8-1-2011

A SOM+ Diagnostic System for Network Intrusion Detection

Chester Louis Langin
Southern Illinois University Carbondale, clangin@siu.edu

A SOM+ DIAGNOSTIC SYSTEM FOR NETWORK INTRUSION DETECTION

by

Chester L. Langin

B.S., Southern Illinois University Carbondale, 1974
M.S., Southern Illinois University Carbondale, 2003

A Dissertation
Submitted in Partial Fulfillment of the Requirements for the
Doctor of Philosophy Degree

Department of Computer Science
in the Graduate School
Southern Illinois University Carbondale

August 2011

DISSERTATION APPROVAL

A SOM+ DIAGNOSTIC SYSTEM FOR NETWORK INTRUSION DETECTION

By

Chester L. Langin

A Dissertation Submitted in Partial

Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in the field of Computer Science

Approved by:

Shahram Rahimi, Chair

Ajith Abraham

Henry Hexmoor

Mohammad R. Sayeh

Michael S. Wainer

Graduate School
Southern Illinois University Carbondale
July, 2011

AN ABSTRACT OF THE DISSERTATION OF

CHESTER L. LANGIN, for the Doctor of Philosophy degree in COMPUTER SCIENCE, presented on July 7, 2011, at Southern Illinois University Carbondale.

TITLE: A SOM+ DIAGNOSTIC SYSTEM FOR NETWORK INTRUSION DETECTION

MAJOR PROFESSOR: Dr. Shahram Rahimi

This research created a new theoretical Soft Computing (SC) hybridized network intrusion detection diagnostic system including complex hybridization of a 3D full color Self-Organizing Map (SOM), Artificial Immune System Danger Theory (AISDT), and a Fuzzy Inference System (FIS). This SOM+ diagnostic archetype includes newly defined intrusion types to facilitate diagnostic analysis, a descriptive computational model, and an Invisible Mobile Network Bridge (IMNB) to collect data, while maintaining compatibility with traditional packet analysis. This system is modular, multitaskable, scalable, intuitive, adaptable to quickly changing scenarios, and uses relatively few resources.

ACKNOWLEDGMENTS

I appreciate the time my committee members spent listening to my presentations and reviewing my work. Thanks to Dr. Henry Hexmoor for helping me to get started in the Ph.D. program, and to Dr. Mohammad R. Sayeh who taught me Self-Organizing Maps (SOM). Thanks to Dr. Michael S. Wainer for the continued encouragement and ideas and for providing the expertise to create 3D graphics from SOM output. Thanks to Dr. Ajith Abraham whose many related papers provided a foundation for much of my work and who agreed to be on my committee from around the world, even though we had not previously met each other. Thanks to fellow students Purvag Patel and Feng Yu for listening to and critiquing my rationale and for making major original contributions to the LLNIDS Types of Intrusion Detection and to the LLNIDS Computational Model. I appreciate the many lunches with colleague Yung-Chuan Lee where theoretical and technical strategies were discussed. Most of all, thanks to Dr. Shahram Rahimi, my committee chair, who led me through many ideas, directed me towards Artificial Immune System Danger Theory, and kept me focused. Any mistakes, of course, are my own.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1-- INTRODUCTION

The widespread need for improved network intrusion detection is clear. A recent United States Air Force manual, for example, emphasizes, “The United States is vulnerable to cyberspace attacks by relentless adversaries attempting to infiltrate our networks at work and at home – millions of times a day, 24/7.” [1] (US Air Force 2010) Computer intrusions are pervasive in our networking and have easily cost billions of dollars of resources in recent years considering the lost time of users continually updating operating systems and other software, the cost of hiring information security teams, the lost time of rebuilding computer operating systems after they have become infected, and the cost of the recovery processes of individuals whose identities have been stolen.

Information security is a complex and inexact process involving devious malignant people with a broad range of skills from technical to social engineering. Intrusion detection involves subterfuge and many deceptions and spoofs where things are not as they appear to be. Even security officers can be disgruntled employees posing insider threats. This complexity and ambiguity involved with intrusion detection leads to Soft Computing for solutions.

