim1, dim2, dim3, dim4, and dim5 fields respectively. Also, a count of the number of 'U' messages received during the life-time of the connection is stored in the field icnt.

Upon receiving a 'C' message, the entry corresponding to the network connection is removed from the hash table. The dim1, dim2 and dim3 fields are printed out as INTER, ASOQ, and ASOA - the first, second, and third dimensions. The log10 of

QAIT and the log10 of AQIT are printed as L_QAIT and L_AQIT - the fourth and fifth dimensions. The time difference is calculated between the timestamp found in the 'C' message and the start_timestamp field stored in the entry. This time difference is printed as DOC - the sixth dimension.

We decided to use the L_QAIT and L_AQIT (log10 of QAIT and AQIT respectively) as the fourth and fifth dimensions in place of QAIT and AQIT, to smooth out the false-positives that were found in our experiments when we used QAIT and AQIT. For e.g., if our profile had a normal average QAIT value of 0.00002 and we received a real-time connection with the value of 0.00009, it was perceived to be highly anomalous as the connection had a value about 4.5 times the mean. However, this might later prove to be a false-positive. We found that what we were really interested in was the order of QAIT and AQIT fields - whether it is in the order of milli seconds or micro seconds etc. Hence we decided that the fourth and fifth dimensions would be more useful, if we used the log base 10 of QAIT and AQIT in place of QAIT and AQIT respectively.

2.1.3.2 Normalizer Submodule

The six dimensions output produced by TRC2INP module are collected for each class of network traffic to train the SOM for that class of traffic. If each such vector of six dimensions is directly used to train the SOM, the dimension with high variance tends to dominate the map organization. Hence it is important to normalize each of the individual dimensions, so that the variance of each of the dimensions in the training data is unity.

We wrote the program *normalizer* to perform this normalization. The *normalizer* program acts in two passes, taking as input the set of six-dimensional vectors to be used for training. In the first pass it calculates the mean (μ) and standard deviation (σ) for each of the six dimensions. In the second pass, each of the six dimensional vectors $< d_1, d_2, d_3, d_4, d_5, d_6 >$ are normalized to yield the vector $< n_1, n_2, n_3, n_4, n_5, n_6 >$ such that $n_i = \frac{d_i - \mu_i}{\sigma_i}$ where μ_i and σ_i are the mean and standard deviations of dimen-

sion i in the training data set. The set of normalized vectors thus obtained, would have a variance of unity for each of the six dimensions, and can then be used to train the SOM.

At the end of the normalization process, the *normalizer* program also stores the mean and standard deviation values of the six dimensions in a data file.

2.1.3.3 SOM Training

We use the SOMPAK Self-Organizing Map software package and the SOMTOOLBOX toolbox for MATLAB specified in Section 1.3.3 in the training of the SOMs.

The *som_lininit* function from SOMTOOLBOX is used to initialize the neurons of the SOM lattice. The SOMs were chosen to have a two dimensional lattice of neurons with hexagonal topology. We used the euclidean distance measure for its simplicity of use.

The following method based on Principal Component Analysis [32] is used to arrive at the lattice dimensions. The goal of this method is to stretch the SOM in that orientation where the data exhibits the most variance. For this, the two eigen vectors corresponding to the two largest eigen values of input data are identified, and the lattice dimensions are chosen based on the ratio of the two largest eigen values.

For a given matrix A, eigen values and eigen vectors are found from the solution of the equation

$Ax = \lambda x$

where the solutions to λ are the eigen values, and the solutions to the vector x are the eigen vectors. If the matrix A is viewed as a representation of the transformation made to any input vector x, the eigen vector x can be said to identify the direction in which the effect of the tranformation matrix A, is equivalent to the linear transformation of the vector x by the scalar λ.

By calculating the auto-correlation matrix of the data set and calculating its eigen vectors and eigen values, the orientations in which the data set exhibits the largest variance can be found. These orientations are found along the direction of the two

eigen vectors corresponding to the two largest eigen values. Further details on the basis for this procedure can be found in [1].

The *som_lininit* function part of the SOMTOOLBOX for MATLAB, chooses the lattice dimensions as follows. The lattice dimensions are chosen proportional to the number of training data samples, and the number of neurons in the lattice. The number of neurons in the lattice(munits) in turn, depends on the number of training data samples (num), such that munits = 5*sqrt(num)

The auto-correlation matrix of the training data set is calculated. If the training data set has say 1000 samples, with each sample being a 6-dimensional vector, it can be viewed as a 1000x6 matrix M. The auto-correlation matrix A is found by calculating the matrix product MM^T where M^T is the transpose of M. The eigen values of the auto-correlation matrix are then found. The two biggest eigenvalues are then selected, and the ratio (r) between the largest and the second largest is calculated. The map dimensions are then chosen such that the ratio of the lattice dimensions is proportional to the square root of this ratio r. The dimensions are then adjusted such that the product of the dimensions is as close to the value munits as possible. Note that in the case of a hexagonal lattice, the sidelengths along y-axis are squeezed by a factor of $\sqrt{0.75}$ taking into account the geometrical properties of a hexagonal lattice.

Once the lattice dimensions and lattice size are found, the six dimensional vectors of the neurons in the lattice are initialized to values linearly chosen in the range of values of the six-dimensional vectors used in the training data set. Linear initialization was chosen over random initialization to aid in quicker convergence of the SOM algorithm.

The *vsom* program, part of the SOMPAK [10] software package is used to train the initialized SOM. Training is performed in two phases: an initial phase, and a final fine-tuning phase. The values for training parameters were chosen according to the heuristics recommended in [27].

Most of the map organization happens in the initial phase. In the initial phase, a gaussian neighborhood function is used. The number of iterations in training is chosen to be low, a few 1000 typically. The neighborhood radius is chosen to be high, typically the smaller of the map dimensions. This radius linearly reduces to a value of 1 at the end of training. The learning rate factor $\alpha(t)$ is set to a high value close to unity - typically 0.9 and linearly reduces to 0 at the end of training.

At the end of the initial phase, a final fine-tuning phase of the training begins. The number of iterations in training is chosen to be high, in the order of 100,000. We continue to use a gaussian neighborhood function, with a relatively small neighborhood radius. The learning rate factor is chosen to be low, typically 0.05 which reduces linearly to 0 at the end of training.

At the end of the two phases of training, we evaluate the trained lattice of neurons, with the initial un-normalized training data set. A program *locator* was written for this purpose. The *locator* program takes the initial un-normalized training data set, the mean and standard deviation of each of the dimensions of the data set found by *normalizer*, and the file storing the trained map, as input. It then normalizes each vector according to the mean and standard deviations as specified in Section 2.1.3.2 and feeds it to all the neurons in the trained map. It calculates the winner, and the distance between the winner and the normalized vector in six dimensional space. The output of *locator* has the normalized vector, the topographical location of the winner in the SOM, and the measured distance. This procedure is repeated for all neurons in the six dimensional space.

From the output of *locator*, we calculate the number of vectors in the training data set with distance from the winner more than 2 units. If the distribution of training data were perfectly gaussian, 95.44% of the samples must fall within 2σ of the mean, according to the properties of gaussian distribution. We use this as a heuristic to validate our maps. Our goal at the end of training is to have at least 95.44% of the training data vectors have a winner within 2 units in six-dimensional space. This

ensures that the vector is within a net distance of 2σ from its winner. If at the end of the training, we do not have 95.44% of training data within 2 units of their respective winner neurons, it must be because the vectors that do not fall within 2 units represent network connection behavior that happens in-frequently. We need to amplify the training of these vectors in the map, so that we "show" the in-frequent vectors more to the neurons in the lattice. The *vsom* program has a mechanism to do this with the "weights" qualifier for each vector. If a value of say, "weights 10" is added to the line having an infrequently occuring six-dimensional vector in the training data file, it is treated as if 10 such samples were found in the data file. We amplify in-frequently occurring vectors thus, and repeat the two-phases of training until we satisfy the 95.44% gaussian heuristic specified above. Once a map clears the gaussian heuristic, we conclude the SOM training phase.

There is a false-positive/false-negative trade-off in choosing the gaussian heuristic. Generally any class of traffic has a behavior that is exhibited by the bulk of the samples, and a corner-case behavior that is exhibited by few samples, called outliers. The goal of our heuristic is to let the model capture the bulk behavior of the traffic and not the outlier behavior. Hence, the outliers will be classified as false-positives in such a model. If we were to raise the heuristic and try to capture more outliers into our model, we would get rid of some false-positives, but would be subjecting ourselves to more false-negatives. Note that to include an outlier into our model, we need a neuron within 2 units of standard deviation from it in six-dimensional space. 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 the neuron, it would give rise to a false-negative. As we increase our heuristic value, we would add a lot of such hyper-spheres for the outliers into our model. The SOM model would get more and more general in nature, losing its specificity of modeling only the class of traffic under consideration.

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.

2.1.3.4 SOM Operation

The training phase of the SOM described in the previous sections is done off-line from a network dump file captured for the specific type of network traffic. The real-time operation phase in which the SOM is used to perform intrusion detection however, is performed online. Data Source, Data Processor, and the ANDSOM module, communicate with each other using the *icomm* library. The *icomm* library offers a collection of interface functions that the modules use. The goal of using the *icomm* library is to offer a layer of Application Programming Interface (API) to the modules, so that the exact implementation of the communication mechanism can be made independent of the modules themselves. The current icomm library we use, is implemented with the TCP sockets API.

During the real-time operation phase, data source modules running *tcpurify* communicate the "sanitized" network packets to the Data processor modules. The Data processor modules receive the sanitized network packets and run the *tcptrace* program with INBOUNDS module and produce the 'O', 'U', and 'C' messages which are sent to the ANDSOM module. The ANDSOM module runs the trc2inp and locator sub-modules. The *trc2inp* program, converts the 'O', 'U', and 'C' messages and outputs six-dimensional vectors and the type of network traffic (the TCP or UDP port number). This output is then fed to the *locator* program. The *locator* program is then run for the specific class of network traffic. The six-dimensional vectors are normalized using the mean and standard deviation values used in the normalization process in training of the SOM for that class of network traffic. The normalized vector is now fed into the neurons of the SOM lattice and a Winner is found. The network

connection is then classified as an intrusion if it is more than 2 units from the winner in six-dimensional space.

