TIME-BASED APPROACH TO INTRUSION DETECTION USING MULTIPLE 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

Ankush Sawant

March 2005

This thesis entitled

TIME-BASED APPROACH TO INTRUSION DETECTION USING MULTIPLE SELF-ORGANIZING MAPS

by

ANKUSH SAWANT

has been approved for

the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology by

Carl T. Bruggeman

Assistant Professor of Computer Science

Dennis Irwin

Dean, Russ College of Engineering and Technology

SAWANT, ANKUSH. M.S. March 2005.
Electrical Engineering and Computer Science

Time-based Approach to Intrusion Detection using Multiple Self-Organizing Maps (82pp.)

Director of Thesis: Carl T. Bruggeman

Anomaly-based intrusion detection systems identify intrusions by monitoring network traffic for abnormal behavior. Integrated Network-Based Ohio University Network Detective Service (INBOUNDS) is an anomaly-based intrusion detection system being developed at Ohio University. The Multiple Self-organizing map based Intrusion Detection System (MSIDS) module for INBOUNDS analyzes the time-based behavior of normal network connections for anomalies, using the Self-Organizing Map (SOM) algorithm. The MSIDS module builds profiles of normal network behavior by characterizing the network traffic with four parameters. A SOM, developed for each time interval, captures the characteristic network behavior for that time interval using the four parameters. This approach achieves better characterization of normal network behavior, leading to better intrusion detection capability. During real-time operation, the four-dimensional vectors, representing the attack connection for the time intervals, are fed into respective trained SOMs. For each input vector in the four-dimensional space, a "winner" neuron is determined. If the distance between the input vector and the winner neuron for any SOM is greater than a certain threshold value, the MSIDS module classifies the network connection as an intrusion. Moreover, detecting the attack in early stages of the connection leads to near real-time response to intrusions.

Approved:

Carl T. Bruggeman

Assistant Professor of Computer Science

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Bruggeman, for his kind guidance during the development of this study. I remain greatly indebted to him for his indispensable advice, constructive criticism and constant words of motivation.

I express sincere gratitude towards Dr. Ostermann for granting me the opportunity to work on the INBOUNDS project. I would also like to thank the members of INBOUNDS and IRG research groups for their valuable help and feedback.

I am forever indebted to my parents, my brother Aniket and my relatives for their continual love and support. Additionally, I am thankful to my friends at Ohio University and back home, who have contributed either directly or indirectly towards this thesis. And last but not the least, I am grateful to my friend Veena for her unending support and encouragement throughout the development of my thesis.

I dedicate this thesis to my parents.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

Since its inception in 1969, the Internet has dramatically transformed the traditional method of information access and exchange. The Internet has overcome geographical limitations and provided a medium for collaboration and interaction between individuals across the globe. Anyone with a computer and a network connection can gain access to this world-wide mechanism of information dissemination. As of Jan 2004, the Internet connected an estimated 233 million hosts [21].

However, significant risks are associated with this convenience and ease of access to information. Computer systems connected to the Internet are more susceptible to security attacks than isolated ones. Sensitive information such as financial transactions, employees records, and passwords, become potentially accessible to millions of users when an enterprise's networked system is part of the Internet. With today's booming e-commerce economy, it is even more important to protect the loss or modification of critical business data, disruption of services (availability), and compromise of sensitive information (confidentiality and integrity).

1.1 Network Security

To preserve the availability, confidentiality and integrity of computer network resources connected to the Internet, network and security engineers take measures for prevention and detection of attacks as they occur. Following are some of the common security threats experienced by existing networked systems:

- Unauthorized access, modification, or destruction of networked resources
- Misuse of authorized access to networked resources

- Malicious software programs (viruses/worms/Trojans)
- Misconfigured or poorly designed information systems

To counter such threats, security personnel typically employ techniques such as firewalls, user authentication, access control, and data encryption. Firewalls aim to regulate and control the flow of information into and out of a network, and are usually deployed at an access point between the private network and the Internet. Authentication mechanisms like individual passwords, a master password, and biometrics protect an organization's data against illegitimate access. Access control and data encryption techniques further enhance security by protecting against accidental or unauthorized access.

1.2 Intrusion Detection Systems

Commonly deployed security measures such as firewalls, user authentication and encryption algorithms are effective mechanisms to protect and prevent unauthorized access to systems. However, they lack the capability to examine the network traffic where majority of the attacks occur. To constantly monitor the network traffic for attacks, an organization's network security kit must include Intrusion Detection Systems (IDSs).

Intrusion detection can be defined as "the process of monitoring the events occurring in a computer system or network and analyzing them for signs of *intrusions*, defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network" [55]. Intrusion Detection Systems are software or hardware systems that automate this process of monitoring and analyzing such events. Thus, an IDS, similar to a burglar alarm in the digital world, monitors a specific host or a network of hosts for intrusions and reports detected intrusions.

Depending on the source of input data, an IDS can be classified either as host-based or network-based IDS. A host-based IDS monitors the activity of a specific

host computer and reports detected intrusions. A network-based IDS, on the other hand, performs intrusion detection by analyzing the activity of a network of hosts. Depending on the approach used for intrusion detection, an IDS can be classified either as misuse-based or anomaly-based IDS. An IDS based on misuse detection classifies a network activity as an intrusion if it matches the stored network activity pattern of well-known attacks. Although, such IDS easily detects well-known attacks with a low false positive rate, never-before-seen attacks generally go undetected. On the other hand, an IDS based on anomaly detection principle is capable of detecting novel attacks. An anomaly-based IDS monitors the network for activities that differ from normal activity, and classifies such deviations as intrusions. However, IDS based on anomaly-based principle exhibits a high false positive rate.

Integrated Network Based Ohio University Network Detective Service (INBOUNDS) is an intrusion detection system developed at Ohio University. The ANDSOM module [41] for the INBOUNDS system performed intrusion detection using Kohonen's Self-Organizing Map algorithm. It captured the essential characteristic patterns in the multi-dimensional input space and signaled an alarm if the observed network connection was significantly anomalous from the normal values. As the module analyzed the network connections on an individual basis, each network connection was treated as a whole. In other words, individual components of a normal network connection, which could be anomalous, were not considered for analysis. In addition, the module deferred its analysis until the network connection was reported to be complete.

In this thesis, we describe in detail the development of a network-based and anomaly-based intrusion detection module for the INBOUNDS system. The module utilizes multiple Self-Organizing Maps for intrusion detection. We take into consideration time-based behavior of normal network connections, thus analyzing individual parts for anomalous behavior. We anticipate such an analysis would aid in a better and quicker response to network intrusions. Moreover, by lowering the false posi-

tive rate of the anomaly-based module, we expect the system to have an acceptable performance.

1.3 Organization of Thesis

In chapter 2, we provide an overview of the classification of intrusion detection systems, followed by a discussion on Self-Organizing Maps and the various intrusion detection techniques used by INBOUNDS. In chapter 3, we detail the design and implementation of the newly-developed intrusion detection module, based on multiple Self-Organizing Maps. We evaluate the implementation with a set of experiments in chapter 4, and a summary of the new approach along with some recommendations for future work is presented in the final chapter.

2. Background

In this chapter, we provide the background necessary to understand the intrusion detection module developed for our research. Section 2.1 presents an overview of the classification of intrusion detection systems. Section 2.2 introduces the basis of our intrusion detection approach, the Self-Organizing Maps. Finally, Section 2.3 deals with the existing intrusion detection techniques used by INBOUNDS, along with their limitations.

2.1 Classification of Intrusion Detection Systems

Intrusion detection systems attempt to detect attacks against computer systems and networks. Traditionally, intrusion detection systems can be classified using two approaches, namely, data source and detection approach. Section 2.1.1 and Section 2.1.2 provide a brief overview of each of these approaches.

2.1.1 Classification Based on Data Source

Based on the source of input data, an Intrusion Detection System (IDS) can be classified as follows:

- Host-based IDS: Deployed on a host computer, the IDS monitors only the activity of that particular host. Information such as operating system audit trails, registry entries and file accesses is used to detect an intrusion. Commonly used host-based IDSs include Haystack [58], MIDAS [42], and IDES [40].

- Multi host-based IDS: A set of hierarchical host-based IDSs running on individual hosts coordinate to detect any suspicious network activity as an intrusion. The master host is responsible for verification and analysis of the activities of

the network as a whole. Some examples of multi host-based IDSs include NIDES [29] and CSM [32].

- Network-based IDS: Deployed on a host computer, a network-based IDS passively monitors the network activities of a particular host or a network of hosts. Unlike host-based systems that seriously impact the performance of the host computer, a network-based IDS operates in a non-intrusive manner. The Network Security Monitor (NSM) [60] was the first IDS based on this approach. Other more recent network-based IDSs include SNORT [13] and CyberCop [3].

- Hybrid/Hierarchical IDS: A Hybrid/Hierarchical IDS combines the advantages of host-based and network-based IDSs. Improved intrusion detection capability is achieved through analysis of both host and network data. SRI International's Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) [49] is the most popular IDS based on this approach.

2.1.2 Classification Based on Detection Approach

Based on the approach used for intrusion detection, an IDS can be classified as follows:

- Misuse-based IDS: Also known as signature-based detection, an IDS based on misuse detection maintains a database of "signatures", unique network activity patterns, of well-known intrusions. A pattern matching program compares the observed activity pattern against the stored signatures, and raises an alert whenever a match is found. Although a misuse-based IDS detects known intrusions with fair amount of certainty, novel attacks go undetected. In other words, such an IDS generates few *false-positives*, false interpretations of normal behavior, but large *false-negatives*, false interpretations of abnormal behavior.

- Anomaly-based IDS: Also known as profile-based detection, an IDS based on anomaly detection classifies abnormal network activity as intrusion. Such an

IDS identifies critical network activity parameters and then builds profiles of normal genuine traffic. An observed network activity that is sufficiently deviant from the stored profile is then flagged as an intrusion. Although an anomaly-based IDS has the advantage of detecting never-before-seen attacks (i.e. few false-negatives), it is highly prone to false positives.

2.2 Self-Organizing Maps

Self-Organizing Map (SOM) [37], also known as Kohonen Self-Organizing Feature Map, is one of the most prominent artificial neural network methods. Based on unsupervised, competitive learning, the SOM algorithm provides a topology-preserving, ordered mapping from the high-dimensional input space onto a much lower-dimensional output space. Accordingly, the resulting image of the input space tends to depict clusters of input information and their relationships on the output space.

Developed by Professor Kohonen, the SOM has proved useful in the visualization of high-dimensional systems and processes, and discovery of categories and abstractions from raw data [59]. The SOMs have been successfully applied in the identification and monitoring of machine states, fault diagnosis and robot control [33]. Other practical applications of SOMs include automatic speech recognition [38], computer vision [31] and image analysis [22].

The SOM is a two-dimensional array of processing elements, typically called neurons. The SOM algorithm maps the data points from a complex high-dimensional input space onto the two-dimensional lattice of neurons. For example, if the input signal space could be characterized by k parameters or dimensions, then each input signal is represented as a k-dimensional vector in the input space. Accordingly, each neuron i in the two-dimensional lattice is also assigned a k-dimensional weight vector. Thus, the SOM algorithm compresses information from the k-dimensional input space onto the two-dimensional output space, thereby assisting in visualization of high-dimensional input data.

2.2.1 The SOM Algorithm

The SOM algorithm operates in two phases: the training phase and the testing phase. In the training phase, the neurons are trained to capture the essential characteristics of input data set. In the testing phase, the trained lattice of neurons is used for real-time operations. The subsequent sections detail with each of the SOM phases.

2.2.1.1 The Training Phase

As mentioned previously, the neurons in the two-dimensional lattice are initialized to k-dimensional values in the input range. The initialization of the neurons can either be random or linear, however convergence to final values is faster for the latter case. The neurons in the lattice are connected to adjacent neurons through a neighborhood relation that determines the topology or structure of the map. The lattice could either have rectangular or hexagonal topology, with the latter having better visualization properties.

The SOM training phase has the following important properties:

- Competitive: For each given input, all the neurons determine their activations, and the neuron with the lowest activation is declared as the "winner" neuron.
- Cooperative: The winner neuron spreads its activation over the neurons in its neighborhood to stimulate topographical ordering.
- Adaptive: The winner and neighboring neurons adjust their weight vectors so that they respond better for a similar input in the future.

The training phase of SOM, depicted in Figure 2.1, is as follows:

1. Select a k-dimensional input vector x from the training data set and feed it in parallel to all the neurons in the lattice. Each neuron determines its distance from the input data point in the k-dimensional input space. The most

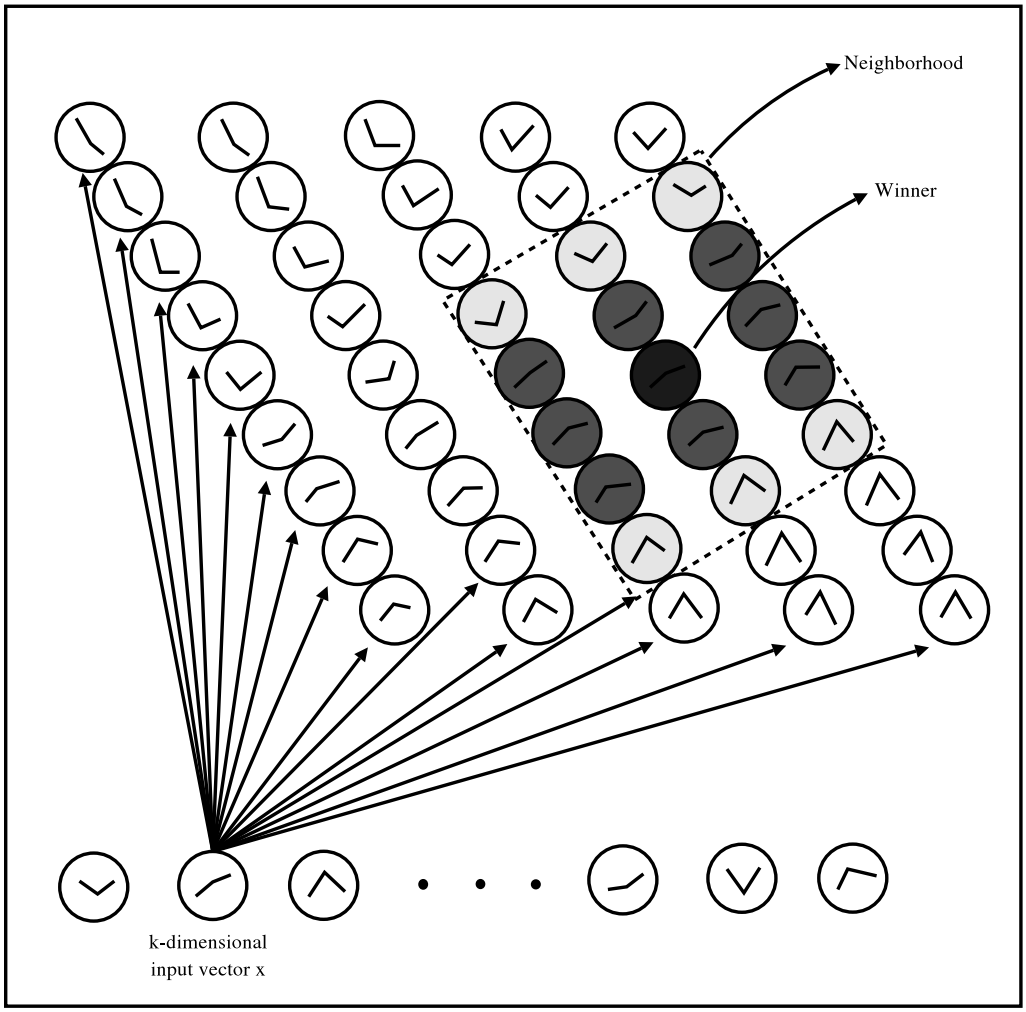

Figure 2.1. SOM Training
Each input vector is fed in parallel to all the SOM neurons, and the neuron that is closest to the input vector in the k-dimensional space is declared as the 'winner' neuron.

commonly used distance criterion for determining the winner neuron is the Euclidean distance. For k-dimensional space, the Euclidean distance between two points X$(x_1, x_2, \ldots, x_k)$ and Y$(y_1, y_2, \ldots, y_k)$ is given as follows:

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

The neuron that has the smallest Euclidean distance from the input data point is declared the "winner". Another commonly used distance criterion is the dot-product. For k-dimensional space, the dot product for two vectors X$(x_1, x_2, \ldots, x_k)$ and Y$(y_1, y_2, \ldots, y_k)$ is given as follows:

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

However, in this case the neuron with the maximum dot product is considered as the winner.