This research represents six years of studying network intrusion detection, progressing from traditional packet and log analysis to general Soft Computing methods, and, then, specifically, to create, implement, test, and evaluate a new theoretical Soft Computing (SC) hybridized network intrusion detection diagnostic system to project Self Organizing Map (SOM) output to a 3D full color visual display and process this output through a second layer Fuzzy Inference System

(FIS) to better interpret the results. Artificial Immune System Danger Theory (AISDT) was studied to see if this could be hybridized with the SOM. This SOM+ diagnostic archetype included an Invisible Mobile Network Bridge (IMNB) to collect data, a descriptive intrusion detection computational model to describe the methodology, and newly defined intrusion types to facilitate diagnostic analysis of results, while maintaining compatibility with traditional packet analysis. The goals for this system included being modular, multitaskable, scalable, intuitive, and adaptable to quickly changing scenarios while using relatively few resources.

While Soft Computing methods have been researched in intrusion detection, several theoretical ideas of this research are new: the 3D full color SOM visual display with the hybridized FIS and AISDT, the Invisible Mobile Network Bridge (IMNB) for data collection and monitoring, the intrusion detection computational model, and the newly defined intrusion types. All of these novel theories together form a cohesive new diagnostic system for intrusion detection. This ID diagnostic system is based on a medical center metaphor: medical diagnosis is based on a multitude of possible analytical tests such as x-rays, CAT scans, urine tests, and blood tests. Various medical tests are often used and a diagnosis is typically based on a fusion of multiple test results. Likewise with network intrusion detection, multiple analytical methods are needed with analytical methods to fuse the results.

This research originated with creation of ANNaBell, a SOM Intrusion Detection System (IDS) placed into production which has successfully detected feral malware, including Storm Worm bot infections. A SOM is a type of Artificial

Neural Network (ANN), and ANNaBell is named for being an ANN that rings a bell (sends an alert) when malware is detected. ANNaBell discovered an infected computer on March 29, 2008, being the first known time that a self-trained computational intelligence has discovered previously unknown feral malicious software. ANNaBell first discovered a Storm Worm infected computer on August 12, 2008, and was still in production as this was written approximately three years later.

This research redoubles capabilities of computational intelligence for intrusion detection for malware not previously known. Traditional packet analysis intrusion detection is stymied by the continuously evolving command and control network traffic. This diagnostic model assists in overcoming this barrier by using Internet traffic recorded by an Invisible Mobile Network Bridge (IMNB) to indicate locally infected computers. This research concerns the analysis of multidimensional data. One of the advantages of this methodology is the representation of multidimensional data into a smaller dimensional space. A meta-hexagonal map mitigates the so-called curse of dimensionality by labeling complex data areas with simple explanatory names such as *green zone* and *red zone* to help further visualize the data.

Chapter 2 provides background, review of the literature, and state of the art information for initial components of this research including types of intrusion detection, data preparation, feature reduction, clustering and classification, the intrusion detection problem, and the use of Soft Computing methods in intrusion detection. Chapter 3 covers the early methodology and development of

components of this research including Local Landline Network Intrusion Detection System (LLNIDS) Types of Intrusion Detection, the LLNIDS Computational Model, and the design of the LLNIDS Hybrid used in this research including the Invisible Mobile Network Bridge (IMNB), the SOM component, the FIS component, the Svensson and Josang (SJ) Fusion component, and the AIS Danger Theory component. Chapter 4 continues with the implementation and results of the hybridized SOM/FIS/AISDT aspects of the system, including new dynamic procedures. Chapter 5 discusses the significance of this research, and Chapter 6 is the conclusion.

CHAPTER 2—BACKGROUND AND LITERATURE

This research proposes a nearly complete overhaul of how intrusion detection research is viewed and accomplished from types, to a data computational model, Soft Computing methods, and an overall diagnostic system. This chapter provides background, review of the literature, and state of the art information for each of the initial issues of this research, including types of intrusion detection, data preparation, feature reduction, clustering and classification, the intrusion detection problem, and Soft Computing intrusion detection methods including Self-Organizing Maps (SOM), Fuzzy inference Systems (FIS), and Artificial Immune Systems Danger Theory (AISDT). No other known work covers a similar comprehensive diagnostic system for intrusion detection.