2.1.4 Intrusion Detection Module

The Intrusion Detection Module is part of the architecture to accomodate one or more modules beside the ANDSOM module to perform intrusion analysis and convey their respective intrusion detection decision. The goal of this module is to corroborate the decisions on intrusions from more than one module and come up with the final decision on intrusion. Currently this module is not implemented, as ANDSOM is the only current intrusion detection module. However, as other intrusion detection mechanisms are added to INBOUNDS in the future, this module shall be implemented.

This module once implemented, can make use of a Data store to record its intrusion decisions. This can be used for future reference and for un-doing the effects of actions taken on false-positives.

2.1.5 Display

The Display module is used to give a real-time display of the connections in and out of the network being monitored by INBOUNDS. The program *networkgraphserver* has been written for this purpose. This program is written in Java and gives a real-time picture of the network, with each host in the network being represented by icons and the connections between hosts indicated by lines between hosts. This module is a work in progress.

2.1.6 Active Response Module

The goal of the Active Response module is to take active response on the connections being perceived as intrusions by INBOUNDS. Some of the response actions against intrusions include: active firewall blocking of that specific network connection, reduction of priority for that connection or that class of network connections at the border router, etc. This module is also a work in progress.

3. Experimental Results

In this chapter, we present some of our results with the operation of the ANDSOM module. We used the ANDSOM module to build SOMs to model normal network traffic patterns of Domain Name System (DNS), Simple Mail Transfer Protocol (SMTP), and HyperText Transfer Protocol (HTTP) traffic. Once the SOM models were built and trained, intrusions, or otherwise anomalous traffic, were generated on the network. These intrusions were obtained from publicly available security vulnerability web-sites, and are potentially the types of intrusions that malicious users in the Internet might use while trying to break into our network. Then, an analysis was made to see if the ANDSOM module could classify these intrusions as anomalous.

We organize this chapter as follows. For each type of network traffic, namely DNS, SMTP, and HTTP, we describe the six-dimensional statistical patterns of normal traffic, the nature of the intrusion, and the training and anomaly detection phases of the ANDSOM module.

3.1 Domain Name System

The Domain Name System (DNS) is a distributed database architecture for the Internet. The DNS concepts and facilities are specified in RFC 1034 [35] and the implementation and specification details are specified in RFC 1035 [36]. The main goal of DNS is to provide domain name to IP address mappings, i.e., given a host name say www.foo.com, DNS is used to translate it to its corresponding host IP address. DNS can also provide inverse mappings to get the domain name given the IP address of a host; provide mail exchange records; and a host of other features specified in RFC 1035 [36].

Table 3.1 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

3.1.1 Normal Traffic

DNS traffic runs on top of both the UDP and TCP transport protocols. Though DNS can perform a variety of functions, the bulk of DNS traffic involves name to address mapping - involving a DNS request and a response. These transactions run typically on top of UDP. DNS uses TCP in special cases, for example, when the UDP response to a request is marked truncated, which happens when the response size is more than 512 bytes; while performing zone transfers, which happen when a secondary name server of a domain gets a copy of the entire domain database from the primary name server.

We built our SOM to model DNS traffic run on top of UDP, as it covers the bulk of DNS traffic. The inbounds module for tcptrace was tuned to have a low connection timeout value of 1 second, so that multiple DNS query-response operations occurring in a short time frame do not get classified as a single connection. 8857 DNS connections were part of the connection data set used in the SOM training process. The mean and standard deviation values of the connection set used are listed in Table 3.1.

We can observe the following traits of the DNS traffic from this table :

- The INTER of DNS traffic is a low value of 0.653 per update interval (set to 1 second for DNS). The average DOC is observed to be 2.033 seconds. Hence the total average INTER in the length of a DNS connection is approximately 1.3 questions (0.653 x 2.033). This value is expected, considering the fact that most of the DNS connections involve just a single query-response.

- The ASOQ value is approximately 29 bytes and the approximate average ASOA value is 112 bytes. This indicates that the responses tend to be relatively bigger than the queries, approximately three times as big. This is expected, because the responses carry the answers to the queries, and hence tend to be bigger. The high standard deviation value found for ASOA indicates that the size of answers is highly variant.

- The L_QAIT value has a mean of -1.14, which means that the QAIT value is in the order of hundredths of a second per second. The L_AQIT value has a mean value -0.01, very close to 0. This is because the AQIT value tends to be close to 1.0 second per second for the bulk of the training data. This again, is because, most of the DNS traffic is single request-response, and there is no request following the received response. Thus the AQIT value is calculated to be 1.0 second per second, as the idle time between an answer and the next question is found to be its maximum value.

3.1.2 Anomalous Traffic

In this section we describe the intrusive network traffic generated in the network.

The Berkeley Internet Name Domain (BIND) server [2] is an implementation of the DNS protocols which includes a DNS server (*named*), a DNS resolver library, and a suite of tools for verifying the proper operation of the DNS server. The BIND DNS server is one of the most widely used implementations of DNS in the Internet.

BIND DNS server *named* versions 8.2.x, have a bug in the processing of transaction signatures (TSIG). Transaction signatures are a secret key authentication mechanism for DNS clients and servers involved in DNS transactions. Transaction signatures are specified in RFC 2535 [20] and RFC 2845 [38].

The BIND exploit is a typical buffer-overflow type attack, in which the execution stack of a running process is corrupted by writing past the bounds of the data allocated in the process stack. A general introduction to how buffer-overflow attacks work can be found in the article "Smashing the Stack for Fun and Profit" [33].

The TSIG bug in the BIND *named* server can be exploited when a malicious client sends a transaction signature resource record, without giving the secret key to be used for authentication. The *named* server program makes wrong assumptions about its buffer sizes when reporting this error condition, which lets the malicious client overflow the buffer space in the server stack. By suitably crafting a DNS query with a dummy TSIG resource record, a malicious client can overflow the return address stored in the stack space, and execute arbitrary code packaged in the packet payload. A detailed description of this vulnerability can be found in [12]. Another security vulnerability, called the Infoleak vulnerability, has also been found in the 8.2.x versions of the BIND named server. This vulnerability causes the BIND *named* server to leak the contents of its stack space. A detailed description of this vulnerability can be found in [13].

The BIND exploit available in public domain [3] exploiting both these vulnerabilities was generated on the network. The exploit first makes use of the infoleak vulnerability to get the return address of the server process function stored in the stack. Then, this return address is used to craft a malicious DNS query with a dummy TSIG resource record. This packet overwrites the stack return address of the server process, so that it points to the code sent as part of the packet payload, and spawns a shell for the attacker upon execution of the code. This shell executes with the user and group permissions of the *named* server process.

3.1.3 Training

As specified in 3.1.1, six dimensional values of 8857 DNS connections were collected from our network, to be the training data set. These six dimensional vectors were then normalized with the Normalizer submodule as described in Section 2.1.3.2. A SOM of dimensions 19x25 was built and initialized to linear values in the range of training data set as specified in Section 2.1.3.3.

The initial phase of training was performed with the following parameters :

- The learning rate factor α was set to a high value of 0.9.
- A gaussian neighborhood function with an initial neighborhood radius of 19 that linearly reduced to 1 at the end of training.
- The number of iterations set to 8857 with the goal of showing to the SOM, each of the training data set vectors once.

The SOM constructed at the end of the initial phase was used as input in the final fine-tuning phase of training. The training parameters were set as follows :

- The learning rate factor α was set to a low value of 0.05.
- A gaussian neighborhood function with an initial neighborhood radius of 5 that linearly reduced to 1 at the end of training.
- The number of iterations set to a high value of 237500, based on the heuristic of having the number of iterations to be 500 times the number of neurons in the SOM (500x19x25=237500).

At the end of the two phases of training, the training data set itself was fed back to the SOM to see the efficiency of SOM model. The SOM model modeled 98.81% of the training data set vectors with a winner neuron within 2σ distance, clearing our 95.44% gaussian heuristic.

Table 3.2 DNS Exploit Vector

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

3.1.4 Anomaly Detection

The INFOLEAK/TSIG exploit appears as two request-response packet sequences in the network, that occur back to back, and get classified as a single UDP connection. The six dimensional vectors of the exploit are shown in Table 3.2.

The *locator* program was fed this DNS exploit vector as input. It first normalizes this vector based on the mean and standard deviation statistics of the DNS training data set shown in Table 3.1. The normalized exploit vector is shown in Table 3.3. This table shows the value of each of the six dimensions of the DNS exploit vector, measured in units of standard deviations from the mean values of the DNS training data set vectors. We can notice from this table 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 just 29.082 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 $10^{-2.375}$, i.e., in the order of milli seconds of a second 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 six-dimensional values of the winner neuron for this normalized DNS exploit vector, and the distance in six dimensional space are shown in Table 3.4. 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 to be classified as an intrusion.

Table 3.3 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 3.4 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

To aid in the visualization of the six-dimensional space, we split the space into two three-dimensional space 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 3.1 and Figure 3.2.

3.2 Simple Mail Transfer Protocol

The Simple Mail Transfer Protocol (SMTP) is the protocol used in the Internet to transfer email traffic amongst hosts. SMTP was originally specified in RFC 821 [37] and has been obsoleted with the updated specification in RFC 2821 [23]. SMTP by itself is a simple protocol, originally designed to transfer email messages with characters from the 7 bit US - ASCII character set. SMTP has been further extended to allow email to contain non-textual messages with the Multipurpose Internet Mail Extensions (MIME). The MIME extensions specified in RFCs 2045-2049 [30, 31, 25, 28, 29], let the email be in characters from other character sets, to attach multi-media messages, and so on. Email traffic is handled by Mail Tranfer Agents (MTA). These agents use the SMTP protocol to communicate amongst each other.