2. After the winner has been determined, the weight vectors of the winner and neighboring neurons are adjusted according to the following learning function:

$$m_i(t+1) = \begin{cases} m_i(t) + h_{ci}(t)[x(t) - m_i(t)] & \text{for each } i \in N_c(t), \\ m_i(t) & \text{otherwise,} \end{cases}$$

where,

t is an integer that represents discrete time coordinate. t is incremented by 1 for during every iteration of the training process.

$N_c(t)$ represents the neighborhood radius during iteration t of the training process.

x(t) represents the input vector chosen during iteration t of the training process.

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

$h_{ci}(t)$ represents the neighborhood function. $h_{ci}(t) = h(||r_c, r_i||, t)$. Here, r_c and r_i are the locations of the winner and the neuron i in the lattice.

There are two commonly used neighborhood functions: bubble and gaussian. In the bubble function, all the neurons in the neighborhood radius, N_c, are adjusted by the same factor. The bubble function can also be specified as $N_c(t)$ so that the neighborhood radius reduces during the training process. In this case, the neighborhood function $h_{ci}(t) = \alpha(t)$ for all neurons in the neighborhood, where $\alpha(t)$ is the learning rate factor. The gaussian neighborhood function, on the other hand, lets the adjustment factor vary as a bell-shaped gaussian function. The winner neuron gets adjusted by the maximum amount, with the adjustment factor for the neighboring neurons decreasing with increase in distance from the winner neuron. The gaussian function is specified as follows:

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

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

3. Repeat steps 1 and 2 until the training is complete. The number of steps needs to be determined prior to the beginning of the training phase as the rate of convergence of the neighborhood function and the learning rate are calculated accordingly.

2.2.1.2 The Testing Phase

After the completion of the training phase, the SOM is used for real-time operations. During the testing phase, the input vector is fed in parallel to all the neurons of the trained lattice. Each of the neurons determine their activations and calculate the winner neuron. The input vector is then associated to the k-dimensional value represented by the winner neuron.

2.2.2 Multiple Self Organizing Maps

One of the first applications of multiple self-organizing maps was in the field of script theory [43]. Ever since multiple SOMs have been applied successfully in areas such as exploratory data analysis, image retrieval, machine learning, and automatic speech recognition. In recent years, multiple SOMs, in conjunction with other neural network techniques, have also been applied in the area of intrusion detection [56, 57]. Several variants of the basic SOM algorithm have been proposed. Some of the variants aim at reducing the computational complexity, while others tend to preserve the topographical features using flexible map structures [26, 30].

Following are some of the most prominent applications of multiple SOMs:

- WEBSOM is a two-level architecture that provides an interactive graphical exploration of a full-text database by utilizing multiple SOMs [34].

- PicSOM, developed at Helsinki University of Technology, is an image retrieval system that employs multiple parallel SOMs for image indexing [39].

- GHSOM is a hierarchical structure composed of independent growing SOMs facilitating global orientation along with ease of navigation across individual branches [30]. The GHSOM acts as a basis for data organization in the SOM-based digital library (SOMLib) [53] and SOM-enhanced Juke Box (SOMeJB) [54].

2.2.3 Software Packages

Several public-domain software packages such as SOM_PAK [15], SOFM [14], NeNet [9] and CRAN-SOM [18] that implement the SOM algorithm are readily available. Similarly, SOMTOOLBOX [16] and xsom [20] are some software packages used for visualization of the SOM algorithm. SOM_PAK and SOMTOOLBOX have been used for our experimentation purposes.

The software packages, SOM_PAK and SOMTOOLBOX, are developed at the

Laboratory of Computer and Information Science, Helsinki University of Technology, Finland. SOM_PAK is a collection of programs written in ANSI C that provides the correct application of the SOM algorithm in the visualization of complex experimental data. Training parameters such as the learning rate factor, neighborhood radius and the number of iterations are specified as command-line options to the programs.

SOMTOOLBOX is a function package for MATLAB 5 implementing the SOM algorithm. Most features found in SOM_PAK are in some form available also in the SOMTOOLBOX. However, SOMTOOLBOX's superior support for advanced graphics and a powerful graphical user-interface make it an ideal candidate for visualization tasks. Moreover, since the SOM_PAK files can be accessed in SOMTOOLBOX, SOM training can be accomplished via SOM_PAK while visualization through SOMTOOLBOX.

2.3 INBOUNDS

INBOUNDS (Integrated Network Based Ohio University Network Detective Service) is a real-time, network-based, anomaly-based IDS being developed at Ohio University. The INBOUNDS system detects intrusions by examining network traffic gathered from various data sources, either in off-line or real-time mode. Based on anomaly detection principle, the system classifies any deviations from normal data patterns as intrusions. The primary goals of INBOUNDS 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, to detect never before seen attacks [24].

Prior to the work in this thesis, INBOUNDS used three anomaly-based approaches for intrusion detection: a statistical based technique [24], a neural network based technique [41], and a Bayesian network technique [61]. Sections 2.3.3 to 2.3.5 provide a brief description of each technique and Section 2.3.9 discusses some of their limi-

tations. Figure 2.2 represents the existing INBOUNDS architecture along with the various intrusion detection modules.

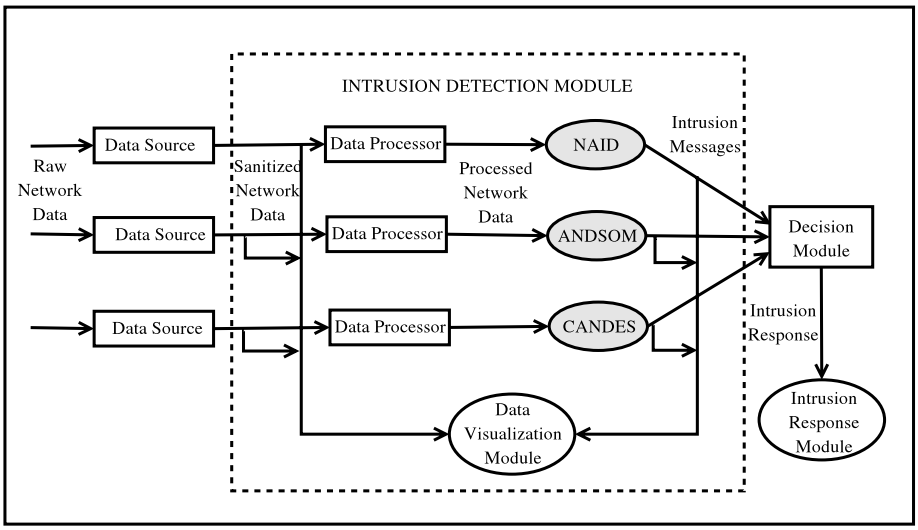

Figure 2.2. Existing INBOUNDS Architecture
The NAID, ANDSOM and CANDES modules are responsible for intrusion detection in the INBOUNDS system.

2.3.1 Data Source Module

The Data Source module gathers raw network data, and provides it as input to the Intrusion Detection module for analysis. Since the INBOUNDS system is a network-based IDS, it requires a Data Source module for each monitored network.

Raw network data can be collected with the help of packet sniffing tools such as tcpdump [17], Ethereal [5] and EtherApe [4]. INBOUNDS uses *Tcpurify* [27], a packet sniffing program, as its Data Source module. Since INBOUNDS is based on anomaly detection principle, application data in the network packets are irrelevant for our analysis. *Tcpurify* collects raw network packets from the wire, and discards the application data from the packet. Only the first 64 bytes of each packet, typically

covering the IP and TCP/UDP protocol headers, are reported. *Tcpurify* program can also be used to provide privacy by obfuscating the sender and receiver IP addresses. Symmetric key encryption, based on RC5 algorithm [23], is used to encrypt the IP addresses of the two communicating hosts.

2.3.2 Data Processor Module

The Data Processor module processes the "sanitized" network data obtained from the Data Source module. Depending on the intrusion detection module, it reports certain parameters that characterize the observed traffic.

2.3.3 Statistical-based Anomaly Detection

The statistical-based anomaly detection module, called Network Anomaly Intrusion Detection (NAID), generated certain network parameters that characterized the individual connections. The NAID module periodically reported five parameters, also known as dimensions, at a default time interval of 60 seconds[1]. Following criteria was used to compute the various parameters:

- Question - a packet from client to server containing data in the payload
- Answer - a packet from server to client containing data in the payload

Following parameters were reported for each network connection:

- Interactivity (Inter) - number of questions observed per second during the time interval
- Average Size Of Questions (ASOQ) - average size of payload data bytes transmitted from client to server during the time interval
- Average Size Of Answers (ASOA) - average size of payload data bytes transmitted from server to client during the time interval

[1]The time interval is tunable

- Question-Answer Idle Time (QAIT) - the idle time observed per second from the reception of a question to the transmission of the subsequent answer during the time interval
- Answer-Question Idle Time (AQIT) - the idle time observed per second from the reception of an answer to the transmission of the subsequent question during the time interval

In addition to the above parameters, a sixth parameter called Number of Connections (NOC) was also reported. This parameter monitored the total number of connections on a specific port.

TCPtrace [48], a TCP/IP network traffic analysis program, received "sanitized" network data as input and reported the above mentioned network parameters. *TCPtrace*, with the real-time module, generated the following three kinds of messages for each connection:

- An 'O' message when a new network connection was opened in the network.
- An 'I' message, generated periodically, reported the five parameters. The time interval had a default value of 60 seconds.
- A 'C' message when an active network connection was closed in the monitored network.

The NAID module had two different approaches to monitor network activity:

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

In the ACSH approach, the NAID module monitored all classes of network traffic destined to a particular host. This enabled the detection of an abnormality of a

particular service on a single host. In the ACAH approach, a specific type of network traffic could be monitored for all hosts on a network. Using this approach, the NAID module enabled the detection of an anomaly on a specific port on all hosts in a network.

For both the approaches, the NAID module used two methods for intrusion detection:

- Abnormality Factor Method

  In the Abnormality Factor method, a database called the historical data repository maintained the average and the standard deviations of each of the six dimensions of network connections. A key, which could either be an IP address, port or a combination of both, was used to access the database. During the real-time operation, the six dimensional values of live network connections were compared with the corresponding values stored in the database. For each dimension, the difference between the current value and the stored value was calculated and then divided by the corresponding standard deviation. The obtained value for each dimension measured the difference in units of standard deviations. A distance value greater than or equal to certain threshold was assigned an abnormality factor. If the sum of all such values for each dimension was greater than a pre-defined threshold value, the network connection was termed as an intrusion.

- Moving Average Method

  In the Moving Average method, the NAID module maintained a moving window of the average value for each dimension of a particular key. The key could be either an IP address, port or a combination of both. The moving or sliding window was defined to be active over a specified time period 'T'. Initially, the average values for each dimension were set to zero. During the real-time operation, a moving average was calculated by taking the average of all the

values collected in the past time windows. If the difference between the observed values and the moving average was greater than a pre-defined threshold value, the network connection was classified as an intrusion.

2.3.4 Neural Network-based Anomaly Detection

The neural network-based anomaly detection module, called Anomalous Network traffic Detection with Self-Organizing Maps (ANDSOM), identified certain network parameters that characterized the individual connections. The ANDSOM module used the Self-Organizing Map (SOM) algorithm to convert non-linear statistical relationships between data points in a high dimensional space into geometrical relationships between points in a two-dimensional map [37].

For each type of network traffic, the module periodically reported the five parameters, namely: Inter, ASOQ, ASOA, QAIT and AQIT, as explained in Section 2.3.3. An additional parameter, called Duration Of Connection (DOC), indicating the length of the connection was also generated. The Data Processor module for INBOUNDS reported the following three kinds of messages for each active network connection:

- An 'O' message signified the opening of a new connection in the monitored network. The 'O' message had the following format:

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

  The Timestamp field indicates the time when the connection was opened. The Protocol field specifies the protocol used for communication. At present, only two protocols are supported, namely TCP and UDP. The source and destination IP addresses and ports of the communicating hosts are reported next. The Status field indicates how the connection was opened. For a TCP connection, a value of 0 indicates opening SYN packets were seen during analysis while a value of 1 indicates the connection was already open prior to the start of analysis i.e.

SYN packets for the connection were not seen. For UDP connections, an 'O' message is generated when a packet is seen from one host to the other for the first time. However for UDP connections, the Status field is always assigned a value of 0.

- An 'U' message, generated periodically, reported the five dimensions: Inter, ASOQ, ASOQ, QAIT, AQIT. The time interval had a default value of 60 seconds[2]. The 'U' message had the following format:

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

- A 'C' message indicated the closing of a connection in the monitored network. The 'C' message had the following format:

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

  The Status field has a value of 0 if a TCP connection was closed with two FINs, and 1 if a RST packet closes the connection. For UDP traffic, the Status field is always allocated a value of 0.

Figure 2.3 illustrates the various steps in the training of the ANDSOM module. The ANDSOM module received these messages as input from the Data Processor module. A submodule, TRC2INP, generated six dimensional 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) for each network connection. The rationale for the use of L_QAIT and L_AQIT values instead of QAIT and AQIT values respectively was to reduce the number of false positives found earlier. The time difference between the generation of 'O' and 'C' messages for a network connection was reported as the sixth dimension, Duration Of Connection (DOC), for

[2]The time interval is tunable

that connection. The Normalizer submodule, which received these values as input from the TRC2INP submodule, produced normalized six dimensional vectors as its output. After determining the lattice dimensions and size, the neurons in the lattice were linearly initialized within the range of the input six dimensional vectors. The ANDSOM module performed SOM training in two phases: an initial phase with a high learning factor, large neighborhood radius and lesser number of iterations, and final fine-tuning phase with a low learning factor, relatively low neighborhood radius and larger number of iterations.

After the completion of training, a validation check was performed to evaluate the trained lattice of neurons. The locator program, written for this purpose, took as its input the initial un-normalized training data set, mean and standard deviation of each dimension of the data set and the fully trained map. After normalization of each input vector, the locator program calculated the distance between the input vector and winner neuron. If 95.44% of all input vectors fall within a distance of 2σ from a winner neuron, the map was assumed to be trained adequately. During the operation phase of SOM, the test vector was fed into the trained map and the winner neuron was determined. An anomaly was raised if the distance between the test vector and the winner neuron was greater than 2 units.