TYPES OF INTRUSION DETECTION

Intrusion detection is the process of identifying and responding to malicious activity targeted at computing and networking sources [2]. Over the years, types of intrusion detection have been labeled in various linguistic terms, with often vague or overlapping meanings. Not all researchers have used the same labels with the same meanings. To demonstrate the need for consistent labeling of intrusion types, previous types of intrusion detection are listed below in order to show the variety of types of labeling that have been used in the past.

Denning [3] in 1986 referred to intrusion detection methods which included profiles, anomalies, and rules. Her profiling included metrics and statistical

models. She referred to *misuse* in terms of insiders who misused privileges.

Young in 1987 [4] defined two types of monitors: appearance monitors and behavior monitors, the first performing static analysis of systems to detect anomalies and the second examining behavior.

Lunt [5] in 1988 referred to the misuse of insiders; the finding of abnormal behavior by determining departures from historically established norms of behavior; *a priori* rules; and using expert system technology to codify rules obtained from system security officers. A year later, in 1989, Lunt mentioned knowledge-based, statistical, and rule-based intrusion detection. In 1993, she referred to model-based reasoning [6].

Vaccaro and Liepins [7] in 1989 stated that misuse manifests itself as anomalous behavior. Hellman, Liepins, and Richards [8] in 1992 stated that computer use is either normal or misuse. Denault, et al, [9] in 1994 referred to detection-by-appearance and detection-by-behavior. Forrest, et al, [10] in 1994 said there were three types: activity monitors, signature scanners, and file authentication programs.

Intrusion detection types began converging on two main types in 1994: misuse and anomaly. Crosbie and Spafford [11] defined misuse detection as watching for certain actions being performed on certain objects. They defined anomaly detection as deviations from normal system usage patterns. Kumar and Spafford [12] also referred to anomaly and misuse detection in 1994. Many other researchers, too numerous to mention them all, have also referred to misuse and anomaly as the two main types of intrusion detection, from 1994 up to the

present time.

However, other types of intrusion detection continue to be mentioned. Ilgun, Kemmerer, and Porras [13] in 1995 referred to four types: Threshold, anomaly, rule-based, and model-based. Esmaili, Safavi-Naini, and Pieprzyk [14] in 1996 said the two main methods are statistical and rule-based expert systems.

Debar, Dacier, and Wespi, [15] in 1999 referred to two complementary trends: (1) The search for evidence based on knowledge; and, (2) the search for deviations from a model of unusual behavior based on observations of a system during a known normal state. The first they referred to as misuse detection, detection by appearance, or knowledge-based. The second they referred to as anomaly detection or detection by behavior.

Bace [16] in 2000 described misuse detection as looking for something bad and anomaly detection as looking for something rare or unusual. Marin-Blazquez and Perez [17] in 2008 said that there are three main approaches: signature, anomaly, and misuse detection.

While descriptive, these various labels over time are inconsistent and do not favor an analytical discussion of network intrusion detection. Not all of them are necessary, they are not mutually exclusive, and as individual groups they have not been demonstrated as being complete. Rather than arbitrate which of these labels should be used and how they should be defined, new labels have been created to describe types of local network intrusion detection in a manner which favors an analytical environment. These new types are explained in the Methodology section of this paper.

DATA PREPARATION

Data selection begins with manual feature selection and many strategies have been used. Lunt's [18] features included bad login attempts, amount of network activity by type and host, and number of times each account was accessed. Cannady [19] used the protocol, source port, destination port, source address, destination address, ICMP type, ICMP code, data length, and raw data.