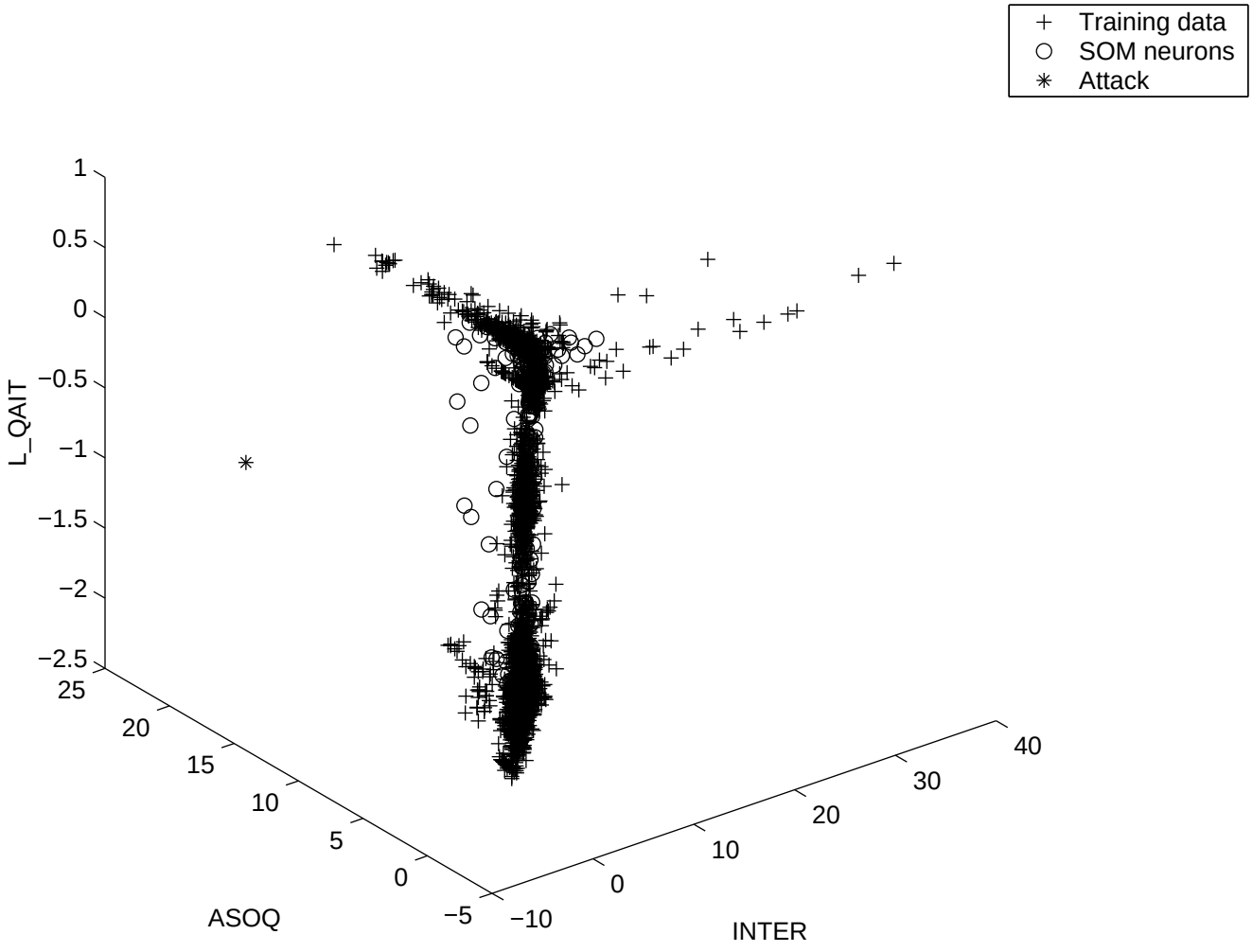

Figure 3.1. DNS Exploit 3D View #1

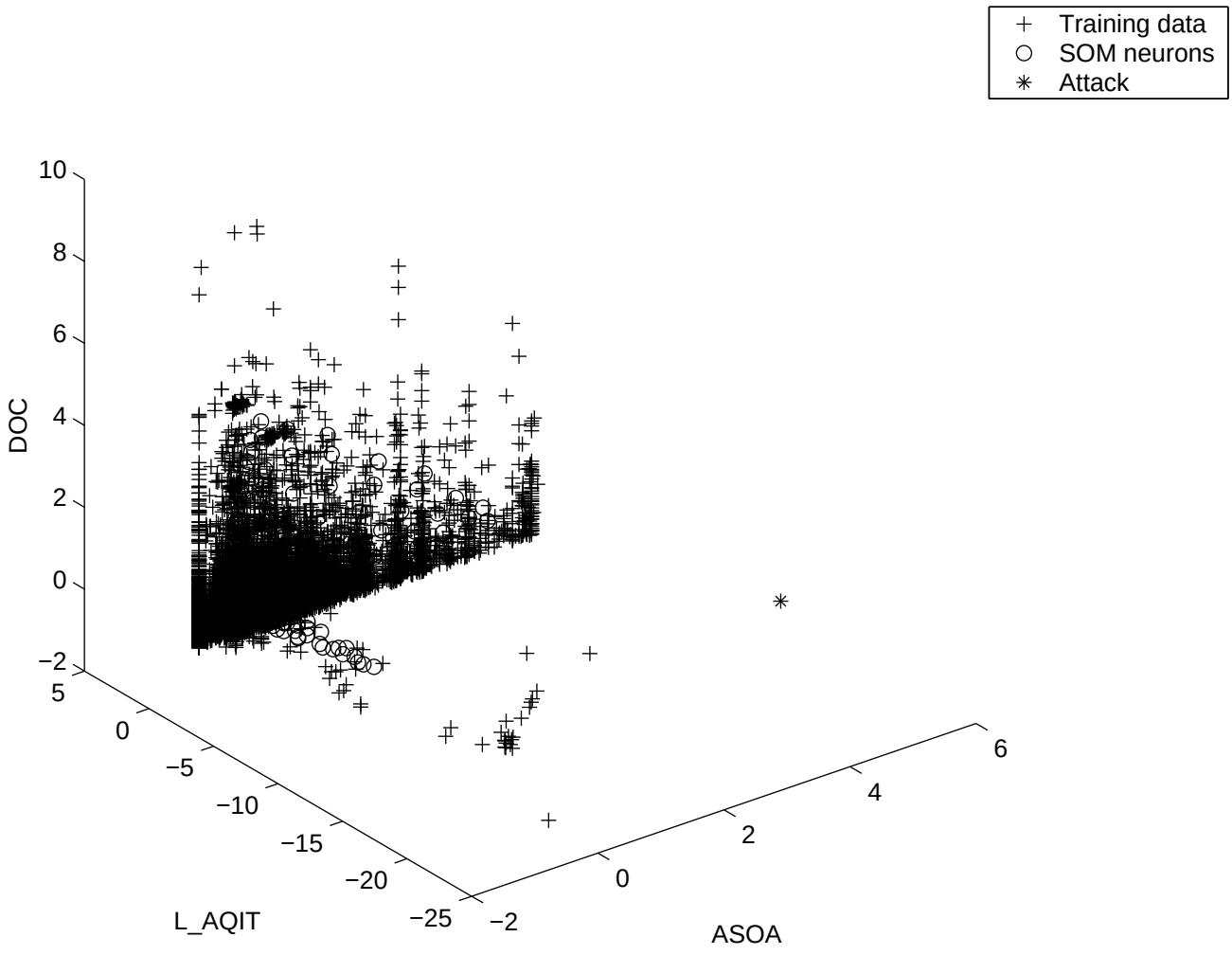

Figure 3.2. DNS Exploit 3D View #2

3.2.1 Normal Traffic

To train and evaluate SOMs for SMTP traffic, we used training and attack data sets provided by the MIT Lincoln Laboratory - DARPA Intrusion Detection Evaluation project [6]. The data sets provided by the MIT Lincoln Laboratory include training data sets (normal traffic free of attacks) and attack data sets (containing labeled attacks). The inbounds module for tcptrace had the update interval value set to its default value of 60 seconds. Unlike the DNS traffic, a connection timeout value for the connection was not critical because SMTP runs on TCP and we could generate the 'C' messages after seeing the FIN packets of the TCP connection.

The training data set included 2305 SMTP connection samples. The timeline graph of a typical SMTP connection in the training data set is shown in Figure 3.3. We can see from the timeline graph that a typical SMTP connection in the training data set involves the following steps:

- SMTP clients initiate the connection with an EHLO command, after receiving the initial "Server Ready" message from the SMTP server. The EHLO command in which the client identifies itself also has the semantics to check if the server understands SMTP extensions. If the SMTP server does not support SMTP extensions, it responds with a "Command Unrecognized" error message.

- If the EHLO command request failed, the SMTP client falls back to the HELO command and identifies itself. The server responds with an "OK" message.

- The client then identifies the sender of the email with the "Mail From" command, which is acknowledged by the server.

- The client then specifies the receiver of the email (which typically is a user in the SMTP server), with a "Rcpt To" command. If the specified email receiver is a valid user, the server responds with a positive acknowledgement.

- Then the client indicates that it is ready to the send the contents of the email,