2.3.5 Bayesian Network-based Anomaly Detection

Causal Tree Anomaly-based Network-based DEtection System (CANDES) is an anomaly detection module based on Bayesian network technique. A Bayesian Network is a directed graph model that encodes the relationship between the parameters of input domain into its structure and probability tables [50]. The CANDES module modeled the Bayesian network as a causal tree, wherein the nodes represented either the parameters of input domain or hypotheses accorded some degree of belief, and the links symbolized the relationship between the parameters. Each node possessed discrete values equal to the number of states of the corresponding parameter or degrees of belief of the corresponding hypothesis. Bayes Theorem calculated the

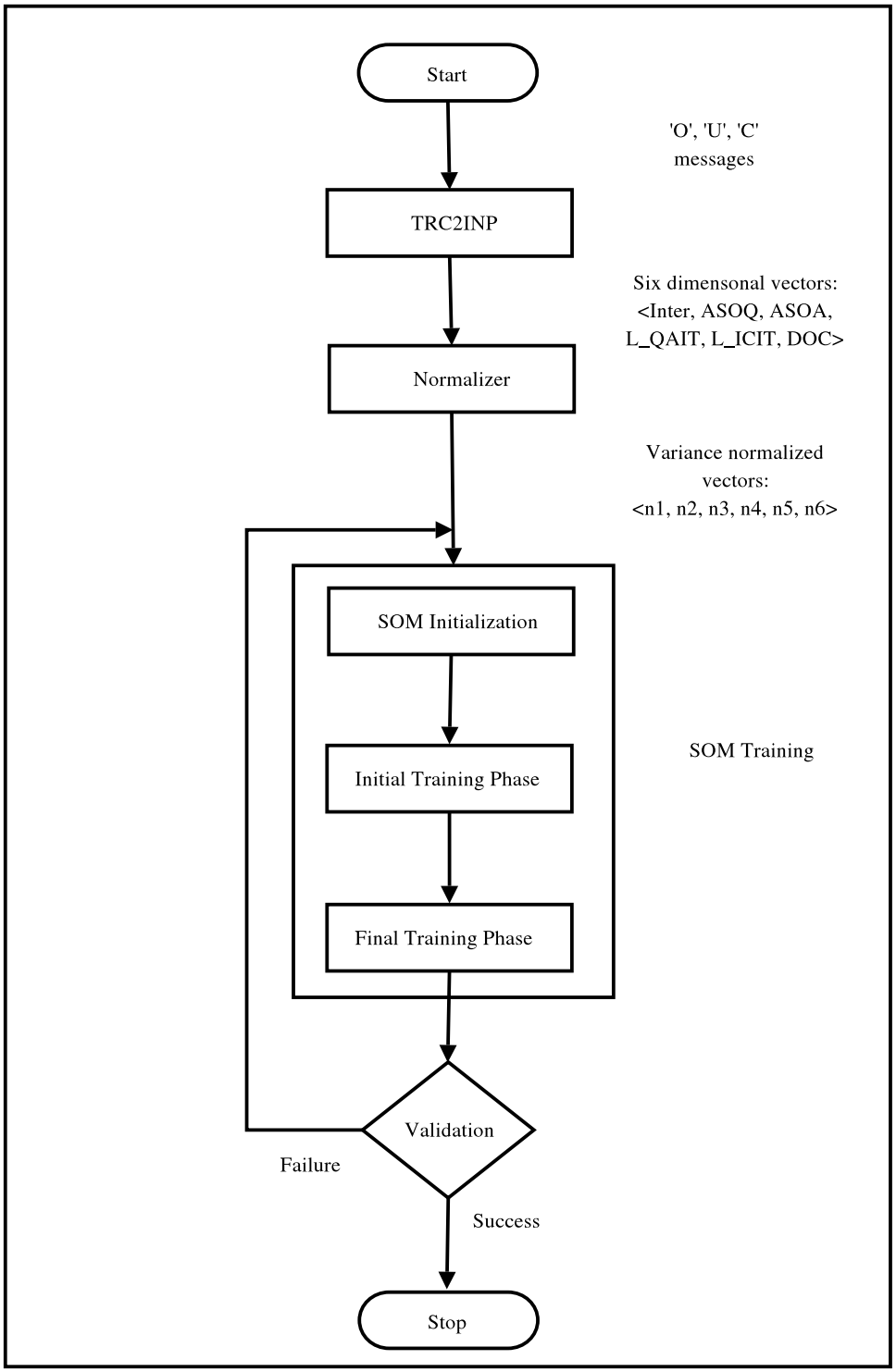

Figure 2.3. ANDSOM Training
The 'O', 'U' and 'C' messages reported the six parameters characterizing the network connections. Normalized six-dimensional values were used to train the SOM lattice neurons [41].

conditional probabilities for each node that quantified the relationships between the input parameters. Each entry of the Conditional Probability Table (CPT) indicated the probability of the child node assuming a particular value, given a particular parent node value.

To aid in the classification of certain kinds of attacks like Denial-Of-Service (DOS) attacks, the CANDES system proposed new network parameters. These network parameters characterized both a particular network service as well as the connections to that service. A module for *TCPtrace*, called Bayes module, reported these parameters for each network service at a default time interval of 60 seconds [3].
The following network parameters were reported periodically:

- Number Of Open Connections (NOOC) - number of connections opened to a particular network service in the last time interval
- Number Of Failed Connections (NOFC) - number of connections failed to a particular network service in the last time interval
- Number Of Packets (NOP) - total number of packets received by a monitored host, or a network on a particular network service in the last time interval
- Number Of Same client IP Connections (NOSC) - total number of connections made to a particular network service from the same Client IP address in the last time interval

The Bayes module generated as its output three types of messages:

- The 'A' message reported the NOOC, NOFC, NOP and NOSC parameters for a particular port for all the monitored hosts.
- The 'H' message reported the NOOC, NOFC, NOP and NOSC parameters for a particular port for each host of the monitored network.

[3]The time interval is tunable

- The 'C' message reported the ASOQ and ASOA parameters, as described in Section 2.3.3, for each active network connection.

The CANDES module received these messages as input from the Bayes module. The leaf nodes of the Bayesian network represented the six parameters reported by the Data Processor module. The values (probabilities) of leaf nodes, also known as evidence nodes, were calculated from reported values. Each evidence node could assume four discrete values: no anomaly, low anomaly, medium anomaly, and high anomaly, depending on the deviation from the normal value. The internal nodes, also known as hypothesis nodes, Network Service Behavior, and Connection Behavior represented the hypotheses: "Network Service Behavior is anomalous", and "Connection Behavior is anomalous" respectively. The values of these internal nodes were calculated by Bayesian Network algorithms and used to determine the presence of an anomaly.

2.3.6 Decision Module

The Decision module as depicted in Figure 2.2 is work in progress. Its goal is to integrate and evaluate the intrusion alerts from the three IDS modules: statistical-based NAID module, neural network-based ANDSOM module, and Bayesian Network-based CANDES module, and convey the appropriate final decision to the Intrusion Response module.

2.3.7 Intrusion Response Module

As shown in Figure 2.2, the Intrusion Response module is also work in progress. The responsibility of this module is to take active response on the intrusion alert generated by the Decision module. Some of the active responses include: firewall rules to block the particular network connection or the particular port, rate limiting the bandwidth of that class of network traffic, etc.

2.3.8 Data Visualization Module

The Data Visualization module provides a real-time picture of the various connections made in and out of the network monitored by the INBOUNDS system. The Data Visualization module runs the *networkgraphserver* program developed for this purpose. Implemented in Java, the program depicts in real-time the hosts of the monitored network as icons, and connections between the hosts as lines.

2.3.9 Limitations of Existing Approaches

The existing approaches used for anomaly detection in INBOUNDS have certain limitations. The statistical-based technique described in Section 2.3.3 had only one reference data point for each class of network traffic in six-dimensional space, namely, the average values of all six dimensions for that particular network traffic. Since the distribution of training data could occur as multiple distinct clusters, the abnormality factor method for statistical technique gave rise to a significant number of false-positives. Moreover, the moving average method of statistical technique can generate false-negatives when an intrusion gradually increases its activity in successive time windows. Although the moving average method detected attacks carried over a very long period of time, it generated false-positives because of its dependence of appropriate threshold values.

The neural network-based approach described in Section 2.3.4 overcomes the limitations of the statistical-based approach by storing multiple reference points for each type of network traffic. Nevertheless the ANDSOM module had its own set of drawbacks. The ANDSOM module captured the essential characteristic patterns of the input signal space and signaled an alarm if the observed input signal was sufficiently deviant from the normal values. Since analysis was performed on a per-connection-basis, certain types of intrusions such as Denial Of Service (DOS) attacks went undetected. Moreover, as every network connection was treated as a single entity, the ANDSOM module ignored the time-based behavior of the network connection. In

other words, individual sections of an otherwise normal network connection were not considered for analysis. Additionally, the ANDSOM module performed intrusion detection only after the connection was closed. That is, analysis was carried out when the Data Processor module reported the 'C' message for the network connection. Furthermore, the ANDSOM module could generate false-negatives for network traffic that exhibited parametric values similar to those of normal traffic. Also, the SOM training phase was assumed to be complete when 95.44% of the training data vectors had a winner neuron within 2σ. In other words, the remaining 4.56% of the training data vectors were themselves classified as network intrusions by the ANDSOM module.

Though the Bayesian network-based anomaly detection approach used in CANDES successfully detected DOS attacks such as *finger bomb* [6], it had its own limitations. The CANDES module was based on the categorization of the input parameters into four different anomaly levels: no anomaly, low anomaly, medium anomaly and high anomaly. Since these levels were manually defined at the end of training phase, an error in calculations resulted in false-positives or false-negatives. Moreover, since the criteria for anomaly classification was time-independent, increased network activity, which could be normal at that particular time of the day, would trigger a false positive. Besides, the CANDES module used simple relationships between input parameters to detect anomalies. As these relationships were determined without analysis of their exact dependency, combining unrelated parameters to classify intrusions lead to false-positives or false-negatives.

Our research is motivated by the need to design a more powerful technique for intrusion detection that overcomes some of the aforementioned limitations. In particular, we would like to perform a time-based characterization of normal network behavior. We anticipate such a characterization would lead to a better network intrusion detection system. Moreover, analyzing the network connection before it closes would result in a near real-time response to network intrusions.

3. Multiple Self-Organizing Map-based Intrusion Detection System (MSIDS)

This chapter describes the design and working of Multiple Self-Organizing Map-based Intrusion Detection System (MSIDS) module developed for INBOUNDS. Section 3.1 outlines the updated INBOUNDS architecture, followed by the description of the Data Processor module in Section 3.2. Section 3.3 deals at length with the new approach proposed for intrusion detection.

3.1 INBOUNDS Architecture

As shown in Figure 3.1, the INBOUNDS system consists of the following modules: Data Source, Data Processor, Intrusion Detection, Data Visualization, Decision and Intrusion Response. The INBOUNDS system inputs the network data, received from the various data sources, to the intrusion detection module for anomaly detection. The intrusion detection module analyzes the network data for intrusion and reports accordingly to the intrusion decision module.

3.2 Data Processor Module

The Data Processor module obtains "sanitized" network data from the Data Source module, and processes it to produce messages indicating the opening, activity, and closing of live network connections.

TCPtrace, which acts as the Data Processor module, receives the network data as input and reports certain network parameters for each individual network connection. We redefine the criteria [41] used to compute the new parameters for our MSIDS module:

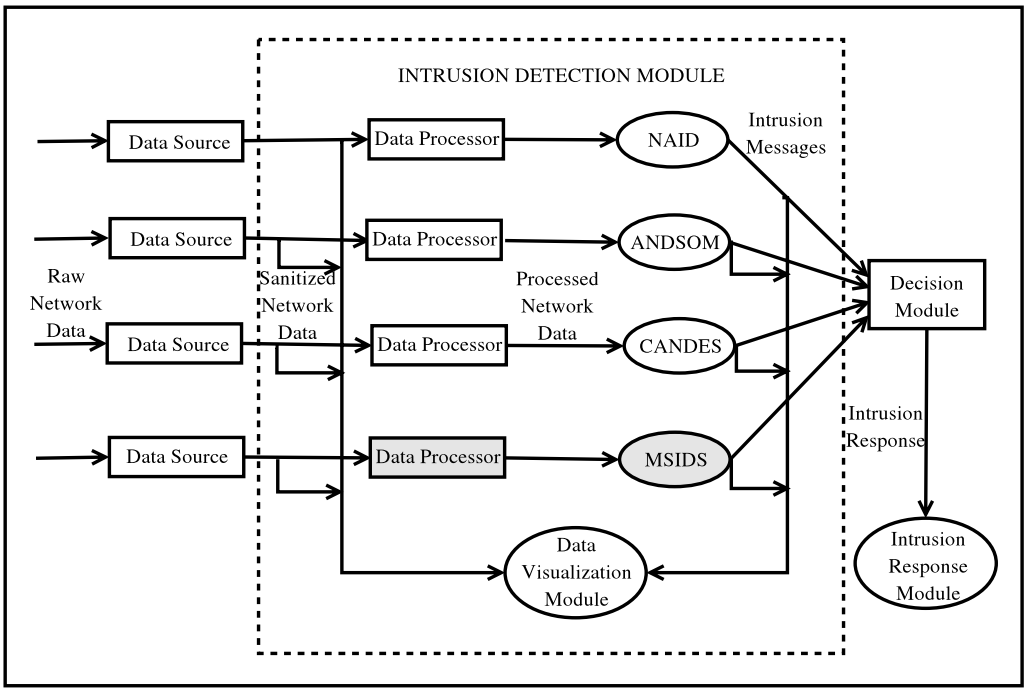

Figure 3.1. Updated INBOUNDS Architecture
The MSIDS module analyzes the time-based behavior of network connections for anomalies in the INBOUNDS system.

- Question - the set of packet(s) containing data in the payload transmitted from client to server, before the transmission of next packet containing data in the payload in the other direction from server to client
- Answer - the set of packet(s) containing data in the payload transmitted from server to client, before the transmission of next packet containing data in the payload in the other direction from client to server
- Cycle - the collection of data from the first packet of a question until the last packet of the subsequent answer

It should be noted that the above values are calculated at the transport layer. For many application protocols, there is one-to-one correspondence between the cycles that occur at the transport layer and those that occur at the application layer because

of the simple query-response behavior of a typical client-server interface. For such protocols, the cycle behavior at the transport layer directly corresponds to the cycle behavior at the application layer. However, for application protocols in which there may be multiple outstanding queries made before an answer is received, possibly while receiving answers to prior queries, there is no logical correspondence between the cycles at the transport and application layers. An example of one such application protocol is HTTP/1.2 pipelined queries. Therefore, we hypothesize the above criteria would not work well for such protocols.

Following network parameters are reported by the Data Processor module for each active network connection:

- Size Of Question (SOQ) - the total number of payload data bytes transmitted from client to server within one cycle
- Size Of Answer (SOA) - the total number of payload data bytes transmitted from server to client within one cycle
- Question-Answer Idle Time (QAIT) - the time elapsed (in seconds) between the reception of the last packet of a cycle's question and the transmission of the first packet of the cycle's answer
- Inter-Cycle Idle Time (ICIT) - the time elapsed (in seconds) between the reception of the last packet of a cycle's answer and the transmission of the first packet of the subsequent cycle's question, or close of the connection

The output of the Data Processor module consists of messages that are used as input to the Intrusion Detection module of INBOUNDS. Three types of messages are generated for each active network connection, namely: 'O', 'L' and 'C'. The 'O' and 'C' messages have the same semantics as described in Section 2.3.4. We introduce a new message, 'L', that reports the four parameters for the network connection. The following paragraph provides a summary of the three messages:

- The 'O' (Open) message is generated when a new connection is opened in the monitored network. The format of an 'O' message is:

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

- The 'L' (Loop) message is generated throughout the lifetime of the network connection. The format of an 'L' message is:

  `L Timestamp Protocol <src host:port> <dst host:port> SOQ SOA QAIT ICIT`

  The 'L' message reports the parameters SOQ, SOA, QAIT and ICIT, as explained earlier, for each network connection for the particular traffic.

- The 'C' (Close) message is generated when an active connection is closed in the monitored network. The format of an 'C' message is:

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

Figure 3.2 illustrates the above mentioned criteria, along with the four MSIDS network parameters. The timestamps, T_A, T_B, T_C, and T_D, indicate the exact time instances where the parameters are calculated. At timestamp T_B and T_D, we determine the SOQ and SOA parameters respectively. While the difference between timestamps T_A and T_B is reported as QAIT parameter, the difference between timestamps T_C and T_D is reported as ICIT parameter. The 'L' message, generated at timestamp T_D, reports all the four network parameters for the cycle. It should be noted that, for the last cycle, the 'C' message for the connection triggers the generation of the 'L' message that reports the parameters for it.