A standardized vector of 41 elements resulted from a data set produced in 1998 by MIT's Lincoln Laboratory under DARPA sponsorship and was listed by Kayacik, Zincir-Heywood, and Heywood [20]: duration of the connection, protocol, service, flag, source bytes, destination bytes, land, wrong fragment, urgent, hot, failed logins, logged in, number of compromised conditions, root shell, su attempted, number of root accesses, number of file creation operations, number of shell prompts, number of operations on access control files, number of outbound commands in an ftp session, is hot login, is guest login, count of connections, service count, SYN errors, service SYN errors, REJ errors, service REJ errors, same service rate, different service rate, percent of connections to different hosts, count of connections with same destination host, count of connections with same host and service, percent of connections having the same destination host and service, percent of different services on same host, percent of connections to the host having the same source port, percent of connections to the same service coming from different hosts, percent of SO errors for host, percent of SO errors for host and service, percent of RST errors for host, and percent of RST errors for host and service. See that paper for more detailed information on these features.

Lee and Heinbuch [21] in 2001 experimented with SYNs received, SYNs dropped, SYN-ACKs sent, number of new connections made, number of queued SYNs at end of the last window, number of queued SYNs at end of this window, queued SYNs timed out, maximum number of connections open, FIN-ACKs sent, FIN-ACKs received, resets sent, resets received, number of connections closed, number of source sockets for received data packets, number of destination sockets for sent packets, number of destination ports for received packets, and number of source ports for sent packets.

Mukkamala, Janoski, and Sung [22] in 2001 assigned weights to commands, such as 1 to exit and 10 to chown, and determined the average and highest weight for a user. Web related data was also examined, such as the number of 404 errors.

Term frequency inverse document frequency (tf-idf) for text categorization in relation to intrusion detection was discussed by Zhang and Shen [23] in 2005.

LaRoche and Zincir-Heywood [24] in 2006 presented a vector for wireless intrusion detection consisting of the subtype of the frame, the destination address, the sender address, the BSSID of the access point, the fragment number, the sequence number, and the channel.

Livadas et al. [25] in 2006 added network information such as the maximum initial congestion window, who initiated the flow, percentage of packets pushed in a flow, variance of packet inter-arrival time for flow, and variance of bytes-per-packet for flow.

FEATURE REDUCTION

Feature reduction means to reduce the amount of input data in order to reduce the resources needed for analysis. It can also be called *feature selection* and *feature ranking*. The original feature reduction must be done by a knowledge engineer who determines what the first input data is going to be from the possibly huge amount of data that is available. The data can be computationally reduced later, usually in reference to reducing the number of elements in an input vector. Feature reduction is needed especially for Soft Computing methods because they typically access the data repeatedly, such as during a training process, and using a full data set can be computationally prohibitive.

Mukkamala, Sung, and Abraham in 2004 [26] explained that a complete analysis of feature ranking would require 2^n experiments to examine all possibilities, analyzing two variables at a time, then three variables at a time, etc., and would still not be infallible because the available data might be of poor quality.

Mukkamala and Sung in 2002 [27] proposed this feature reduction algorithm which was implemented after an Artificial Neural Network (ANN) was trained:

1. Delete one input feature from the (training and testing) data.
2. Use the resultant data set for training and testing the classier.
3. Analyze the results of the classier, using the performance metrics.
4. Rank the importance of the feature according to the rules.

5. Repeat steps 1 to 4 for each of the input features.

Using OA for Overall Accuracy, FP for False Positive rate, and FN for False Negative rate, these are the rules which were used to rank the importance of the features:

1. If OA increases and FP decreases and FN decreases, then the feature is unimportant.
2. If OA increases and FP increases and FN decreases, then the feature is unimportant.
3. If OA decreases and FP increases and FN increases, then the feature is important.
4. If OA decreases and FP decreases and FN increases, then the feature is important.
5. If OA does not change and FP does not change then the feature is secondary.

Sung and Mukkamala in 2003) [28] identified features with Support Vector Machines (SVM) and ANN. Abraham and Jain in 2004 [29] based feature reduction on the contribution the input variables made to the construction of a decision tree.

Chen, Abraham, and Yang in 2005 [30] proposed a Flexible Neural Tree (FNT) for feature reduction following this mechanism:

1. Initially the input variables are selected to formulate the FNT model with same probabilities.
2. The variables which have more contribution to the objective

function will be enhanced and have high opportunity to survive in the next generation by an evolutionary procedure

3. The evolutionary operators provide an input selection method by which the FNT should select appropriate variables automatically.