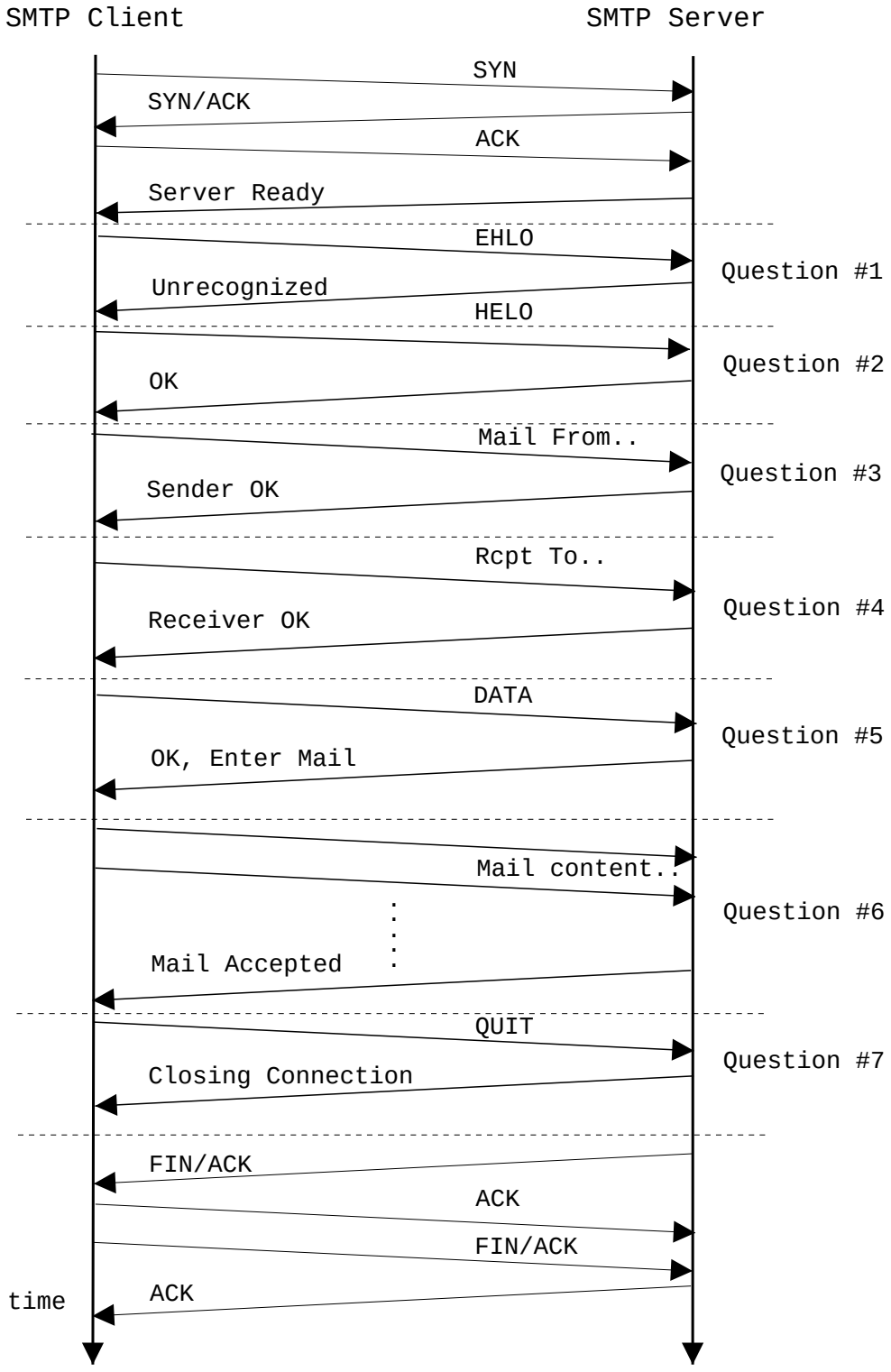

Figure 3.3. SMTP Connection Timeline Graph

Table 3.5 SMTP Training Data Statistics

Dimensions	Mean	Standard Deviation
INTER	6.954	4.687
ASOQ	316.221	579.071
ASOA	36.911	9.971
L_QAIT	-1.525	0.518
L_AQIT	-1.344	0.702
DOC	3.640	75.111

with a "DATA" command. The server acknowledges this and conveys its readiness to receive the email.

- The client then sends the contents of the email to the server. Depending on the size of the email, multiple TCP segments worth of data may be sent. The server responds with a "Mail Accepted" message.

- Finally the client conveys its willingness to close the connection with a "QUIT" message. The server acknowledges the QUIT request, then closes the connection.

The mean and standard deviation values of the six dimensions of the connections in the training data set are presented in Table 3.5.

We can observe the following traits of SMTP traffic from this table :

- The mean INTER value indicates that on an average, SMTP connections involve about 7 questions asked per update interval (60 seconds, in our case). This value is expected considering the fact that a typical connection involves

7 questions: EHLO, HELO, Mail From, Rcpt To, DATA, Mail contents, and QUIT, as specified in the timeline graph shown in Figure 3.3.

- The ASOQ value is approximately 316 bytes while the ASOA is approximately 37 bytes. We see that the average ASOQ is approximately an order of magnitude greater than the average ASOA. This is because the contents of the email sent from the SMTP client to the SMTP server appear as questions. The answers from the SMTP servers to the SMTP clients tend to be short as they typically include just status messages validating the sender/receiver addresses, short instructions for the sender, or any error messages. These messages tend to be relatively small in size and a relatively low standard deviation value for ASOA indicates the less variant nature of the ASOA values.

- The L_QAIT and L_AQIT values tend to have approximately similar values, approximately around -1.5 since the mean QAIT and AQIT values are in the order of hundredths of a second per second. The DOC value is highly variant with a standard deviation value of 75.11 for a mean of 3.64 seconds. This indicates that the duration of SMTP connections tends to be highly variant.

3.2.2 Anomalous Traffic

The anomalous traffic we used for analysis was collected offline from a data set from the MIT Lincoln Laboratory data containing multiple types of attacks. The attack we analyzed, exploits a buffer overflow bug in the sendmail [8] mail transfer agent. The sendmail program is one of the commonly used mail transfer agents in the Internet and implements the SMTP protocol. Sendmail version 8.8.3 had a bug in the processing of MIME headers giving rise to a buffer overflow vulnerability. This vulnerability has been specified in CERT advisory [4]. It has been described in further detail in [14]. The attack [7] exploits this vulnerability by sending a maliciously crafted SMTP request with a large MIME header, to execute arbitrary commands as root by inheriting the privileges of the sendmail program, which typically runs as

root. The attack works by first overflowing the buffer in the server side and modifies the return address to point to the code sent as the MIME header in the packet. The MIME header includes the code to add a root user account to the /etc/passwd file. Upon successful exploit, the attacker can login to the victim system with this new root account.

3.2.3 Training

A training process similar to the one followed in 3.1.3 for training a SOM for DNS traffic was used. Six dimensional values of 2305 SMTP connections were collected to be the training data set. These six dimensional vectors were then normalized with the Normalizer submodule as described in section 2.1.3.2. A SOM of dimensions 13x19 was built and initialized to linear values in the range of training data set as specified in section 2.1.3.3.

The initial phase of training was performed with the following parameters :

- The learning rate factor α was set to a high value of 0.9.
- A gaussian neighborhood function with an initial neighborhood radius of 13 that linearly reduced to 1 at the end of training.
- The number of iterations set to 2305 with the goal of showing to the SOM, each of the training data set vectors once.

The SOM constructed at the end of the initial phase was used as input in the final fine-tuning phase of training. The training parameters were set as follows :

- The learning rate factor α was set to a low value of 0.05.
- A gaussian neighborhood function with an initial neighborhood radius of 5 that linearly reduced to 1 at the end of training.
- The number of iterations set to a high value of 123500, based on the heuristic of having the number of iterations to be 500 times the number of neurons in the SOM (500x13x19=123500).

Table 3.6 SMTP Exploit Vector

INTER	ASOQ	ASOA	L_QAIT	L_AQIT	DOC
0.362	1121.250	128.400	-2.924	-0.008	11.094

At the end of the two phases of training, the training data set itself was fed back to the SOM to see the efficiency of the SOM model. The SOM model modeled 98.18% of the training data set vectors with a winner neuron within 2σ distance, and cleared our 95.44% gaussian heuristic.

3.2.4 Anomaly Detection

The six dimensional vectors of the sendmail buffer overflow attack connection are shown in Table 3.6.

The *locator* program was fed this SMTP exploit vector as input. It first normalizes this vector based on the mean and standard deviation statistics of the SMTP training data set shown in Table 3.5. The normalized exploit vector is shown in Table 3.7. We can notice from this table that the ASOA value is highly anomalous with a distance of 9.176 standard deviations from the mean value of ASOA. When malicious packets are sent to the sendmail server, the server responds with multiple "Command Unrecognized" messages in a response. This response packet is 310 bytes in size, which is much bigger than an average SMTP response packet in size. This drives the net ASOA value to a high value of 128.4 bytes, and is considered highly anomalous since the mean and standard deviation values of ASOA from the training data set are 37 bytes and 9.971 respectively.

It is interesting to note that although the attack packets themselves are observed as questions by the inbounds module, they are not anomalous in size, given that the mean and standard deviations of ASOQ values are 316.221 bytes and 579.071

Table 3.7 SMTP Normalized Exploit Vector

INTER	ASOQ	ASOA	L_QAIT	L_AQIT	DOC
-1.407	1.390	9.176	-2.701	1.903	0.099

Table 3.8 SMTP Winner Neuron

INTER	ASOQ	ASOA	L_QAIT	L_AQIT	DOC	Distance
-0.112	-0.171	0.411	-1.026	1.567	-0.023	9.158

respectively in the training data. However, the size of responses happen to be much more anomalous, and aid in the classification of the attack as an intrusion.

The winner neuron for this normalized SMTP exploit vector, and the distance in six dimensional space is shown in Table 3.8. We can see that the winner neuron was at a distance of 9.158 standard deviations in the six-dimensional space, resulting in the SMTP exploit being successfully classified as an intrusion.

To aid in the visualization of the six-dimensional space, we split the space into two three-dimensional space views. The two three-dimensional views are shown in Figure 3.4 and Figure 3.5.

Note: The training data sample that appears distinctly far from the rest of the samples in Figure 3.5, is a normal SMTP connection that involved sending an email from the SMTP client to the server. However the connection was kept open persistently after the email was sent, for a prolonged period of time, close to 1 hour. This connection is part of the MIT Lincoln Labs training data set containing normal traffic free of attacks. However, it would be classified as an intrusion by the SOM model due to the highly anomalous value of DOC found.