3.3 MSIDS Module

The steps in the training of Multiple Self-Organizing Map-based Intrusion Detection System (MSIDS) module are illustrated in Figure 3.3. The MSIDS module employs the Self-Organizing Map algorithm, mentioned in Section 2.2.1, to build profiles of normal network traffic and thereby detect anomalous network traffic.

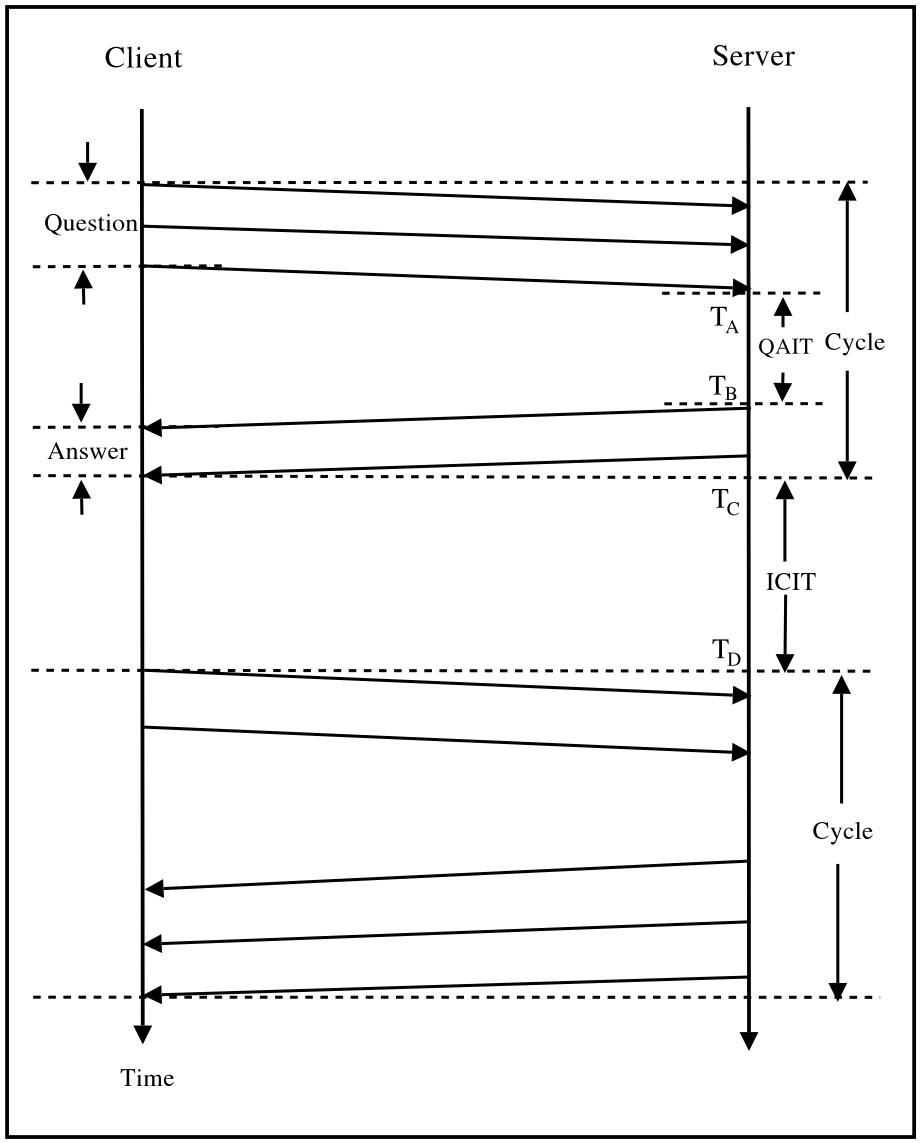

Figure 3.2. Criteria for MSIDS Parameters
The timestamps, T_A, T_B, T_C, and T_D, indicate the exact time instances where the various MSIDS network parameters, SOQ (Size Of Question), SOA (Size Of Answer), QAIT (Question-Answer Idle Time), and ICIT (Inter-Cycle Idle Time), are calculated. At timestamp T_B, transmission of first data packet from server to client enables the calculation of SOQ and QAIT values for that cycle. Similarly, at timestamp T_D, transmission of first data packet from client to server allows the calculation of SOA and ICIT values for the preceding cycle.

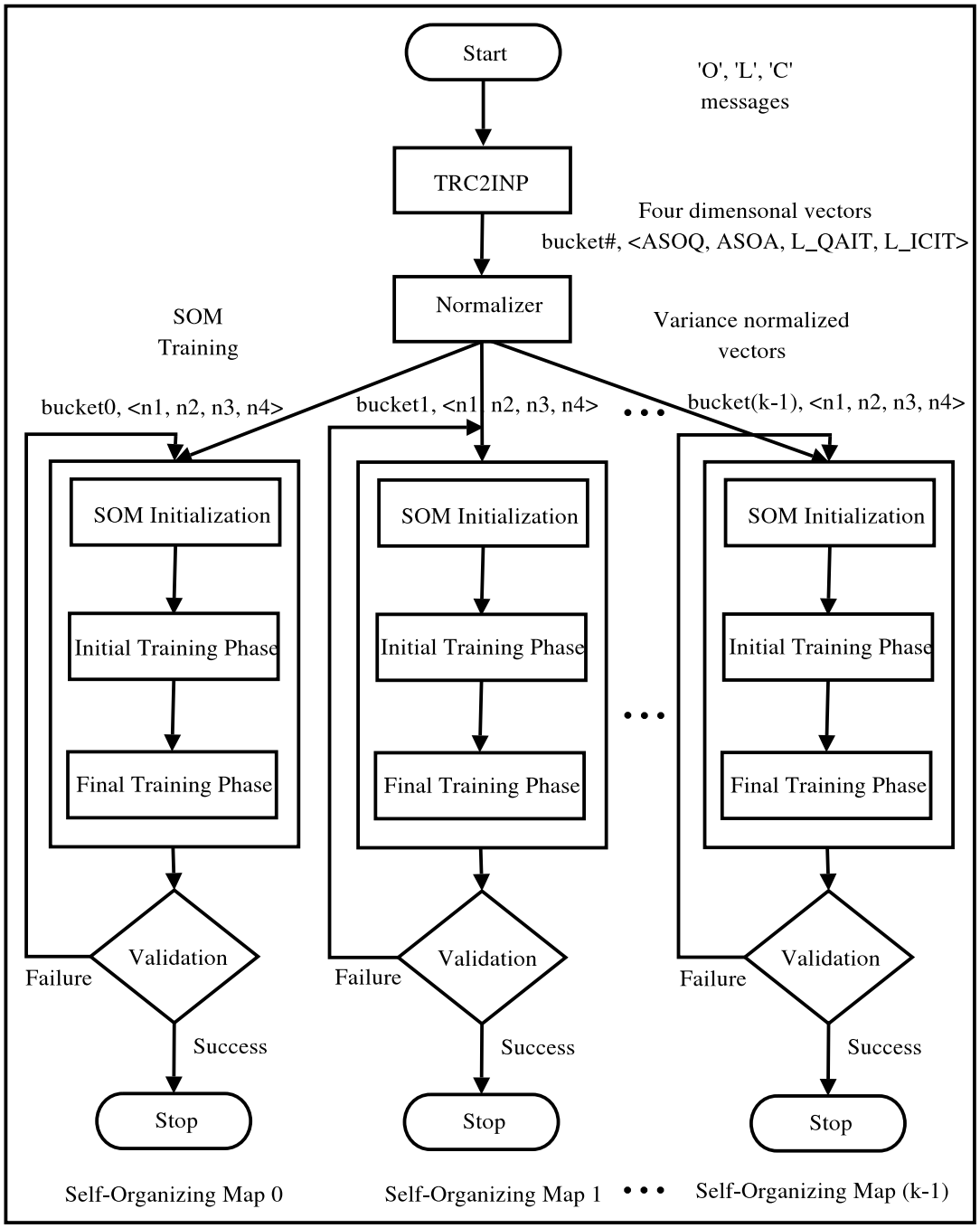

Figure 3.3. MSIDS Training
The 'L' messages report four parameters that characterize the network connections. The TRC2INP module 'exponentially' groups these values in different buckets. Based on the bucket number, normalized four dimensional vectors train the neurons of various SOMs.

3.3.1 TRC2INP Submodule

The TRC2INP submodule, that runs the *trc2inp* program, processes the 'O', 'L' and 'C' messages received from the Data Processor module and generates four dimensional values: Average Size of Questions (ASOQ), Average Size of Answers (ASOA), Log base 10 of Question-Answer Idle Time (L_QAIT) and Log base 10 of Inter-Cycle Idle Time (L_ICIT) for each "bucket", a collection of 'L' messages, per network connection.

The *trc2inp* program maintains a hash table of active network connections. Each entry in the hash table is identified by a four-tuple <Source IP address, Source Port, Destination IP address, Destination Port>. Moreover, each hash table entry maintains a list of buckets to group the 'L' messages of that particular network connection. Each bucket in the list maintains four values: ASOQ, ASOA, QAIT and ICIT, for the 'L' messages stored in that bucket. The current implementation of MSIDS module "exponentially" stores the 'L' messages in a list of buckets. In other words, the first 'L' message is stored in bucket 0, the first two 'L' messages are stored in bucket 1, the first four 'L' messages are stored in bucket 2, and so on. The rationale for grouping multiple 'L' messages into buckets is to better characterize the behavior of normal network traffic over the lifetime of a connection. The goal is to potentially reduce the number of false positives that would be generated if an individual SOM is dedicated for each 'L' message. The other benefit of grouping the 'L' messages is to limit the number of buckets to a constant value. Upon receiving an 'O' message, the *trc2inp* program calculates the hash value for the new network connection and adds an entry in the hash table.

Upon receiving an 'L' message, the hash table entry for the network connection is found using the four-tuple <Source IP address, Source Port, Destination IP address, Destination Port> in the message. A counter, *lcnt*, is maintained to keep track of the 'L' messages for the network connection. Depending on the value of *lcnt*, the values SOA, SOA, QAIT, ICIT reported in the 'L' message are added to the dim1, dim2,

dim3 and dim4 fields of the appropriate bucket and new values for ASOQ, ASOA, QAIT and ICIT are calculated. This procedure is repeated for each 'L' message received, updating the ASOQ, ASOA, QAIT and ICIT values of the appropriate bucket of the connection.

Upon receiving a 'C' message, the hash table entry corresponding to the pertinent network connection is removed. For each bucket in the linked-list, the *trc2inp* program reports dim1 and dim2 as ASOQ and ASOA respectively. On the other hand, the log base 10 values of dim3 and dim4 are reported as L_QAIT and L_ICIT respectively. The decision to use L_QAIT and L_ICIT values instead of QAIT and ICIT was not only to reduce the number of false positives found in earlier experiments, but also to better characterize widely varying time values.

3.3.2 Normalizer Submodule

The four-dimensional vectors, generated by TRC2INP submodule, can be directly used to train SOMs for each class of network traffic. However, the SOM algorithm uses the Euclidean distance as its criterion to determine the winner neuron, so scaling of dimensions becomes essential. For example, if one dimension has values in the range of [0,. . . ,1] and another in the range of [0,. . . ,100], the latter will certainly dominate the map organization. In most cases, one would like to assign all dimensions equal importance. One approach would be to normalize the data so that each dimension of input vectors will have unit variance and zero mean. The *normalizer* program, written for this purpose, normalizes the input vectors so that none of the dimensions has a considerable impact on the training result.

The *normalizer* program takes as input the four-dimensional vectors of each bucket and produces the corresponding normalized four-dimensional vectors. The *normalizer* program operates in two stages. In the first stage, it calculates the mean (μ_{ij}) and standard deviation (σ_{ij}) of each dimension i per bucket j. It stores these calculated values the mean in separate data files for latter use. In the second stage, the *normalizer* program takes as input each four-dimensional vector $< d_1, d_2, d_3, d_4 >$

of bucket i and produces the corresponding normalized four-dimensional vector $< n_1, n_2, n_3, n_4 >$, such that $n_i = \frac{d_i - \mu_{ij}}{\sigma_{ij}}$, where μ_{ij} and σ_{ij} are the mean and standard deviation of dimension i and bucket j respectively.

At the end of the normalization process, normalized four-dimensional input vectors per bucket, stored in separate files, are used for SOM training.

3.3.3 SOM Training

The software packages SOM_PAK and SOMTOOLBOX, mentioned in Section 2.2.3, are used to build a SOM for each bucket. While SOMTOOLBOX's *som_lininit* function initializes the neurons of the SOM lattice, SOM_PAK's *vsom* program performs SOM training for each bucket.

Before the onset of SOM training, there are four values that need to be determined: number of neurons, dimensions of the lattice, lattice shape, and topology. With the neighborhood function controlling the smoothness and generalization of the mapping, the number of neurons can be chosen as large as possible. The *som_lininit* function uses the following heuristic formula to determine the number of neurons: *munits*=5*sqrt(*num*), where *munits* denote the number of neurons in the lattice and *num* denote the number of training samples.

Once the number of neurons in the lattice are calculated, the lattice dimensions can be computed. The *som_lininit* function determines the two biggest eigenvalues of the training data set, and sets the ratio between the two dimensions of the lattice to the ratio between these two eigenvalues. The actual dimensions are then adjusted such that the product of the dimensions is as close as possible to the value *munits*.

The default recommended shape for the lattice is sheet, unless the shape of the data manifold is known beforehand. Hence, the torrid and circular shapes should be used only if the data is known to be circular. The lattice topology can either be hexagonal or rectangular, although the former is known to have better visualization properties as all the neighboring neurons are equidistant. The *som_lininit* function creates a two dimensional lattice of neurons with hexagonal topology.

After determining the lattice dimensions and lattice size, the neurons are then initialized, either linearly or randomly, to four-dimensional values in the input data range. The SOM algorithm is known to converge faster to a good solution in case of linear initialization. For linear initialization, the *som_lininit* function selects a mesh of points from the four-dimensional min-max cube of training data set. It is important to note that the initialization of neurons is irrespective of the lattice shape; the lattice is always assumed to be sheet-shaped.

SOM_PAK's *vsom* program performs SOM training in two phases: an initial training phase, and a final fine-tuning training phase. The *vsom* program determines the winner neuron for each input vector and updates the associated vectors of those neurons in the neighborhood of it according to the selected neighborhood function. The initial value of the neighborhood radius is defined for each phase and is linearly reduced to one during training. The initial learning rate factor is also defined for each phase and is linearly reduced to zero at the end of training.

In the initial phase, the number of iterations is chosen to relatively low, in order of thousands. A gaussian neighborhood function is used, with the neighborhood radius set to a high value, typically the smaller of the lattice dimensions. The neighborhood radius is linearly reduced to value of 1 as the training proceeds. The learning rate factor, linearly reduced to zero, is also chosen to a high value close to unity, typically 0.9. Most of the map organization happens in the initial training phase. In the final fine-tuning phase, the number of iterations is set to a high value, typically 100,000. The gaussian neighborhood radius is set to a relatively smaller value in this case. Similarly, the learning rate factor, chosen to be relatively small, say 0.05, is linearly reduced to a value of 0 as the training concludes. At the end of the final fine-tuning phase, each SOM lattice captures the essential characteristic patterns of the corresponding (bucket) data set.

At the end of the two phases of training, each SOM lattice is evaluated with the *locator*, a program written for this purpose. The *locator* program takes as input

un-normalized training data set for each bucket, the corresponding trained lattice of neurons, and the corresponding mean and standard deviations generated by the *normalizer* submodule. For each input vector, the *locator* program first normalizes it based on the corresponding mean and standard deviation, and then feeds it to all the neurons of the corresponding trained SOM lattice. The winner neuron for the input vector is then determined, and the normalized vector, the topographical location of the winner in the SOM lattice, and the measured distance are reported as output by the *locator* program. The output gives the number of input vectors that have a winner neuron within a distance of 2σ units in the four-dimensional space. Since gaussian distribution was used as the neighborhood function during the training phase, ideally at least 95.44% of the samples must fall within 2σ of the mean. Using this property as a heuristic for SOM validation, we consider the SOM lattice to be completely trained if 95.44% of training data have a winner neuron within 2σ units in the four-dimensional space.