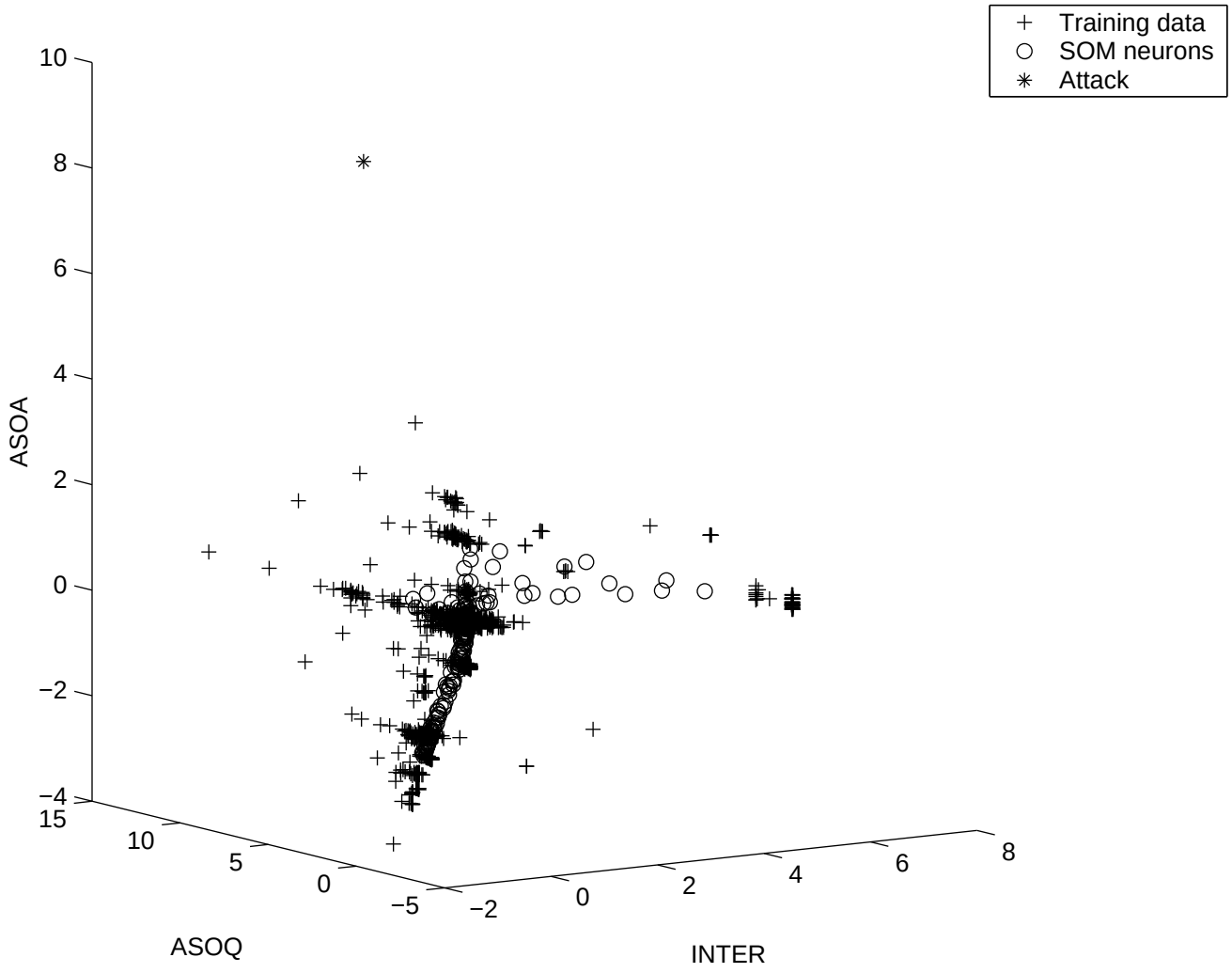

Figure 3.4. Sendmail Buffer Overflow Exploit 3D View #1

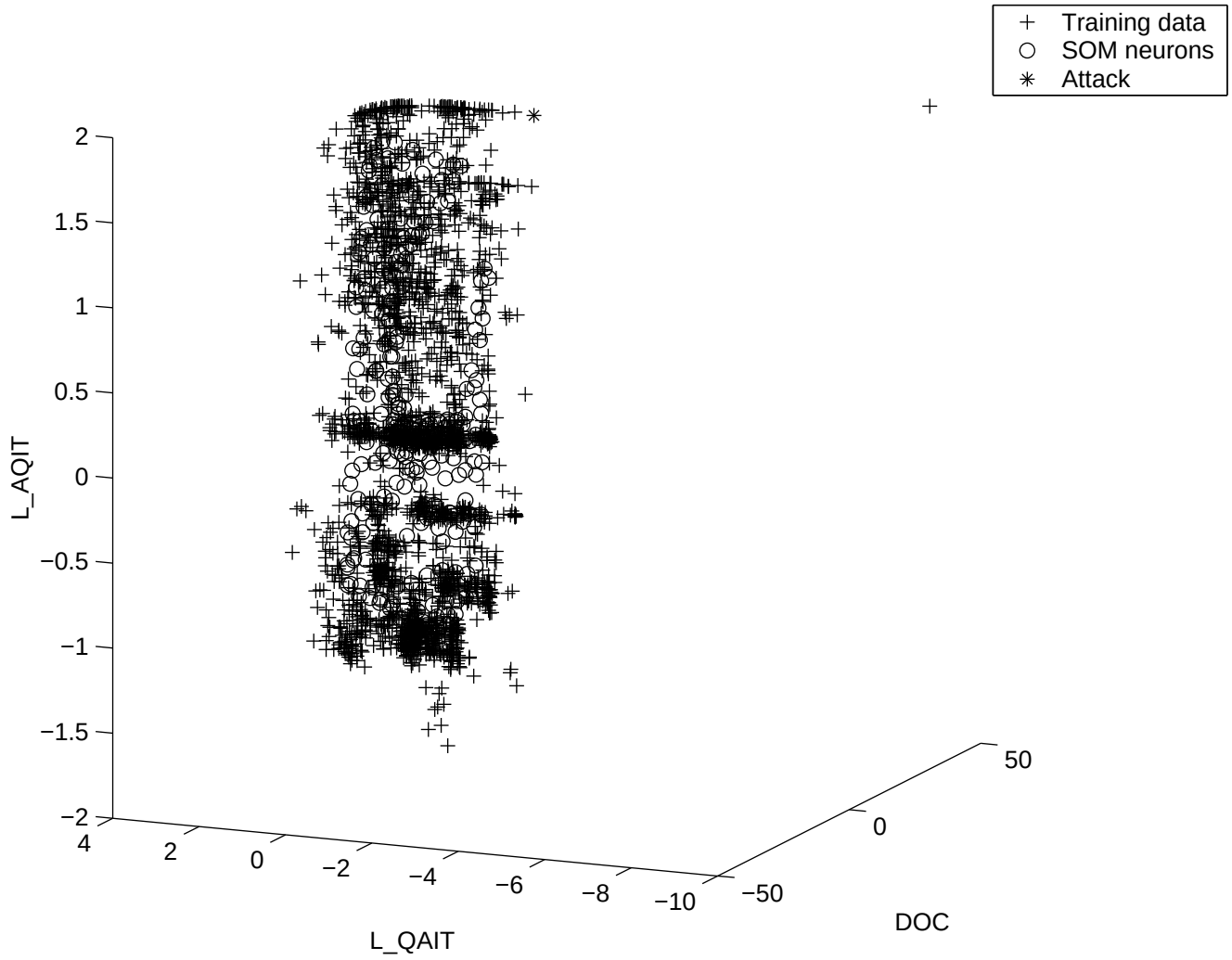

Figure 3.5. Sendmail Buffer Overflow Exploit 3D View #2

3.3 HyperText Transfer Protocol

The HyperText Tranfer Protocol is the application layer protocol used by hosts for communication in the world-wide web. Users run web-browsers to act as their HTTP clients, to communicate with web servers. HTTP allows the transfer of HTML documents, which may include images, multi-media documents and several other types of data, besides textual data. HTTP version 1.0 was specified in RFC 1945 [18]; the current HTTP version 1.1 is specified in RFC 2616 [22] and includes support for persistent HTTP connections, support for web-proxies, etc. Persistent connections let web-clients use a single HTTP connection to get all the objects in a web-page, instead of opening a new TCP connection to fetch each individual object, as was done in HTTP version 1.0.

Although most of the HTTP traffic follows the client-server paradigm with web-browsers (web clients) requesting web-pages, and the web-server serving them, HTTP also lets the web client furnish information to the web-server when the client fills out a so called "form".

3.3.1 Normal Traffic

Training data for HTTP traffic included six-dimensional characteristics collected from 7194 HTTP connections. The training data statistics are presented in Table 3.9.

We can observe the following traits of HTTP traffic from this table :

- The INTER dimension has a mean value of approximately 0.8 questions per second, with a standard deviation value of 0.75.

- The mean ASOQ value is approximately 600 bytes, while the mean ASOA value is approximately 6800 bytes. This is an indication of the fact that most of the data in web-traffic tends to flow from web-servers to web-clients, though some data is also seen flowing in the opposite direction. The high standard deviation values found for ASOQ and ASOA dimensions indicates the extreme variance of the two dimensions.

Table 3.9 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

- The QAIT value seems to be on the order of hundreths of a second per second, while the AQIT value tends to be in the order of ten-thousandths of a second per second. This seems to indicate that relatively longer time is taken for the web-server to answer a question from the web client, than in the opposite direction; this is expected since web-client requests typically travel across the Internet to reach web-servers. The relatively low value of AQIT 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. Web browsers tend to pipeline web requests one after the other so that a request could be sent immediately after the response is received.

- The mean DOC is approximately 9 seconds. However the connection duration tends to be highly variant in nature, which is indicated by the high variance value of approximately 27.

3.3.2 Anomalous Traffic

The anomalous traffic generated in the network was based on the HTTP Tunnel program. HTTP Tunnel [5] is a public domain program that can be used to create

application layer HTTP tunnels between two hosts. Once an HTTP tunnel is setup between two hosts, any type of traffic can be run on top of HTTP. The HTTP tunnel program can potentially be used by attackers inside an organization to break firewall rules.

For example, Let us assume that an organization had a firewall policy to allow web traffic to a host A, while denying all other types of traffic. An insider having access to host A can setup an HTTP tunnel on A and run an HTTP tunnel server on web port 80. Then, an HTTP tunnel client could be started on a host B outside the organization, to communicate with A. If the HTTP tunnel server on A was configured to connect to the local telnet server port on A, a user on host B can use the HTTP tunnel to telnet to A. All application data generated by the user on B, encapsulated in HTTP, appear as HTTP traffic to the firewall, and successfully reaches host A. The tunnel on Host A then decapsulates the data riding on top of HTTP, and sends it to the local telnet port, enabling the user on host B to telnet to A. The replies sent by the telnet server are received by the local HTTP tunnel server on A, encapsulated as HTTP packets, and sent across the tunnel to the HTTP tunnel client on host B.

HTTP tunnel uses the POST and GET methods of HTTP to encapsulate arbitrary application data traffic on top of HTTP. The POST method is meant for web-clients to send data to web-servers, typically while submitting a "form", while the GET method is designed for HTTP clients to retrieve specific web-pages from the web-server. The HTTP tunnel server is started on a port allowed by the organization firewall, typically port 80. It is also configured by the malicious user inside the organization, to establish a local connection with the application port in which the user on host B wants to communiicate. The HTTP tunnel client establishes two TCP connections with the HTTP tunnel server port: One for sending data using the POST method (POST connection), and another for receiving data using the GET method (GET connection).

Thereafter, data is sent on the POST connection and replies are fetched from

the tunnel server via the GET connection. The tunnel client times out the POST connection after it has been open for a time threshold, which defaults to 300 seconds. Once the POST connection had been open for the time threshold, the tunnel client closes and re-establishes the POST connection with the HTTP tunnel server. The HTTP tunnel server closes the GET connections, after content-length bytes have been sent on the connection. The content-length value is the value chosen and replied by the HTTP tunnel server, in response to the initial GET request, defaulting to 10K bytes. HTTP tunnel clients reopen their GET connections when they find that their connection had been closed by the server.

To generate anomalous data using the HTTP tunnel, an HTTP tunnel server was setup on our lab machine on port 80 and it was configured to connect to the local telnet server running on port 23. An HTTP tunnel client was started on a host across the Internet, to connect to the telnet server running on the lab machine on the HTTP tunnel. Network data was collected when the remote machine "telnet"-ed to the lab machine via the tunnel on port 80. This anomalous port 80 traffic was analyzed with a SOM built for genuine port 80 HTTP traffic.

3.3.3 Training

A training dataset of 7194 HTTP connections collected from our network was used in the SOM training phase. These six dimensional vectors were normalized with the Normalizer submodule; a SOM of dimensions 16x27 was built and initialized to linear values in the range of values of the training data set.

The initial phase of training was performed with the following parameters :

- The learning rate factor α was set to a high value of 0.9.
- A gaussian neighborhood function with an initial neighborhood radius of 16 that linearly reduced to 1 at the end of training.
- The number of iterations set to 7194 with the goal of showing to the SOM each of the training data set vectors once.

Table 3.10 HTTP Tunnel Traffic

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

The training parameters were set to the following values in the final fine-tuning phase of training. The training parameters were set as follows :

- The learning rate factor α was set to a low value of 0.05.
- A gaussian neighborhood function with an initial neighborhood radius of 5 that linearly reduced to 1 at the end of training.
- The number of iterations was set to a high value of 216000, based on the heuristic of having the number of iterations to be 500 times the number of neurons in the SOM (500x16x27=216000).

At the end of the two phases of training, the training data set itself was fed back to the SOM to see the efficiency of SOM model. The SOM model modeled 98.83% of the training data set vectors with a winner neuron within 2σ distance, clearing our 95.44% gaussian heuristic.

3.3.4 Anomaly Detection

The telnet connection running on the HTTP tunnel lasted for approximately 10 minutes, during which 13 connections (3 POST connections and 10 GET connections) were opened.We present in this section, the GET and POST connections that turned out to be highly anomalous amongst the 13 conections. The six dimensional vectors of those GET and POST connections are shown in Table 3.10 in lines 1 and 2 respectively.

The *locator* program was fed the HTTP Tunnel Traffic vectors of GET and POST connections as input. It first normalizes this vector based on the mean and standard deviation statistics of the HTTP training data set shown in Table 3.9. The normalized HTTP Tunnel traffic vector is shown in Table 3.11. We can observe from tables 3.10 and 3.11 that the GET connection (first line in the table) exhibits highly anomalous values of L_QAIT and the DOC values.

In the GET connection, the HTTP tunnel client makes a GET request for the index.html webpage, and requests that the connection be persistent. This initial request is the only data seen flowing from client to server. Thereafter, all the replies flow from the tunnel server to the client.

The inbounds module for *tcptrace* recognizes bursts of data as questions and answers, i.e. a sequence of data packets flowing from the tunnel client to the tunnel server and vice-versa is counted as a single burst of data, and hence, as a single question or answer until a packet of non-zero data length is received in the opposite direction, or a TCP FIN packet is seen to denote the end of a data burst. Pure TCP ACK packets flowing in the opposite direction do not terminate the data burst as perceived by the inbounds module for *tcptrace*. Hence the entire GET connection is found to have a single question and an answer. All replies from the web-server form a single answer, because, once the GET request is made and data starts flowing from the tunnel server, there are only pure TCP ACK packets seen from the tunnel client. 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 L_QAIT value of -1.383. The AQIT value is measured only finally, when the TCP FIN packet is seen to denote the end of the connection. This value is also found to be low, because, the FIN packet is generated at the end of the connection by

the tunnel server itself, right after sending the last byte of data, causing the perceived AQIT to be very low, in the range of microseconds. However, it is not found to be anomalous since the L_AQIT value exhibits a very high standard deviation of 3.324. The duration of the connection (DOC) happens to be 247 seconds approximately, and is highly anomalous with a distance of 8.74 standard deviations, considering that the mean value of DOC in the training data was approximately 9 seconds, with a standard deviation of approximately 27.

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 3.10 because all the 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. Log 10 of 0 is negative infinity, a value which is reported by the trc2inp module as the low value of -10.000. 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 summarise, 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 locator module found the same winner neuron for the GET and POST connection traffic, which is presented along with the six-dimensional distance of GET and POST connections from the winner in Table 3.12.

Table 3.11 HTTP Normalized Tunnel Traffic

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

Table 3.12 HTTP Winner Neuron

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

We can see that the winner neuron was at a distance of 4.835 and 6.743 standard deviations in the six-dimensional space for the GET and POST connections, resulting in them being successfully classified as intrusions. The two three-dimensional views of six-dimensional space for HTTP traffic, are shown in figures Figure 3.6 and Figure 3.7.

3.4 Running Time Analysis

To provide an estimate of the running time of the modules involved in real-time analysis of connections, an off-line analysis was performed by collecting network dump files of varying sizes. The results of this analysis are presented in detail in Appendix A. In this analysis, network traffic collected in tcpdump-style dumpfiles of varying sizes were collected offline and were given as input to the Data Processor module which sent its output to the ANDSOM module. The ANDSOM module then processed the network connections to find intrusions based on the SOM algorithm.