This thesis assumes that normal network data (i.e. training data) follow a standard Gaussian distribution for the four parameters, and that the standard deviation (σ) is a measure of distance in the four-dimensional input space for each datum. If this assumption for normal network data holds true, then σ for the training datum can be interpreted as a probability of the datum occurring using traditional statistical techniques. For many Internet attacks, however, it is clear that the four computed network parameters for attack data are *not normal network data*, and hence the standard Gaussian probability associated with a particular value of σ is not valid and has no meaning. We still use σ, however, for attack data to provide a metric of how abnormal a particular attack datum actually is, even though this usage of σ is not standard and could lead to confusion if interpreted as a probability. For example, if an attack datum exhibits a standard deviation value of 4.5 units, then it would indicate a Gaussian probability of 7.89E-09.

3.3.4 SOM Operation

The Data Source, Data Processor and the MSIDS modules, as shown in Figure 3.1, communicate with each other in real-time using the *icomm* library. The *icomm* library provides the modules with a set of interface functions, so that the underlying communication mechanism can be module-independent. The current implementation of *icomm* library makes use of TCP sockets API. Although SOM training was performed offline using network dump files, the MSIDS module performs intrusion detection in real-time.

During the real-time SOM operation, the Data Source module, running the *tcpurify* program, reports "sanitized" network packets to the Data Processor module. The Data Processor module, running the *TCPtrace* program with the INBOUNDS module, in turn processes these sanitized network packets to produce 'O', 'L' and 'C' messages for each network connection. The *trc2inp* program, a submodule of MSIDS module, receives these messages and reports four-dimensional vectors on a bucket-basis. The output is received by the *locator* program that operates for a specific class of network traffic. The *locator* program normalizes these four-dimensional vectors, reported per bucket, using the mean and standard deviations of the respective buckets generated during the SOM training. Each normalized vector is then fed in parallel to all the neurons of the corresponding trained SOM lattice. A winner neuron is determined for each input vector fed to the respective SOM lattice. If the distance between the winner neuron and the input vector for any of the SOM lattices is greater than 2σ units, the network connection is classified as an intrusion.

4. Experimental Results

As detailed in Chapter 3, the MSIDS module performs intrusion detection by analyzing the time-based behavior of network connections using multiple SOMs. In this chapter, we present some of the experimental results obtained from the operation of the MSIDS module. We organize this chapter in two sections. In Section 4.1, we describe the successful detection of the Sendmail buffer overflow attack by our module [12, 2]. Later, we provide a comparison of the attack detection with that by the ANDSOM module [41]. In Section 4.2, we substantiate our claim with the effective detection of another attack, namely: Apache/OpenBSD stack overflow attack [28].

4.1 Sendmail Buffer Overflow Attack

Simple Mail Transfer Protocol (SMTP) is the default standard protocol for transmission of "electronic mail" (email) across the Internet [51]. SMTP is a relatively simple, text-based protocol that supports 7-bit ASCII characters. The protocol is specified in RFC 2821 [35], which is an updated version of RFC 821 [51]. Multipurpose Internet Mail Extensions (MIME) extends the original specification of SMTP by allowing email messages to contain characters from other encodings as well as 8-bit binary content such as files containing images, sounds and movies. The MIME extensions are specified in RFCs 2045-2049 [46, 47, 36, 44, 45]. Mail Transfer Agents (MTAs) use SMTP for transfer of email messages amongst hosts. The client MTA establishes a connection with a server MTA on tcp port 25 to send an email message. Figure 4.1 depicts a typical network, packet-level interaction between the client MTA and the server MTA.

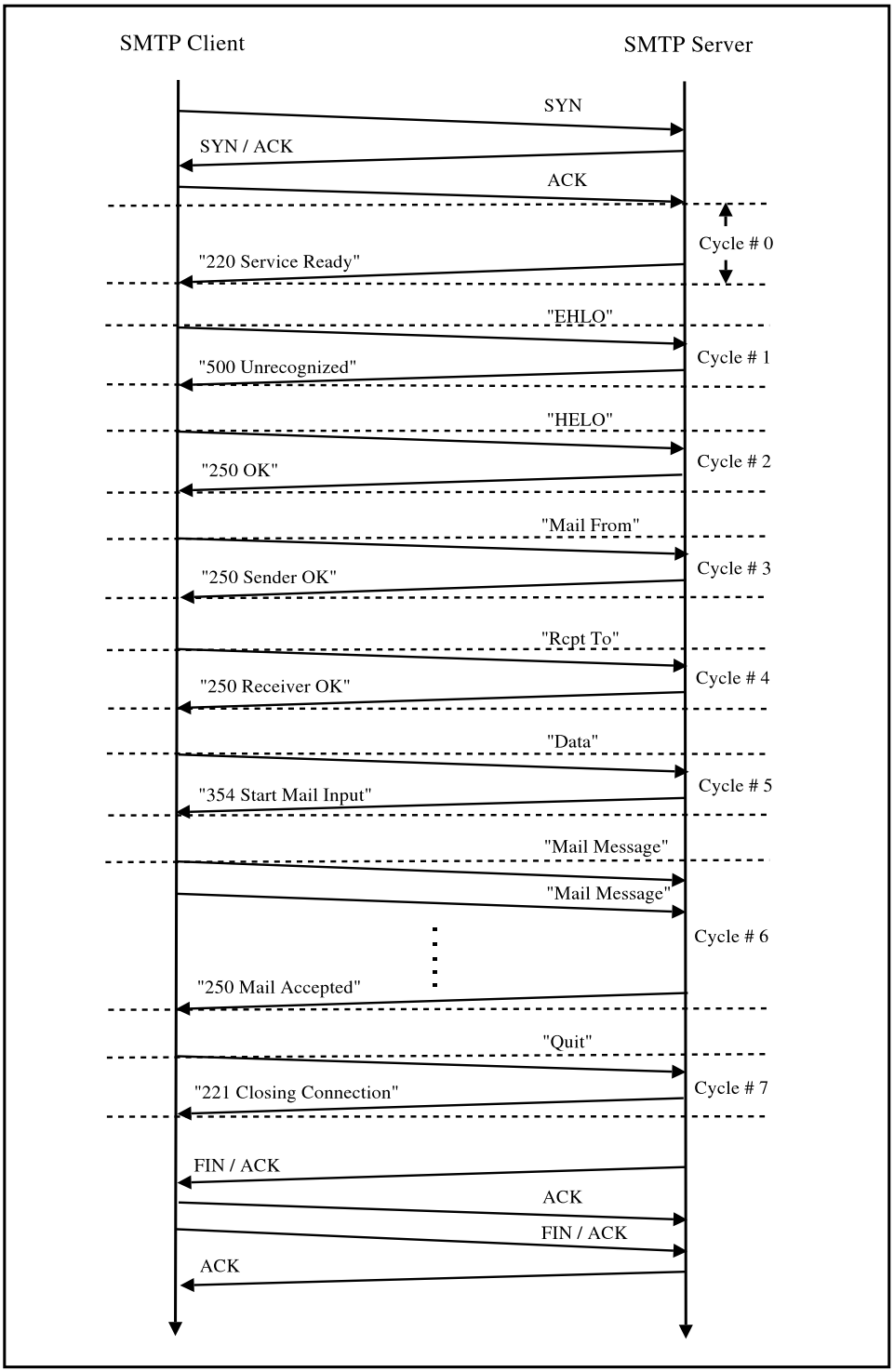

Figure 4.1. SMTP Connection Timeline
The adapted figure illustrates a typical interaction between SMTP client and SMTP server in terms of cycles [41].

Following are the steps involved in the transmission of an email message from the SMTP client to the SMTP server:

- The SMTP client uses the TCP three-way handshake to establishes a connection with the SMTP server, which is listening on port 25.
- After receiving the "220 Service Ready" message from the server, the client sends the "EHLO" command to the server. An SMTP client supporting SMTP extensions should start an SMTP session using the "EHLO" command instead of the default "HELO" command. If the SMTP server does not support SMTP extensions, it responds back with "500 Unrecognized" message.
- If the SMTP server does not support SMTP extensions, the SMTP client sends the "HELO" command in response. The SMTP server responds with "250 OK" message.
- The client then informs the sender of the email with the "Mail From" command. The server replies back with "250 OK" message acknowledging the command.
- The client then specifies the recipient of the email message with the "Rcpt To" command. If the intended recipient is a valid user, the server responds with "250 OK" message else with "550 No such user here" message indicating an invalid user.
- After a positive acknowledgment from the server, the client issues the "Data" command. If the SMTP server is ready to receive the mail contents, it responds to the "Data" command with "354 Start Mail Input" message.
- The SMTP client then transfers the contents of the email using multiple TCP segments, if required. The SMTP server responds back with "250 Mail Accepted" message indicating the receipt of the mail contents.

- After the reception of "250 Mail Accepted" message from the server, the SMTP client issues the "Quit" command. The server replies with "221 Closing Connection" message and then closes the SMTP connection.

4.1.1 SMTP SOM Training

The MSIDS module was trained on normal values of 2304 SMTP connections obtained from MIT Lincoln Laboratory - DARPA Intrusion Detection Evaluation Project [7]. The MSIDS module generated a total of seven SOMs from the training data set with dimensions mentioned in Table 4.1. It should be noted that only 2297 SMTP connections, reported as 'complete' by the Data Processor module, were used for SOM training. Each SOM, with map dimensions of X and Y, was built in two training phases as explained in Section 3.3.3.

Table 4.1 SMTP Map Dimensions

Map #	0	1	2	3	4	5	6
X-dimension	13	13	13	14	8	4	5
Y-dimension	18	18	18	17	11	10	7

Following parameters were used for the initial SOM training phase:

- Learning rate factor α: High value of 0.9.
- Gaussian neighborhood radius: Lower of map dimensions.
- Number of iterations: Number of the training samples for the particular SOM.

Following parameters were used for the final SOM training phase:

- Learning rate factor α: Low value of 0.05.

Table 4.2 SMTP Training Data Statistics

		Map 0	Map 1	Map 2	Map 3	Map 4	Map 5	Map 6
		Cycle (0)	Cycles (0 - 1)	Cycles (0 - 3)	Cycles (0 - 7)	Cycles (0 - 15)	Cycles (0 - 31)	Cycles (0 - 63)
ASOQ	(μ)	0	11.63	21.41	286.85	248.92	179.17	548.92
	(σ)	0	1.05	1.44	536.44	463.73	467.16	451.79
ASOA	(μ)	79.23	53.97	48.82	41.67	41.59	40.39	37.78
	(σ)	15.42	14.62	9.8	4.9	3.77	4.5	0.86
L_QAIT	(μ)	-10.0	-2.77	-2.61	-2.38	-2.45	-2.51	-2.59
	(σ)	0	0.77	0.68	0.49	0.46	0.48	0.21
L_ICIT	(μ)	-1.56	-1.79	-2.01	-2.18	-2.87	-3.28	-3.29
	(σ)	0.87	0.85	0.83	0.76	0.62	0.33	0.28

- Gaussian neighborhood radius: Low value of 5.
- Number of iterations: 500 times the map dimensions (500∗X∗Y).

The SOMs, along with the mean and standard deviations for each of the 4 dimensions, are mentioned in Table 4.2. Each of the trained SOMs was evaluated by testing the training data set itself for anomalies. Table 4.3 summarizes the validation results for the maps, along with the number of training data vectors for each map. If 95.44% of the training data vectors had a winner neuron within a distance of 2σ units, the SOM was assumed to capture the essential input characteristics. It should be noted that our heuristic of 95.44% - 2σ units is barely met by the last two maps because few SMTP connections last until those buckets. In particular, since 57 of 60 and 48 of 50 SMTP connections, that last till buckets 5 and 6 respectively, have a

Table 4.3 SMTP Training Data Validation Results

Map #	0	1	2	3	4	5	6
Number of data vectors	2297	2297	2297	2297	301	60	50
Percentage (%)	100	99.52	99.91	98.56	97.69	95	96

winner neuron within 2σ units, maps 5 and 6 have corresponding training results of 95% and 96%.

4.1.2 Attack Description

Sendmail [12] is a widely deployed open-source mail transfer agent that implements the SMTP protocol. Sendmail versions 8.8.3 and 8.8.4 had a serious buffer overflow vulnerability in the processing of MIME headers. Due to insufficient bounds checking while performing limited 7 to 8 bits MIME conversions on email messages, the internal stack space of Sendmail could be overwritten. As the Sendmail program is usually run as root, the successful exploitation enabled remote users to execute arbitrary commands on the local system with root privileges. Moreover since MIME conversion is performed on final delivery, the bug may be exploited regardless of the presence of firewalls or other network boundary security measures. This vulnerability has been described in detail in a CERT advisory [2]. The attack, that exploited the buffer overflow vulnerability, was obtained offline from MIT Lincoln Laboratory data set that contains several labeled attacks. The attack works by sending a carefully crafted SMTP request with a large MIME header and overflowing the buffer on the server side. The return address on the stack is modified so as to point to the code in the MIME header to add a root account. On successful exploitation, the remote attacker can gain access on the victim with root privileges.

Table 4.4 SMTP Exploit Vector

Bucket #	ASOQ	ASOA	L_QAIT	L_ICIT
0	4474.0	575.0	-1.356	0.258
1	2239.5	300.5	-1.647	-0.043
2	1495.0	214.333	-1.814	-0.219

4.1.3 Anomaly Detection

Table 4.4 shows the four-dimensional values of the SMTP exploit vector. The *locator* program was fed with these four-dimensional vectors as input. It first normalized the vectors based on the mean and standard deviation statistics of the respective SOMs as mentioned in Table 4.2. The normalized values of the SMTP exploit vector are shown in Table 4.5. We can observe from this table and the "normal" values in Table 4.2 that the ASOQ values are significantly deviant from the mean and standard deviations values observed during the training phase. After establishing a TCP connection with the SMTP server on port 25, the attack system sends multiple malicious packets to overflow the server-side stack. These packets, perceived as questions, have ASOQ values that are highly anomalous from the observed mean and standard deviation values for the corresponding buckets. The exploit vector has ASOQ value of 4474 bytes in bucket 0, which differs from the ASOQ value, 0, observed during the training phase for map 0. Similarly, the ASOQ values for bucket 1 and 2, with standard deviation values much greater than the threshold value of 2σ units, are found to be highly anomalous from the respective mean values.

After normalizing the exploit vector, the *locator* program feeds the values into corresponding SOMs to determine the winner neurons. The distance between the normalized vectors and the winner neurons in the corresponding maps is shown in Table 4.6. As mentioned in Section 3.3.3, the distance, expressed in units of standard

Table 4.5 Normalized SMTP Exploit Vector

Bucket #	ASOQ	ASOA	L_QAIT	L_ICIT
0	4474.0	32.134	-1.356	2.085
1	2119.588	16.859	1.464	2.051
2	1022.354	16.880	1.179	2.161

Table 4.6 SMTP Exploit Winner Neuron

Bucket #	ASOQ	ASOA	L_QAIT	L_ICIT	Distance
0	4474.0	32.134	0.000	2.085	4474.110
1	2119.588	16.859	1.464	2.051	2118.391
2	1022.354	16.880	1.179	2.161	1020.425

deviation (σ) from the corresponding mean (μ), represents how abnormal the attack vector is, and should not be interpreted as the probability of occurrence using standard statistical techniques. As the distance from the winner neurons is greater than the threshold value of 2σ units, the MSIDS module classifies the SMTP connection as an attack. Moreover, detecting such anomalous behavior in early stages of the connection leads to a quicker response to intrusion. The graphs illustrating the anomalous values for ASOQ, ASOA, L_QAIT and L_ICIT for the various SOMs are shown in the Figures 4.2 to 4.7. For simplicity, we have split each four-dimensional SOM into two views. For all the graphs, the dimensions are expressed in units of standard deviations from the corresponding means.