The results of this off-line analysis indicate that the ANDSOM module together

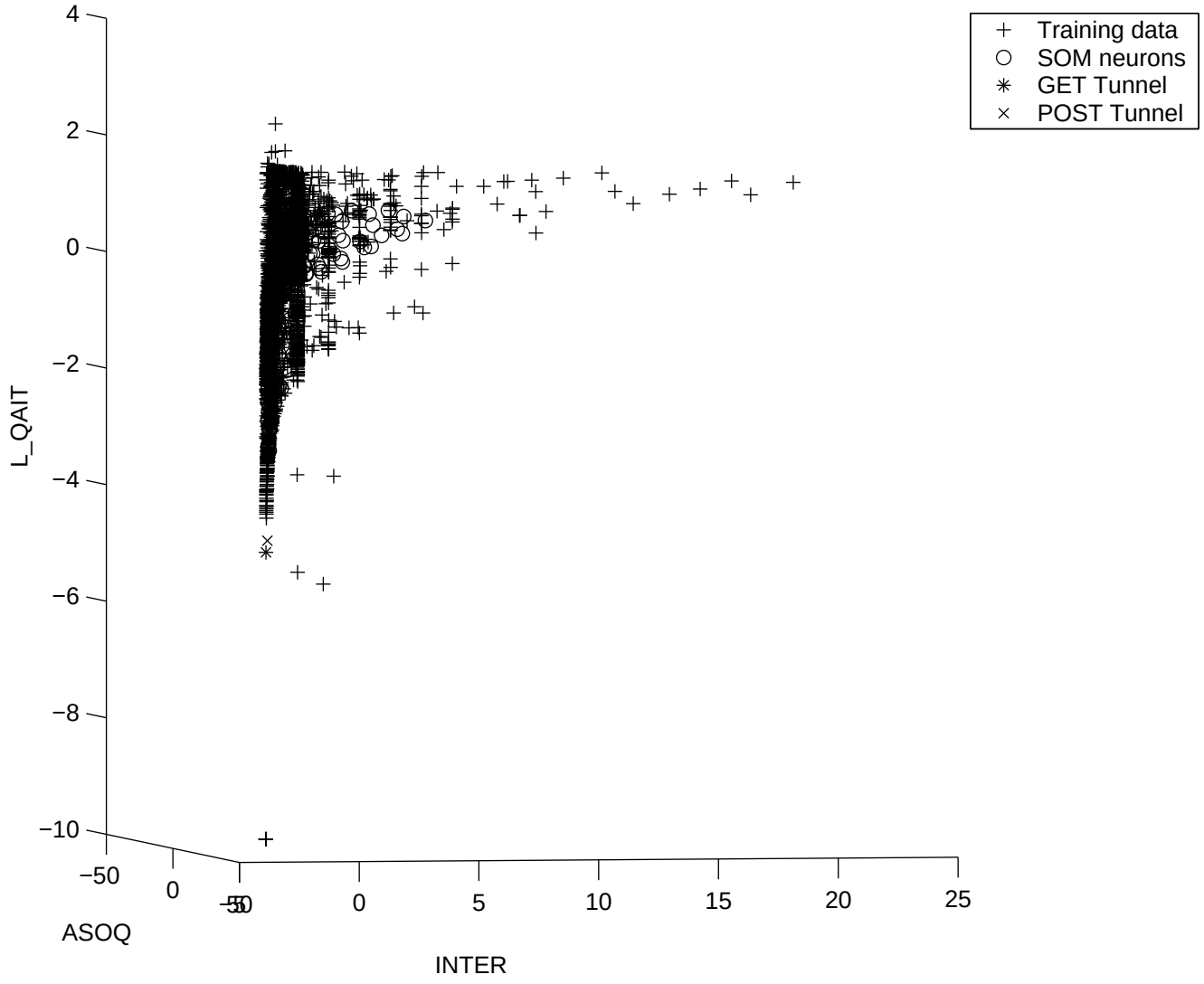

Figure 3.6. HTTP Tunnel Traffic: 3D View #1

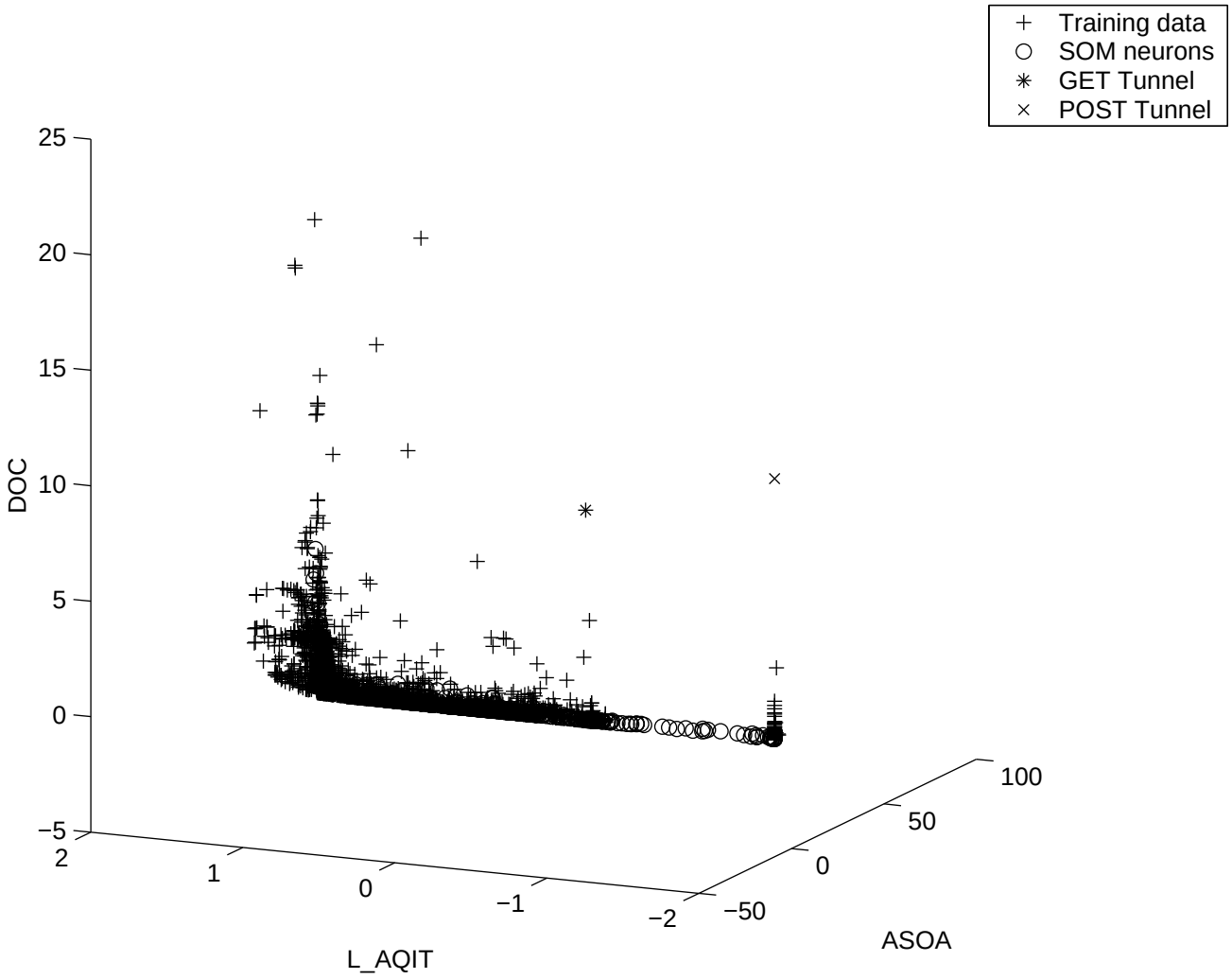

Figure 3.7. HTTP Tunnel Traffic: 3D View #2

with the Data Processor module could process network traffic rates ranging approximately from 70 Mbps to 300 Mbps. The processing rate was found to be dependent on the size of the dumpfile, the number of connections found, and the average data rate found on the network for the duration of the data capture.

4. Conclusion

In this chapter, we present a summary of the design and operation of the ANDSOM module, discuss its advantages and disadvantages, and follow it up with some recommendations for future work.

4.1 Summary

A new approach to intrusion detection was introduced to the INBOUNDS intrusion detection system with the addition of the ANDSOM module. The ANDSOM module uses the Self-Organizing Map algorithm to build a two-dimensional lattice of neurons called a Self-Organizing Map. The goal of the SOM algorithm is to capture the essential characteristics of data from a multi-dimensional input signal space, into geometrical relationships of neurons in a two-dimensonal lattice.

If the input signal space can be characterized by k parameters, individual data points in the input signal space are specified as k-dimensional vectors. Neurons in the two-dimensional lattice are also chosen to be k-dimensional vectors. A training data set of k-dimensional vectors is then selected from the input signal space covering the spectrum of operational behavior. The neurons in the lattice are then initialized to values covering the range of values in the training data set.

During the training phase, training data is "shown" to all the neurons in the lattice, and a winner neuron closest to the training data in k-dimensional space is selected. The winner neuron and its neighbor neurons are then moved slightly towards the training data point in k-dimensional space. This training is done in two phases: an initial phase with a high learning factor that lasts for a lesser number of iterations, and a fine-tuning phase with a lower learning factor lasting for a larger number of

iterations. At the end of training, the SOM lattice captures the essential characteristic patterns of the input signal space.

During the operational phase, data points are fed to the neurons of the trained lattice, and the distance between the winner and the input data point in k-dimensional space is measured. An intrusion alert is raised if this distance is found to be more than a threshold distance.

The ANDSOM module used six dimensions to characterize network traffic: Interactivity (INTER), Average Size of Questions (ASOQ), Average Size of Answers (ASOA), Log base 10 of Question-Answer Idle Time (L_QAIT), Log base 10 of Answer-Question Idle Time (L_AQIT), and the Duration of Connection activity (DOC).

We built Self-Organizing Maps for different classes of traffic, including DNS, SMTP, and HTTP traffic for the six dimensions. A buffer overflow attack exploiting bugs in the BIND daemon version 8.2.x (DNS) was classified as an intrusion by the SOM we built for DNS traffic, because of the anomalous size of ASOQ found during the attack. The SOM we built for the SMTP traffic, classified the buffer overflow attack in the Sendmail program version 8.8.3 (SMTP) as an intrusion based on the anomalous size of ASOA found due to the relatively larger size of responses generated by the Sendmail server under attack. Also, the SOM we built to model HTTP traffic classified the HTTP traffic generated when a user encapsulates arbitrary data on top of an application layer HTTP tunnel as an intrusion. This was done based on the fact that such a tunnel connection lasted much longer than the average HTTP connection and since the L_QAIT value turned out to be relatively low. In all three intrusions specified, a winner was found in the lattice of neurons of the respective SOM, which was more than 2 units in the six-dimensional space. This fact was used to classify them as an intrusion, since the threshold was set to 2 units.

4.2 Comparison With Statistical Approach

We compared the performance of the SOM-based approach with the statistical approach based on the Abnormality Factor method (described in Section 1.2.2.1), used previously in INBOUNDS. Statistical models were built to model DNS, SMTP, and HTTP traffic using the same training data set used for building their respective SOM models. For each class of traffic, the mean and standard deviation values of the five dimensions : INTER, ASOQ, ASOA, QAIT, AQIT were collected from the training data set. The Abnormality Factor method uses two parameters Standardization Factor (SF) and a Threshold (THRESH) to evaluate if the "distance" of the sample from the model was high enough to raise an intrusion alert (Section 1.2.2.1). Since the way this distance measure is evaluated is different from the euclidean distance measure used in the SOM model, we calculated the distance measure as follows to be able to compare with the performance of the SOM model. For each of the five dimensions of a training data set sample, the difference was calculated from its mean in units of standard deviation. For e.g., if dim1 was the value of dimension 1, and m1, sd1 were its mean and standard deviation, the difference $diff1$ was calculated as :

$diff1 = \frac{(dim1 - m1)}{sd1}$

The Root Mean Square value was found from the difference values found for all the dimensions as:

RMS= $\sqrt{diff1^2 + diff2^2 + \ldots + diff5^2}$. If the RMS value found was greater than 2 units, it was classified as an intrusion. Note that the SOM model also uses a distance of 2 units of standard deviation from the winner neuron as its threshold to make an intrusion alert.

We found that for both the DNS and SMTP traffic, the statistical model successfully detected the intrusions. However, when the training data itself was fed back to the model to validate it, the percentage of false-positives generated was significantly higher than the percentage generated by their respective SOM models.

For DNS, only 78.54% of the 8857 training data samples were within a distance of 2 units from the model, giving rise to a false-positive percentage of 21.46%. The corresponding false-positive percentage of training data in the SOM model was 1.19%. Similarly, for the SMTP traffic the statistical model generated 30.85% false-positives for the 2305 training data samples, while the SOM model generated 1.82%.

The HTTP statistical model could not detect anomalous traffic generated using the HTTP tunnel (Section 3.3.2), since it did not have the sixth dimension DOC (Duration of Connection) in its design, on which the anomaly was observed. Further, the HTTP statistical model generated 11.9% false-positives for the HTTP training data set containing 7194 samples, while the SOM model generated 1.17%.

To summarize, we find that though the statistical model is capable of detecting the intrusions detected by the corresponding SOM model, the false-positive rate seems to be much higher upon validation. This is because, having a single sample point to model a class of traffic is a much weaker way to model a traffic class, when compared to having a lattice of trained neurons in a SOM model characterizing it.

4.3 Advantages and Disadvantages

The ANDSOM module can capture and successfully classify an intrusion if its six-dimensional characteristics are different from the normal characteristics used during the training phase. In contrast to typical signature-based intrusion detection systems that need to store the signature of an attack to detect it, the ANDSOM module can detect never-before-seen attacks if the traffic characteristics are different from normal operational characteristics.

However, the ANDSOM model has the limitation that the attacks that resemble normal operational behavior may not be detected, giving rise to false negatives. Further, the SOM model cannot completely capture the characteristics of data points in the six-dimensional space. The training phase is considered complete when 95.44% of the training data set samples fall within 2 units in six-dimensional space from their

winner neurons in the SOM. This implies that the remaining 4.56% of training data set themselves will be classified as intrusions that would turn out be false-positives. This is a limitation of the SOM algorithm; it can capture the behavior exhibited by the bulk of a traffic class, but corner-case behavior occuring infrequently may be classified as intrusions, giving rise to false-positives.

4.4 Future Work

The ANDSOM module uses the basic SOM algorithm, using a gaussian neighborhood function and a hexagonal map topology. As a continuation of our work, it could be interesting to study the effects of modification to the SOM algorithm, including trying other neighborhood functions and different map topologies. Further, we have assumed that the distribution of data in the training data set to be gaussian in nature, to arrive at the 2 σ – 95.44% heuristic. Though this has been quite successful, it would be interesting to construct and validate maps assuming different distributions for data in the training data set, and with various values of threshold with the gaussian heuristic, assuming the distribution to be gaussian.

The run-time analysis of various modules in the INBOUNDS system discussed in Appendix A indicates that the bulk of the time involved is consumed by the *tcptrace* program with the INBOUNDS module. Future work could involve improving the processing time efficiency of this module.