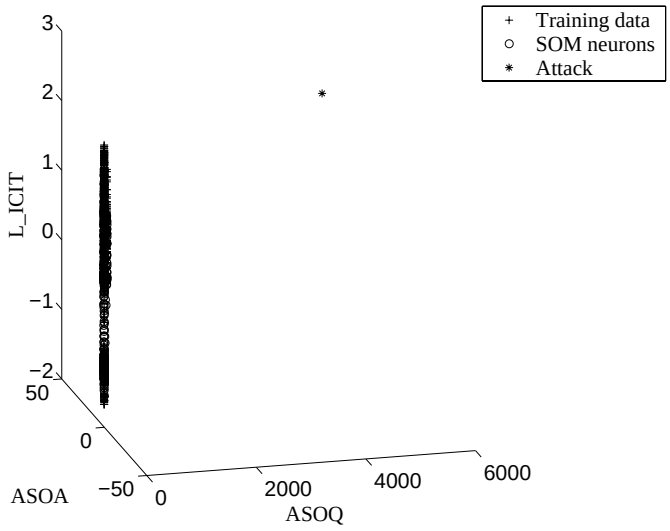

Figure 4.2. Sendmail Buffer Overflow Exploit Bucket 0, View 1: ASOQ, ASOA, and L_ICIT
The graph shows the behavior of SMTP traffic for three network parameters: ASOQ, ASOA, and L_ICIT for Bucket 0, along with the attack vector for the same bucket. All dimensions are expressed in units of standard deviations from the respective means.

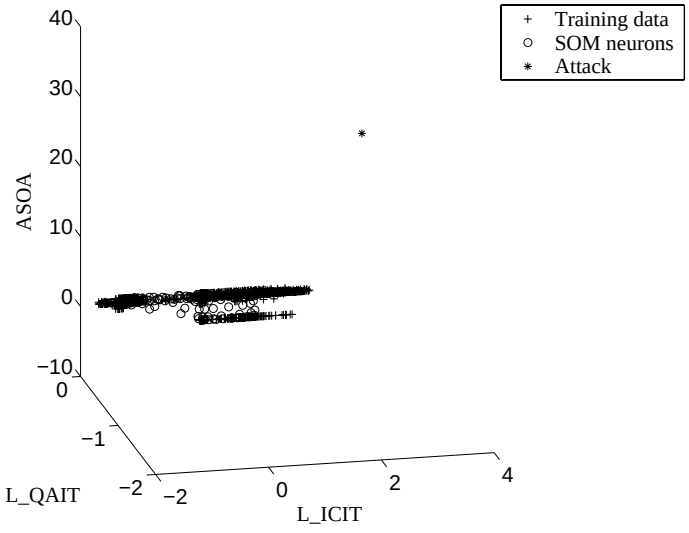

Figure 4.3. Sendmail Buffer Overflow Exploit Bucket 0, View 2: L_QAIT, L_ICIT, and ASOA

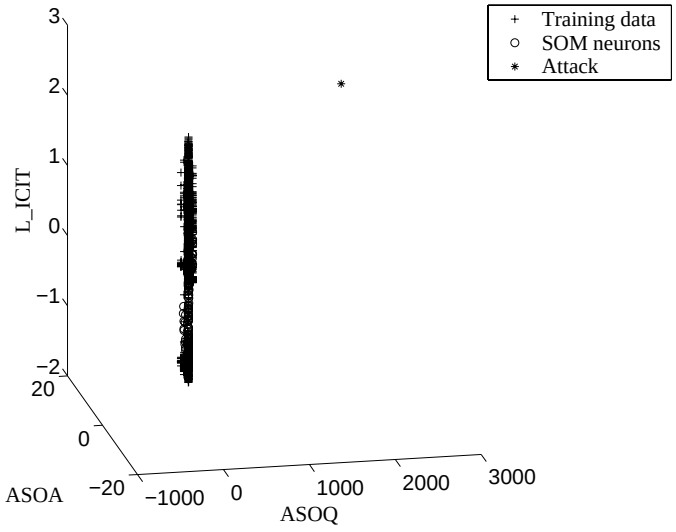

Figure 4.4. Sendmail Buffer Overflow Exploit Bucket 1, View 1: ASOQ, ASOA, and L_ICIT

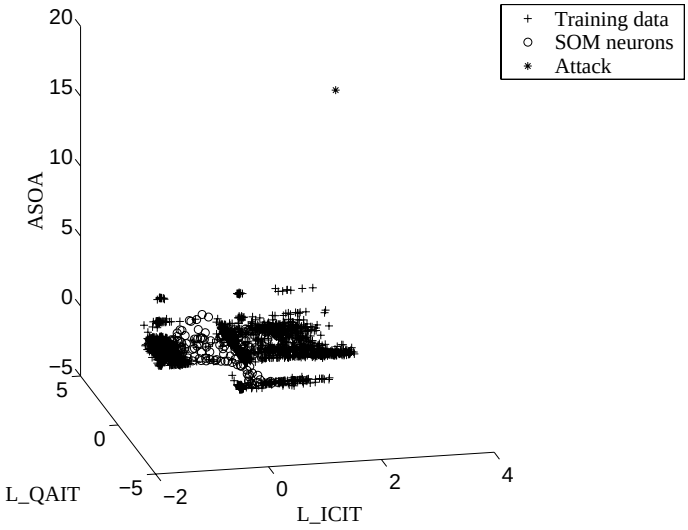

Figure 4.5. Sendmail Buffer Overflow Exploit Bucket 1, View 2: L_QAIT, L_ICIT, and ASOA

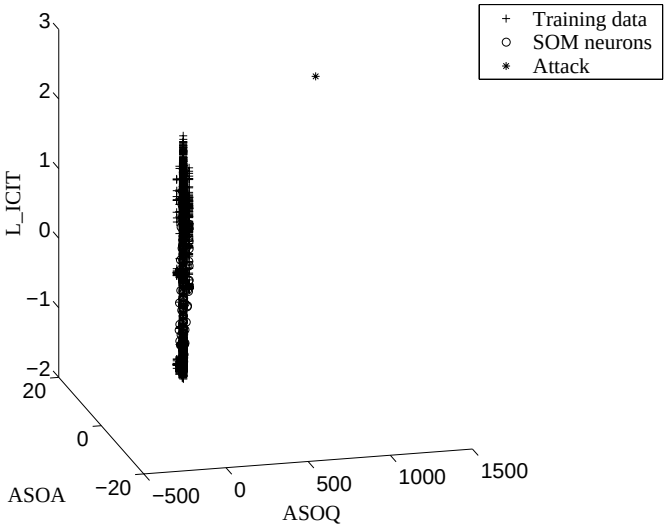

Figure 4.6. Sendmail Buffer Overflow Exploit Bucket 2, View 1: ASOQ, ASOA, and L_ICIT

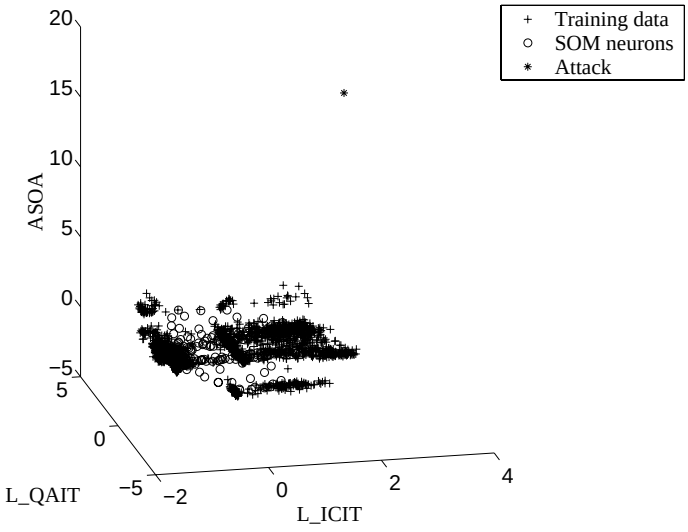

Figure 4.7. Sendmail Buffer Overflow Exploit Bucket 2, View 2: L_QAIT, L_ICIT, and ASOA

Table 4.7 ANDSOM SMTP Training Data Statistics

Dimensions	Mean	Standard Deviation
Inter	6.954	4.694
ASOQ	316.764	579.988
ASOA	36.903	9.984
L_QAIT	-1.524	0.518
L_AQIT	-1.345	0.701
DOC	3.649	75.242

4.1.4 Comparison with ANDSOM module

As explained in Section 2.3.4, the ANDSOM module for INBOUNDS detects anomalous traffic using the SOM algorithm. The module creates a single two-dimensional lattice of neurons that captures the essential characteristic patterns in the input data set. The MSIDS module, on the other hand, analyzes the time-based behavior of normal network connections for anomalous values using the SOM algorithm. It builds multiple two-dimensional maps, each capturing the behavior of the network traffic over a different period of time.

To compare the performance of the two modules, we trained the ANDSOM module with the same training data set of 2304 SMTP connections. The adjoining Table 4.7 summarizes the training data statistics used by the ANDSOM module. After the SOM training, the trained lattice of neurons was evaluated by feeding the training data set itself. The lattice modeled 98.39% of the training data set vectors with a winner neuron within 2σ distance, and cleared our 95.44% gaussian heuristic. It should be noted that the statistics generated by the ANDSOM module are essentially similar to, but not exactly same as, the statistics generated by the MSIDS module for the

Table 4.8 ANDSOM SMTP Exploit Vector

Inter	ASOQ	ASOA	L_QAIT	L_AQIT	DOC
1.465	1495	214.333	-1.648	-0.053	2.069

last map. The reason for the dissimilarity can be attributed to the different criteria used for defining the network parameters. Let us assume, for the sake of simplicity, we are interested in monitoring only one parameter, namely: ASOQ (Average Size Of Question). Suppose the MSIDS module reports five 'L' messages for a normal SMTP connection, with SOQ values of 20, 30, 20, 40, and 10 bytes respectively. Accordingly, the ASOQ (Average Size Of Question) value for the entire SMTP connection will be averaged over five cycles (i.e. 24 bytes). On the other hand, the ANDSOM module might report two 'U' messages for the entire SMTP connection; the first 'U' message reporting the ASOQ value for the first two cycles (i.e. 25 bytes) while the second 'U' message reporting the ASOQ value for the remaining three cycles (i.e. 23.33 bytes). The ASOQ value for the entire SMTP connection will then be calculated as the average of these two ASOQ values (i.e. 24.16 bytes). Thus, it is clear that the two modules use similar, but not exactly the same, statistics for generating the maps. The aforementioned scenario makes the assumption that the 'U' message and the cycle boundaries coincide. This is rarely the case as 'U' messages are reported at a fixed interval (default value of 60 seconds) throughout the lifetime of the connection, independent of the cycles, leading to even more dissimilar training statistics.

Table 4.8 shows the six-dimensional values for the Sendmail exploit vector, mentioned in Section 4.1.2, for the ANDSOM module. The *locator* program for the ANDSOM module was fed with this exploit vector. It first normalized the vector with the training statistics mentioned in Table 4.7. The normalized SMTP exploit vector is shown in Table 4.9. We can observe from this table that the ASOA value

Table 4.9 Normalized ANDSOM SMTP Exploit Vector

Inter	ASOQ	ASOA	L_QAIT	L_AQIT	DOC
-1.169	2.031	17.77	-1.648	-0.238	1.841

Table 4.10 ANDSOM SMTP Exploit Winner

Inter	ASOQ	ASOA	L_QAIT	L_AQIT	DOC	Distance
-1.169	2.031	17.77	-1.648	-0.238	1.841	16.517

is highly anomalous with a standard deviation value of 17.77 units from the mean ASOA value, classifying the connection as an intrusion. When the attack system sends multiple SMTP requests to overflow the buffer, the server responds with multiple "Command Unrecognized" messages in response. The response, perceived as answer, has an average size of 214.33 bytes that is greater than the average response message size of 36.9 bytes sent by the server. As noted in Section 4.1.3, the MSIDS module not only finds ASOA but also ASOQ values to be highly deviant from the respective mean values for the different buckets.

The winner neuron for this normalized SMTP vector, and the distance in the six-dimensional space is shown in Table 4.10. As the distance from the winner neuron is greater than the threshold value of 2σ units, the ANDSOM module classifies the SMTP connection as intrusion. For the MSIDS module, as shown in Table 4.6, the distance from the winner neuron for all the buckets is greater than 2σ units. However, classifying the SMTP connection as intrusion in bucket 0 itself results in a faster response to the intrusion.

4.2 Apache Buffer Overflow Attack

The World Wide Web (WWW, or simply the Web) is a distributed interlink of hypertext servers that operates over the Internet [19]. The Hypertext Transfer Protocol (HTTP) is a commonly used standard for information access over the Web. HTTP is a request/response protocol between clients and servers that operates at the application level. An HTTP client, such as a web browser, requests information from an HTTP server over a standard TCP/IP connection on some well-known port, typically port 80. A typical HTTP request consists of a single line containing the keyword *GET*, followed by the protocol and the required URL. The HTTP server, on the other hand, listens for such requests from HTTP clients over the well-known port. As soon as the request is fulfilled, the HTTP server closes the connection and continues to listen for requests. The aforementioned scenario depicts a typical HTTP client-server interaction as specified in HTTP version 1.0. The first version of HTTP, namely HTTP/0.9, provided raw data transfer capability over the Internet. HTTP version 1.0, specified in RFC 1945 [25], extended the protocol by allowing messages to contain information other than text, such as images and multi-media documents. The current specification, HTTP/1.1 [52], includes support for persistent connections, caching, proxies and virtual hosts. Prior to the introduction of persistent connections, HTTP/1.0 required a separate TCP connection for each requested URL. The approach lead to increased load on HTTP servers and unnecessary congestion of the Internet. With persistent connections, HTTP clients can request multiple objects over a single TCP connection to the HTTP server, thus increasing efficiency. Though mostly information flows from servers to clients, sometimes the server needs to accept data from client, for example when accepting a submitted registration form.

4.2.1 HTTP SOM Training

The MSIDS module was trained on normal values of 7369 complete HTTP connections obtained from our network. The MSIDS module generated a total of seven

SOMs from the training data set with dimensions mentioned in Table 4.11. Each SOM, with map dimensions of X and Y, was built in two training phases as explained in Section 3.3.3. SOM training was accomplished with the same parameters as mentioned in Section 4.1.1.

Table 4.11 HTTP Map Dimensions

Map #	0	1	2	3	4	5	6
X-dimension	19	12	10	9	7	5	3
Y-dimension	23	14	14	10	8	5	3

The SOMs, along with the mean and standard deviations for each of the 4 dimensions, are mentioned in the adjoining Table 4.12. Each of the trained SOMs was evaluated by feeding the training data set itself. Table 4.13 summarizes the validation results for the maps, along with the number of training data vectors for each map. If 95.44% of the training data vectors had a winner neuron within a distance of 2σ units, the SOM was assumed to capture the essential input characteristics. It should be noted that since only 26 of 29 HTTP connections, that last till bucket 5, have a winner neuron within 2σ units, map 5 does not meet our heuristic of 95.44% - 2σ units. On the other hand, as both of the HTTP connections, that last till bucket 6, have a winner neuron within 2σ units, the corresponding map shows 100% success rate.

4.2.2 Attack Description

The *Apache HTTP Server* is an open-source web server that implements a number of protocols, including HTTP/1.1 [1]. A project of the Apache Software Foundation, Apache aims at providing a secure, efficient and extensible server for the various

Table 4.12 HTTP Training Data Statistics

		Map 0	Map 1	Map 2	Map 3	Map 4	Map 5	Map 6
		Cycle (0)	Cycles (0-1)	Cycles (0-3)	Cycles (0-7)	Cycles (0-15)	Cycles (0-31)	Cycles (0-63)
ASOQ	(μ)	706.675	545.94	554.107	578.271	629.05	625.94	628.84
	(σ)	2512.62	264.001	257.697	287.042	333.73	397.92	193.07
ASOA	(μ)	11037.3	11763.7	8785.4	13820.6	4112.6	2832.2	2104
	(σ)	121800	83655.3	43621.9	136012	7416.7	1858.4	1994
L_QAIT	(μ)	-1.265	-1.422	-1.393	-1.346	-1.331	-1.019	-0.829
	(σ)	0.786	0.738	0.734	0.748	0.779	0.616	0.336
L_ICIT	(μ)	-4.253	-0.391	-0.163	-0.094	0.058	-0.056	0.297
	(σ)	3.591	1.157	1.068	0.995	0.838	0.661	0.154

modern operating systems. As of October 2004, more than 67% of the web sites on the Internet run on Apache, making it the most popular web server on the Internet [8]. As mentioned previously, data occasionally flows from clients to server when a client submits information via *forms*. In cases where the HTTP client does not know beforehand how much data will be uploaded, it requests a 'Chunked Encoded' transfer. If supported by the HTTP server, the client organizes the data into *chunks* as it is generated. The server is informed about the chunk size and data is submitted in chunks by the HTTP client. Apache includes support for chunk-encoded data according to the HTTP/1.1 specification.

A security vulnerability in the handling of certain chunk-encoded HTTP requests was reported in June 2002 [28]. Apache web server versions 1.2.2 and above, 1.3 through 1.3.24, and versions 2.0 through 2.0.36 were affected by this vulnerability. Possibly due to improper (signed) interpretation of an unsigned integer value, the

Table 4.13 HTTP Training Data Validation Results

Map #	0	1	2	3	4	5	6
Number of data vectors	7369	1132	744	352	128	29	2
Percentage (%)	99.09	98.76	98.79	99.43	98.44	89.66	100

vulnerable web servers incorrectly calculated the required buffer sizes when processing requests coded with the 'Chunked Encoding' mechanism [11]. The vulnerability allows a remote attacker to execute arbitrary code on the remote system or lead to denial-of-service attacks. The exploit for this vulnerability is publicly available on the Web [10].

The exploit works by sending a large amount of data (shell code and the return address to be placed on the stack) in a single chunk to the web server, and overflowing the buffer space allocated to that chunk. When the vulnerability is exploited, the return address causes the next instruction to point to the shell code on the stack. To increase the likelihood of the exploit working, the code performs a brute-force attack wherein several connections to the web server are attempted with guessed return addresses.

4.2.3 Anomaly Detection

Table 4.14 shows the four dimensional values of the HTTP exploit vector. The *locator* program was fed with these four dimensional vectors as input. It first normalized the vectors based on the mean and standard deviation statistics of the respective SOMs as mentioned in Table 4.12.

The normalized values of the HTTP exploit vector are shown in Table 4.15. We can observe from this table and the "normal" values in the Table 4.12 that the ASOQ values are significantly deviant from the corresponding mean and standard deviation

Table 4.14 HTTP Exploit Vector

Bucket #	ASOQ	ASOA	L_QAIT	L_ICIT
0	29896	4	-1.972	-3.382
1	14991	81	-2.146	0.927
2	7507.25	9262.5	-2.235	1.035
3	3762.25	7096.25	-2.312	0.825
4	3344.778	6307.778	-2.359	0.774

values observed during the training phase. The attack system sends malicious packets to the server containing the shell code and return address, attempting to overwrite the allocated buffer space. These packets, perceived as questions by the Data Processor module, have ASOQ values that are highly anomalous from the observed mean and standard deviation values for the corresponding buckets. The exploit vector has ASOQ value of 29896 bytes in bucket 0, that differs from the ASOQ value of 706 bytes observed during the training phase for map 0. Similarly, the ASOQ values for subsequent buckets, with standard deviation values much greater than the threshold value of 2σ units, are found to be highly anomalous from the respective mean values.

After normalizing the exploit vector, the *locator* program feeds the values into corresponding SOMs to determine the winner neurons. The distance between the normalized vectors and the winner neurons in the corresponding maps is shown in Table 4.16. As previously mentioned, the distance, expressed in units of standard deviation (σ) from the corresponding mean (μ), is a measure of how abnormal the attack vector is, and should not be converted into Gaussian probability using traditional statistical techniques. As the distance from the winner neurons is greater than the threshold value of 2σ units, the MSIDS module classifies the HTTP connection as an attack. Moreover, the attack vector exhibits such behavior in the early stages

Table 4.15 Normalized HTTP Exploit Vector

Bucket #	ASOQ	ASOA	L_QAIT	L_ICIT
0	11.617	-0.09	-0.898	0.242
1	54.715	-0.139	-0.979	1.138
2	26.981	0.01	-1.145	1.121
3	11.092	-0.049	-1.289	0.924
4	8.137	0.295	-1.317	0.853

of the connection, leading to a faster intrusion detection. The graphs illustrating the anomalous values for ASOQ, ASOA, L_QAIT and L_ICIT for the various SOMs are shown in the Figures 4.8 to 4.13. For simplicity, we have split each four-dimensional SOM into two views. For all the graphs, the dimensions are expressed in units of standard deviations from the corresponding means.

Table 4.16 HTTP Exploit Winner Neuron

Bucket #	ASOQ	ASOA	L_QAIT	L_ICIT	Distance
0	11.617	-0.09	-0.898	0.242	2.895
1	54.715	-0.139	-0.979	1.138	51.279
2	26.981	0.01	-1.145	1.121	23.955
3	11.092	-0.049	-1.289	0.924	8.658
4	8.137	0.295	-1.317	0.853	6.401

In Figure 4.8, we observe that the training data vectors, indicated by plus symbols

(+), towards the extreme right represent instances of HTTP connections with large amounts of data exchanged in the initial stages of connection. We notice similar behavior in Figure 4.9 with large ASOA values for some HTTP connections.

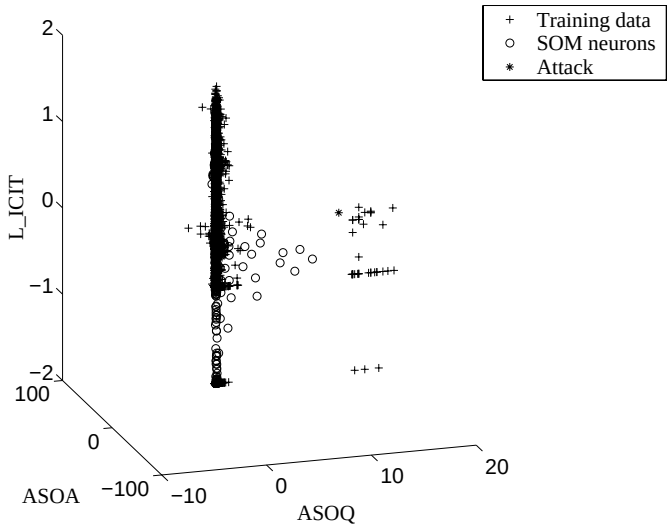

Figure 4.8. Apache Buffer Overflow Exploit Bucket 0, View 1: ASOQ, ASOA, and L_ICIT
The graph shows the behavior of HTTP traffic for three network parameters: ASOQ, ASOA, and L_ICIT for Bucket 0, along with the attack vector for the same. All dimensions are expressed in units of standard deviations from the respective means.

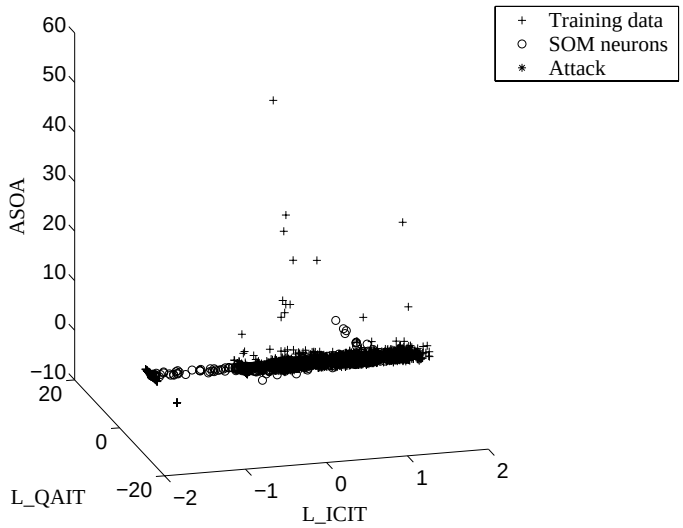

Figure 4.9. Apache Buffer Overflow Exploit Bucket 0, View 2: L_QAIT, L_ICIT, and ASOA
The attack vector has approximate co-ordinate values of (-0.8, 0.2, -0.09) for L_QAIT, L_ICIT, and ASOA dimensions respectively.

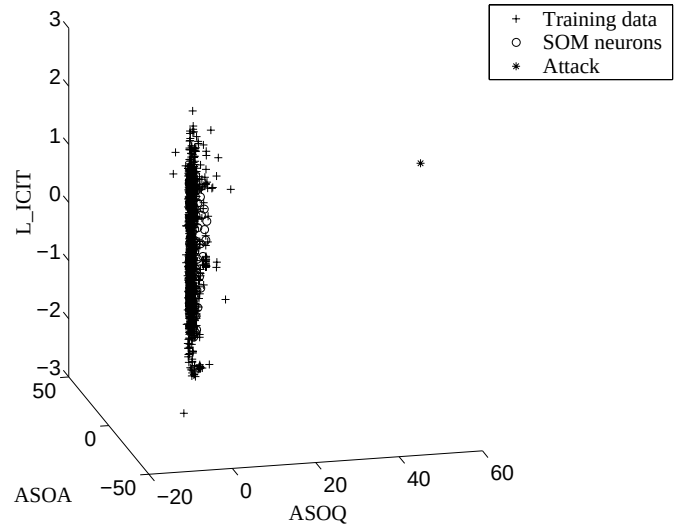

Figure 4.10. Apache Buffer Overflow Exploit Bucket 1, View 1: ASOQ, ASOA, and L_ICIT

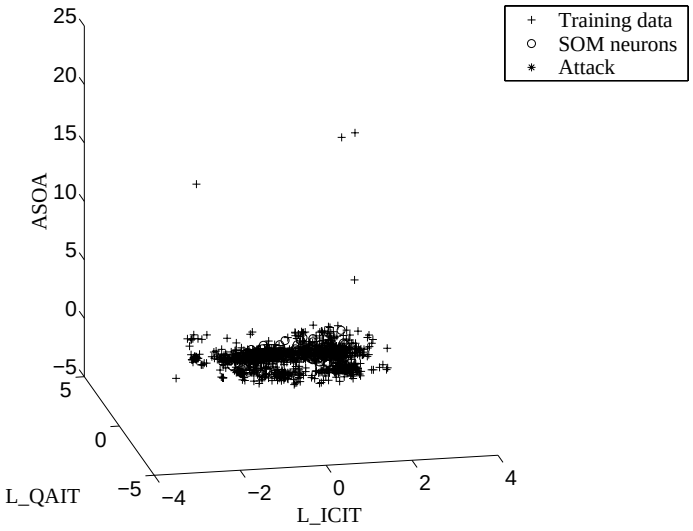

Figure 4.11. Apache Buffer Overflow Exploit Bucket 1, View 2: L_QAIT, L_ICIT, and ASOA
The attack vector has approximate co-ordinate values of (-0.9, 1.1, -0.1) for L_QAIT, L_ICIT, and ASOA dimensions respectively.

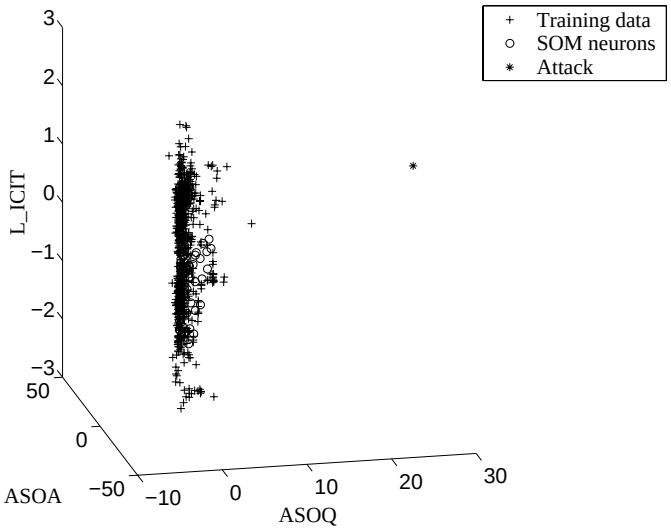

Figure 4.12. Apache Buffer Overflow Exploit Bucket 2, View 1: ASOQ, ASOA, and L_ICIT

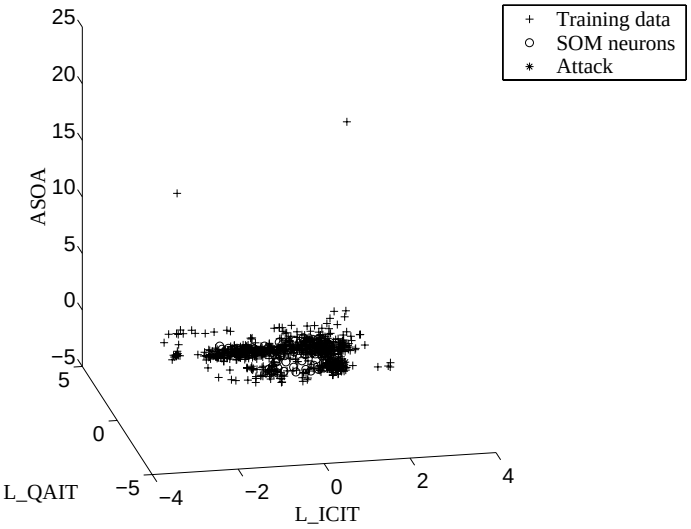

Figure 4.13. Apache Buffer Overflow Exploit Bucket 2, View 2: L_QAIT, L_ICIT, and ASOA
The attack vector has approximate co-ordinate values of (-1.1, 1.1, 0.01) for L_QAIT, L_ICIT, and ASOA dimensions respectively.

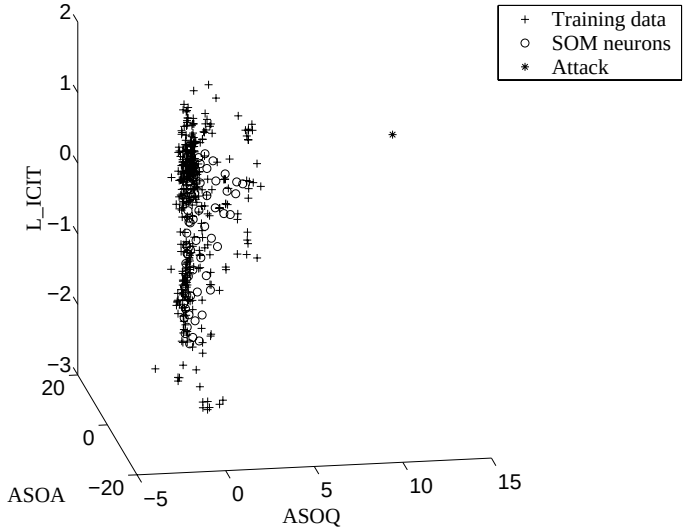

Figure 4.14. Apache Buffer Overflow Exploit Bucket 3, View 1: ASOQ, ASOA, and L_ICIT

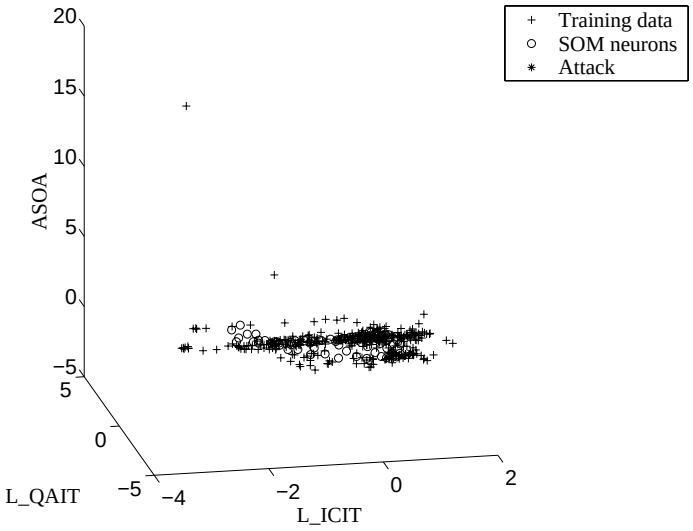

Figure 4.15. Apache Buffer Overflow Exploit Bucket 3, View 2: L_QAIT, L_ICIT, and ASOA
The attack vector has approximate co-ordinate values of (-1.2, 0.9, -0.04) for L_QAIT, L_ICIT, and ASOA dimensions respectively.