Other variants of SOM, like the Adaptive Subspace Self-Organizing Map (ASSOM) may also be tried to build maps, to capture multi-dimensional data characteristics, to see if they yield better results in intrusion detection.

The ANDSOM module analyzes the characteristics of each individual connection and makes a decision on whether the connection seems normal or anomalous. The ANDSOM module can classify intrusions only if each individual connection looks anomalous; it cannot detect a Denial of Service(DOS) attack in which each individual connection would seem normal, but the attack is due to the presence of an anomalous

number of them at the same time in the network. The ANDSOM module may be extended to study the behavior of the set of all connections of a particular class of traffic in the network, for e.g., the set of all e-mail connections in the network as a whole. New dimensions to capture the total number of connections of a particular type in the network at any instant, and other dimensions to capture the net characteristics of a class of traffic may be added. It would then be interesting to see if the ANDSOM module modified thus can classify Denial of Service attack conditions as intrusions.

BIBLIOGRAPHY

[1] URL: http://www.cis.hut.fi/ jhollmen/dippa/node29.html.

[2] BIND, Internet Software Consortium. URL: http://www.isc.org/products/BIND.

[3] BIND named 8.2.x remote exploit. URL: http://downloads.securityfocus.com/vulnerabilities/exploits/tsig.c.

[4] CERT Advisory CA-1997-05 MIME Conversion Buffer Overflow in Sendmail Versions 8.8.3 and 8.8.4. URL: http://www.cert.org/advisories/CA-1997-05.html.

[5] HTTP Tunnel. URL: http://www.nocrew.org/software/httptunnel.html.

[6] MIT Lincoln Laboratories - DARPA Intrusion Detection Evaluation. URL: http://www.ll.mit.edu/IST/ideval/index.html.

[7] Sendmail 8.8.3 buffer overflow attack. URL: http://www.ll.mit.edu/IST/ideval/docs/1999/attackDB.htmlsendmail.

[8] Sendmail Consortium. URL: http://www.sendmail.org.

[9] SNORT The Open Source Intrusion Detection System. URL: http://www.snort.org.

[10] SOM_PAK. URL: http://www.cis.hut.fi/research/som_lvq_pak.shtml.

[11] SOMTOOLBOX. URL: http://www.cis.hut.fi/projects/somtoolbox.

[12] VU#196945 ISC BIND 8 Buffer Overflow in TSIG handling code. URL: http://www.kb.cert.org/vuls/id/196945.

[13] VU#325431 ISC BIND 8 servers may disclose environment variables. URL: http://www.kb.cert.org/vuls/id/325431.

[14] MIME Conversion Buffer Overflow in Sendmail Versions 8.8.3 and 8.8.4, January 1997. URL: http://www.cert.org/advisories/CA-1997-05.html.

[15] Internet Domain Survey, Internet Software Consortium, January 2002. URL: http://www.isc.org.

[16] Baldwin, R., and Rivest, R. The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms, October 1996. RFC 2040.

[17] Balupari, R. Real-time Network-based Anomaly Intrusion Detection. Master's thesis, Ohio University, 2002.

[18] Berners-Lee, T., Fielding, R., and Frystyk, H. Hypertext Transfer Protocol – HTTP/1.0, May 1996. RFC 1945.

[19] B.Mukherjee, T.Heberlein, and K.Levitt. Network Intrusion Detection. *IEEE Network* (May/June 1994).

[20] D.Eastlake. Domain Name System Security Extensions, March 1999. RFC 2535.

[21] E.Alhoniemi, J.Hollmen, O.Simula, and J.Vesanto. *Integrated Computer Aided Engineering 6, 3* (1999).

[22] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext Transfer Protocol – HTTP/1.1, June 1999. RFC 2616.

[23] J.Klensin. Simple Mail Transfer Protocol, April 2001. RFC 2821.

[24] K.Goser, U.Hilleringmann, U.Rueckert, and K.Schumacher. *IEEE Micro 9, 28* (1989).

[25] K.Moore. MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text, November 1996. RFC 2047.

[26] Kohonen, T. In *In Proc. 6ICPR, Int. Conf. on Pattern Recognition (IEEE Computer Soc. Press, Washington, D.C. 1982*, p. 114.

[27] Kohonen, T. *Self-Organizing Maps*, 3rd ed. Springer, 2001.

[28] N.Freed, J.Klensin, and J.Postel. Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures, November 1996. RFC 2048.

[29] N.Freed, and N.Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples, November 1996. RFC 2049.

[30] N.Freed, and N.Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, November 1996. RFC 2045.

[31] N.Freed, and N.Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, November 1996. RFC 2046.

[32] Oja, E. Neural networks, principal components, and subspaces. *International Journal of Neural Systems 1*, 1 (1989), 61–68.

[33] One, A. Smashing The Stack For Fun And Profit . URL: http://www.insecure.org/stf/smashstack.txt.

[34] Ostermann, S. tcptrace. URL: http://www.tcptrace.org.

[35] P.Mockapetris. Domain Names - Concepts and Facilities, November 1987. RFC 1034.

[36] P.Mockapetris. Domain Names - Implementation and Specification, November 1987. RFC 1035.

[37] Postel, J. Simple Mail Transfer Protocol, August 1982. RFC 821.

[38] P.Vixie, O.Gudmundsson, 3rd, D., and B.Wellington. Secret Key Transaction Authentication for DNS (TSIG), May 2000. RFC 2845.

[39] R.Heady, G.Luger, A.Maccabe, and M.Servilla. The Architecture of a Network Level Intrusion Detection System. Tech. rep., University of New Mexico, 1990.

A. Running Time Analysis

We performed an off-line analysis of the modules in the INBOUNDS system to get an estimate of their real-time performance. The results of the analysis are presented here.

Network traffic was captured for varying durations (15 min, 30 min, 45 min, 1 hour, 2 hours, and 3 hours) and stored as tcpdump-style dumpfiles. During the analysis, traffic found in each of the dumpfiles was analyzed by the real-time inbounds module for *tcptrace* which gave the 'O', 'U', and 'C' records as output. This output was then fed to the *trc2inp* program to get the six dimensional vectors of complete connections as output. This output was then fed to the *locator* program to perform analysis of the traffic on the SOM built for http traffic.

The choice of using the SOM built to model http traffic was arbitrary; since the goal was to get an estimate of the running time of modules, a SOM built for any other traffic could have been used as well, and would have given similar results. The communication between the modules was made through pipes, with each module reading from standard input (stdin) and writing to standard output (stdout). Further, as the goal was to measure just the run-time performance, all types of traffic found in the dumpfile (including non-http traffic) was also processed with the SOM model built for http traffic.

In the Table A.1, performance statistics of dumpfiles of varying durations of capture are listed.

- Duration: Duration in which the dumpfile was captured from the network.
- Bytes: Total number of bytes seen on the wire for the Duration.

Table A.1 Offline Data Analysis

Duration	Bytes	Packets	Data Rate (Mbps)	Conns.	Proc. Time (sec)	Proc. Rate (Mbps)
6 Nov 2001 12:00-12:15 (15 min)	88,425,003	238,276	0.79	1,033	4.05	174.66
6 Nov 2001 13:00-13:30 (30 min)	255,205,558	488,712	1.13	1,853	6.86	297.62
6 Nov 2001 14:00-14:45 (45 min)	400,026,477	886,399	1.19	1,865	13.32	240.26
24 Oct 2001 18:20-19:20 (1 hour)	1,096,577,208	2,536,807	2.44	10,704	57.11	153.61
30 Oct 2001 10:00-12:00 (2 hours)	1,954,570,814	5,322,218	2.17	87,817	214.6	72.86
30 Oct 2001 10:00-13:00 (3 hours)	2,810,266,341	7,686,773	2.08	124,935	295.6	76.06

- Packets: Total number of packets seen on the wire for the Duration.
- Data Rate: The average data rate observed during the duration of data capture (Bytes / Duration) in Mbps.
- Conns: Total number of connections found in the captured traffic.
- Proc. Time: The net processing time taken to process the dumpfile and perform intrusion detection analysis using the SOM algorithm.
- Proc. Rate: The processing rate (Bytes / Proc. Time) in Mbps.

We can observe from this table that the processing time increases with the size of the dumpfiles, and the number of packets and connections found in them. The processing rate varies from a minimum value of approximately 73 Mbps to a maximum value close to 298 Mbps.

The running time of each of the modules: *tcptrace*, *trc2inp*, and *locator* was analyzed separately for all the dumpfiles, and the results are shown in Table A.2. The packet-processing rate and the connection-processing rate of each of the modules are also listed in their respective columns. We find from this table that the bulk of the processing time is consumed by the real-time inbounds module with *tcptrace*. We also find that the running time of the *locator* module is roughly proportional to the number of connections. This is expected since the *locator* module feeds the six-dimensional vectors of a connection to the SOM and locates the winner. Thus its running time is proportional to the number of connections processed.

We observe the average packet processing rate of the real-time module with *tcptrace* to be 54,392 pkts/sec. The average rate at which six-dimensional vectors of connections are output by the *trc2inp* module is found to be 4,688 conns/sec. The *locator* module then processes these connections at an average rate of 9,266 conns/sec.

Table A.2 Processing Time Analysis

Duration	Pkts	Conns.	tcptrace (sec) (pkts/sec) (conns/sec)	trc2inp (sec) (pkts/sec) (conns/sec)	locator (sec) (pkts/sec) (conns/sec)
6 Nov 2001			3.4	0.2	0.13
12:00-12:15	238,276	1,033	70,081	1,191,380	1,832,892
(15 min)			303.8	5,165	7,946
6 Nov 2001			6.6	0.34	0.23
13:00-13:30	488,712	1,853	74,047	1,437,388	2,124,835
(30 min)			280.8	5,450	8057
6 Nov 2001			12.77	0.44	0.27
14:00-14:45	886,399	1,865	69,413	2,014,543	3,282,959
(45 min)			146	4,239	6,907
24 Oct 2001			46.64	8.63	1.02
18:20-19:20	2,536,807	10,704	54,391	293,952	2,487,065
(1 hour)			229.5	1,240	10,494
30 Oct 2001			188.3	13.88	7.93
10:00-12:00	5,322,218	87,817	28,265	383,445	671,150
(2 hours)			466.37	6,327	11,074
30 Oct 2001			254.9	21.9	11.24
10:00-13:00	7,686,773	124,935	30,156	350,994	683,877
(3 hours)			490.1	5,705	11,115