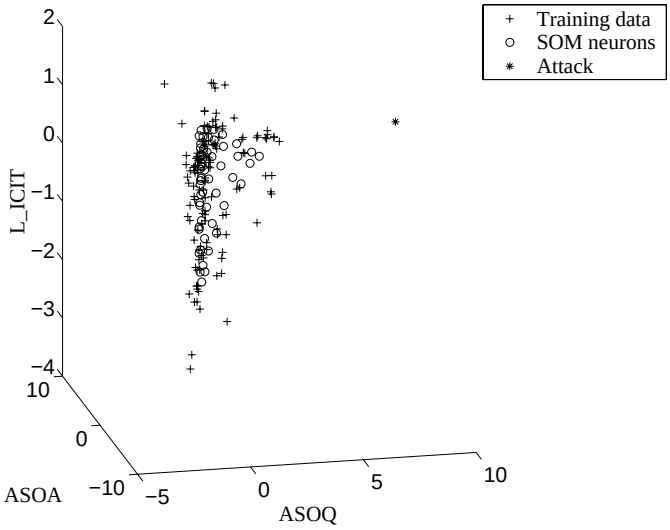

Figure 4.16. Apache Buffer Overflow Exploit Bucket 4, View 1: ASOQ, ASOA, and L_ICIT

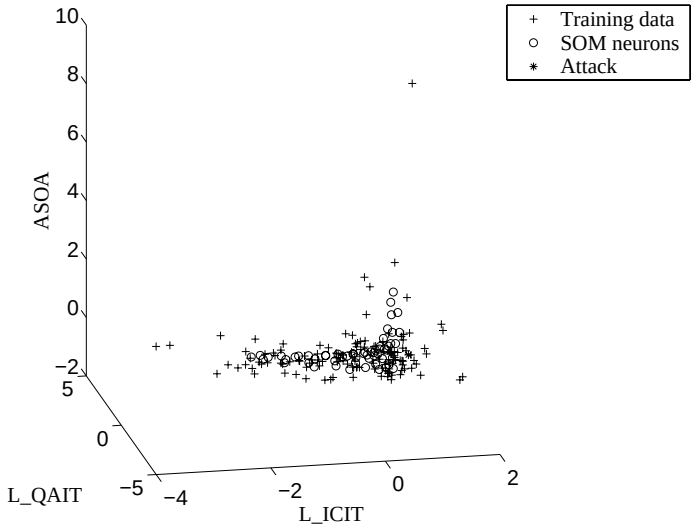

Figure 4.17. Apache Buffer Overflow Exploit Bucket 4, View 2: L_QAIT, L_ICIT, and ASOA
The attack vector has approximate co-ordinate values of (-1.3, 0.8, 0.2) for L_QAIT, L_ICIT, and ASOA dimensions respectively.

5. Conclusion

The MSIDS module detects network intrusions by analyzing the time-based behavior of normal network connections using multiple Self-Organizing Maps. This chapter provides a summary of the approach, followed by a description of its advantages and disadvantages. Lastly, we present some recommendations for future work.

5.1 Summary

The Self-Organizing Map (SOM) algorithm maps the data points from a complex high-dimensional input space onto a low-dimensional output space, thereby facilitating visualization of high-dimensional data. The ANDSOM module, based on the SOM algorithm, introduced a new approach to intrusion detection to the INBOUNDS system. The MSIDS module extends the approach by examining the time-based behavior of network traffic for anomalous characteristics.

The MSIDS module identifies four parameters to characterize the network connections: SOQ (Size of Question), SOA (Size Of Answer), QAIT (Question-Answer Idle Time) and ICIT (Inter-Cycle Idle Time). To analyze the constituent parts for anomalous values, every network connection is broken down into individual 'cycles'. The current implementation of MSIDS 'exponentially' groups these cycles into 'buckets' and reports the four-dimensional vectors <ASOQ, ASOQ, L_QAIT, L_ICIT> for each bucket. Normalized four-dimensional vectors are then used to create multiple SOMs, one for every bucket. SOM training is accomplished in two phases: an initial learning phase with a high learning rate and fewer iterations, and a final fine-tuning phase with a low learning rate and higher iterations. It should be noted that each SOM is trained using the normalized four-dimensional vectors for that bucket only. In other

words, each trained SOM essentially represents the behavior of network connections over that period of time.

During real-time operation, the MSIDS module generates four-dimensional vectors representing the observed network connection. Each normalized four-dimensional vector is then fed in parallel to all the neurons of the corresponding trained SOM. For each input vector fed into the trained SOM, a winner neuron is determined. If the distance between the input vector and the winner neuron for any SOM lattice is greater than a certain threshold value, the connection is classified as an intrusion.

In order to validate this new approach, the MSIDS module was trained for different classes of network traffic, such as SMTP and HTTP. A buffer overflow attack exploiting a vulnerability in Sendmail program version 8.8.3 was successfully classified as an intrusion by the MSIDS module. The attack connection exhibited ASOQ values that were significantly anomalous from the corresponding normal values. Similarly, the SOMs built for normal HTTP traffic classified an Apache/OpenBSD exploit as an intrusion based on the anomalous size of ASOQ value. The exploit, in order to overflow the server-side buffer, sent multiple large HTTP requests to the victim, leading to high ASOQ value.

5.2 Advantages And Disadvantages

The ANDSOM module for INBOUNDS analyzed entire network connections for anomalous values. A network connection, with normal values for the entire connection but with anomalous values over constituent time intervals, may be classified as 'normal', thus creating a false-negative. The MSIDS module, on the other hand, develops multiple SOMs that capture the behavior of normal network traffic over different time intervals, leading to refined analysis. As a result, network connections, with normal behavior over the entire connection but anomalous behavior over different time intervals, will be classified as intrusions.

The MSIDS module considers a network connection as an intrusion if the winner

neuron for any SOM lattice is greater than 2σ units, where σ stands for standard deviation. Thus, if a network connection displays abnormal behavior for the monitored parameters early on, the MSIDS module promptly raises an intrusion alert, leading to a near real-time response. The ANDSOM module, on the other hand, analyzes a network connection only after it is reported closed by the Data Processor module. In other words, a connection lasting for an arbitrarily long period of time would be considered for analysis only after it closes.

The MSIDS module utilizes the SOM algorithm to capture the time-based behavior of normal network connections. However, the SOM algorithm tends to capture the bulk behavior of network traffic, disregarding the corner cases of normal network behavior. The SOM training, for MSIDS module, is regarded as complete if 95.44% of the training data vectors have a winner neuron within a certain threshold value (2σ units). In other words, the remaining 4.56% of the training data vectors will themselves be classified as intrusions, leading to significant number of false-positives.

The inherent difficulty with an anomaly-based IDS is to find the right set of parameters to characterize normal network behavior. The anomaly-based MSIDS module identifies four such network parameters: ASOQ, ASOA, QAIT, and ICIT. The module raises an intrusion alert if a network connection displays anomalous behavior only for these monitored parameters. In other words, attacks that exhibit normal behavior for these four specific parameters will go unnoticed, leading to false-negatives.

5.3 Future Work

The current implementation of MSIDS module 'exponentially' stores the 'L' messages in a list of buckets. As mentioned previously, the rationale for grouping multiple 'L' messages is twofold: better characterization of normal network behavior, and potentially reduction in the number of false-positives. Tables 4.3 and 4.13 illustrate that the trained SOMs appearing towards the end barely meet our validation heuristic,

leading to increased number of false-positives. As a result, it would be interesting to construct various other groupings of 'L' messages and evaluate their performance. Moreover, it would be intriguing to find groupings that are suitable for certain protocols, leading to better characterization of network traffic and hence better intrusion detection.

The MSIDS module utilizes the SOM algorithm to capture the characteristic patterns in the input data set. The SOM training phase uses the heuristic of 2σ units and 95.44% distribution for validation purpose. Although this has given satisfactory performance, constructing and validating SOMs with different values may yield better results. Furthermore, the MSIDS module utilizes the basic SOM algorithm with a gaussian distribution and hexagonal map topology. It would be interesting to evaluate the performance of the module with different training parameters.

BIBLIOGRAPHY

[1] Apache HTTP Server Project. URL: http://httpd.apache.org/.

[2] 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.

[3] CyberCop. URL: http://www.ngc.com/product_info/cybercop/ccdata/ccdata1.html.

[4] EtherApe. URL: http://etherape.sourceforge.net.

[5] Ethereal. URL: http://www.ethereal.com.

[6] Finger Bomb. URL: http://securescannx.vigilante.com/tc/13109.

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

[8] Netcraft Web Server Survey. URL: http://news.netcraft.com/.

[9] Neural Networks Tool - NeNet. URL: http://koti.mbnet.fi/ phodju/nenet/Nenet/General.html.

[10] Packet Storm Security: OpenBSD/X86 Apache Chunked-Encoding Vulnerability. URL: http://packetstormsecurity.org/0206-exploits/apache-scalp.c.

[11] Security Focus: Apache Chunked-Encoding Memory Corruption Vulnerability. URL: http://www.securityfocus.com/bid/5033/discussion.

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

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

[14] SOFM package. URL: http://nn.cs.utexas.edu/pages/software/software.html.

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

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

[17] Tcpdump. URL: http://www.tcpdump.org.

[18] The Comprehensive R Archive Network. URL: http://cran.r-project.org/src/contrib/Descriptions/som/html.

[19] World Wide Web. URL: http://en.wikipedia.org/wiki/WWW.

[20] xsom. URL: http://fuzzy.cs.uni-magdeburg.de/ borgelt/doc/somd/.

[21] Internet Domain Survey, Internet Software Consortium, January 2004. URL: http://www.isc.org.

[22] Alirezaie, J., Jernigan, M. E., and Nahmias, C. Automatic Segmentation of MR Images Using Self-organizing Feature Mapping and Neural Networks. *Proceedings of the SPIE—The International Society for Optical Engineering 3034*, pt. 1–2 (1997), 138–49. (Medical Imaging 1997: Image Processing Conf. Date: 25–28 Feb. 1997 Conf. Loc: Newport Beach, CA, USA Conf. Sponsor: SPIE).

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

[24] Balupari, R. Real-Time Network-Based Anomaly Intrusion Detection. Master's thesis, Ohio University, 2002.

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

[26] Blackmore, J., and Miikkulainen, R. Incremental Grid Growing: Encoding High-Dimensional Structure into a Two-Dimensional Feature Map. In *Proc. ICNN'93, International Conference on Neural Networks* (Piscataway, NJ, 1993), vol. I, IEEE Service Center, pp. 450–455.

[27] Blanton, E. Tcpurify: TCP Packet Sniffer, Sept. 2002. URL: http://irg.cs.ohiou.edu/ eblanton/tcpurify.

[28] CERT. Apache Web Server Chunk Handling Vulnerability, CERT Advisory. URL: http://www.cert.org/advisories/CA-2002-17.html.

[29] D. Anderson, T. Frivold and A. Valdes. Next-Generation Intrusion Detection Expert System. Tech. rep., SRI International, 1995.

[30] Dittenbach, M., Merkl, D., and Rauber, A. The Growing Hierarchical Self-Organizing Map. In *Proc of the International Joint Conference on Neural Networks (IJCNN 2000)* (Como, Italy, July 24. – 27. 2000), S. Amari, C. L. Giles, M. Gori, and V. Puri, Eds., vol. VI, IEEE Computer Society, pp. 15 – 19.

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

[32] G. White and V. Pooch. Cooperating Security Managers: Distributed Intrusion Detection Systems. *Computer & Security*.

[33] Helge Ritter, T. M., and Schulten, K. *Neural Computation and Self-Organizing Maps*. Addison-Wesley Publishing Company, 1992.

[34] Honkela, T., Kaski, S., Lagus, K., and Kohonen, T. WEBSOM—Self-Organizing Maps of Document Collections. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6* (Espoo, Finland, 1997), Helsinki University of Technology, Neural Networks Research Centre, pp. 310–315.

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

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

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

[38] Kurimo, M. Self-Organization in Mixture Densities of HMM Based Speech Recognition. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'98)* (Bruges, Belgium, 1998), G. Deboeck and T. Kohonen, Eds., Springer-Verlag, pp. 237–242.

[39] Laaksonen, J., Koskela, M., and Oja, E. PicSOM—A Framework for Content-Based Image Database Retrieval using Self-Organizing Maps. In *Proc. of 11th Scandinavian Conference on Image Analysis (SCIA'99), Kangerlussuaq, Greenland, June 7–11* (1999), pp. 151–156.

[40] Lunt, T. F., Jagannathan, R., Lee, R., Listgarten, S., Edwards, D. L., Javitz, H. S., and Valdes, A. IDES: The Enhanced Prototype - A Real-Time Intrusion-Detection Expert System. Tech. Rep. SRI-CSL-88-12, Computer Science Laboratory, SRI International, Menlo Park, CA, October 1988.

[41] Manikantan Ramadas, Shawn Ostermann and Brett Tjaden. Detecting Anomalous Network Traffic with Self-Organizing Maps. In *Recent Advances in Intrusion Detection* (Sept. 2003), Springer-verlag.

[42] Michael M. Sebring, Eric Shellhouse, Mary E. Hanna and R. Alan Whitehurst. Expert Systems in Intrusion Detection: A Case Study. In *Proceedings of the 11th National Computer Security Conference* (1988).

[43] Miikkulainen, R. Script Recognition with Hierarchical Feature Maps. *Connection Science 2*, 1&2 (1990), 83–101.

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

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

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

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

[48] Ostermann, S. tcptrace: TCP dump file analysis tool, Jan. 1996. http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html.

[49] P. Porras and P. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. 20th National Information Systems Security Conference.

[50] Pearl, J. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

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

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

[53] Rauber, A., and Merkl, D. The SOMLib Digital Library System. In *European Conference on Digital Libraries* (1999), pp. 323–342.

[54] Rauber, A., Pampalk, E., and Merkl, D. The SOM-enhanced JukeBox: Organization and Visualization of Music Collections based on Perceptual Models. *Journal of New Music Research* (2003).

[55] Rebecca Bace and Peter Mell. Intrusion Detection Systems. Tech. rep., National Institute of Standards and Technology, July 2004.

[56] Rhodes, B.C., M. J. A., and Cannady, J. D. Multiple Self-Organizing Maps for Intrusion Detection. In *National Proceedings of the 23rd National Information Systems* (Baltimore, MD, 2000).

[57] Ryan, J., Lin, M.-J., and Miikkulainen, R. Intrusion Detection with Neural Networks. In *Proceedings of the 1997 conference on Advances in neural information processing systems 10* (1998), MIT Press, pp. 943–949.

[58] S.Smaha. Haystack: An Intrusion Detection System. IEEE Computer Society Press.

[59] Teuvo Kohonen, Erkki Oja, Olli Simula, Ari Visa, and Jari Kangas. Engineering Applications of the Self-Organizing Map. In *Proceedings of the IEEE, vol. 84, No. 10*, p. 1358.

[60] Todd Heberlein, Gihan Dias, Karl Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A Network Security Monitor. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, p. 296.

[61] Yellapragada, R. Probabilistic Model for Detecting Network traffic anomalies. Master's thesis, Ohio University, 2004.