An evolutionary feature reduction method was proposed by Chimphlee et al. in 2005 [31] which was followed by fuzzy clustering.

CLUSTERING VS. CLASSIFICATION

Clustering is the unsupervised division of unlabeled sets into subsets, and classification is the supervised determination of the labeled subset to which an element belongs. In intrusion detection, for example, clustering would ideally consist of the unsupervised dividing of all computer usage into subsets which would put intrusions into some subsets and normal traffic into other subsets. Classification would then be the determination of which subset some new computer usage belongs: intrusion or normal.

Clustering can be done by Evolutionary Computing (EC), Fuzzy Reasoning, Swarm Intelligence, and Self-Organizing Maps (SOM), as well as by other methods. SOM generally both clusters data and also classifies it. Other Soft Computing methods generally classify.

Newsome, Karp, and Song in 2006 [32] explained how to thwart clustering and classification: 1) inseparability attacks to blur distinctions between classes; and, 2) red herring attacks to create false classifications.

THE INTRUSION DETECTION PROBLEM

Intrusion Detection on a high level is an intractable problem with some aspects of it being unsolvable. This has been proven and illustrated many ways.

Cohen noted in 1987 [33] (Page 31) that the determination of a virus was undecidable: "*In order to determine that a program 'P' is a virus, it must be determined that P infects other programs. This is undecidable since P could invoke the decision procedure 'D' and infect other programs if and only if D determines that P is not a virus.*" He also noted that tracing exact information flow requires NP-Complete time. Cohen listed these specific problems as being undecidable:

- Detection of a virus by its appearance
- Detection of a virus by its behavior
- Detection of an evolution of a known virus
- Detection of a triggering mechanism by its appearance
- Detection of a triggering mechanism by its behavior
- Detection of an evolution of a known triggering device
- Detection of a virus detector by its appearance
- Detection of a viral detector by its behavior
- Detection of an evolution of a known viral detection

Other researchers have confirmed the difficulty of the problem. Using misuse to mean all kinds of badness with a scale of 0 (normal) to 1 (misuse), Helman, Liepins, and Richards in 1992 [8] showed that expert systems are NP-Hard. Me in 1998 [34] used a genetic algorithm to manipulate vectors based on

event counts, and said the problem was NP Complete.

These researchers described it linguistically. (Intrusion Detection) *Rules need to be maintained and managed. This process is labor-intensive and error-prone.* [35] *Signature-based network intrusion-detection systems (NIDS) often report a massive number of simple alerts of low-level security-related events.* [36] (Page 1) *It is common for a NIDS (Network IDS) to raise thousands of alerts per day, most of which are false alerts.* [37] (Page 1)

Here are some more colorful ways of illustrating the intrusion detection problem. It is an arms race where one side advances and then the other side advances, *ad infinitum*. It is a treadmill where the researcher figures out a way to detect the latest kind of intrusion, the intrusion changes, and the researcher does it again, *ad infinitum*. As a general summary, a detection method can exist for every kind of intrusion and an intrusion method can exist for every kind of detection. It is an evolutionary system where both sides continually evolve to outwit the other side. Cohen [33] (Page 34) compared this situation to an old western saying: *ain't a horse that can't be rode, ain't a man that can't be throwed*.

SOFT COMPUTING AND INTRUSION DETECTION

One of Cohen's [33] (Page 30) conclusions from the intrusion detection problem above was *this leaves us with imprecise techniques*. This overall difficulty with intrusion detection leads to Soft Computing for solutions. Although the proper term *Soft Computing* was initiated by Zadeh in the early 1990's [38], some components of Soft Computing are much older: Bayes probability was

published in 1763 [39]; Artificial Neural Networks (ANN) was published in 1943 [40]; and, Fuzzy Sets was published in 1965 [41]. Soft Computing is called *soft* in order to contrast it with hard computing*, i.e.*, exactness. Some characteristics of Soft Computing include probability, randomness, inexactness, and biological attributes. Soft Computing has similarities with Artificial Intelligence (AI), but AI researchers have traditionally used hard computing. An additional occasional characteristic of Soft Computing is emergence [42]. Many refer to this characteristic as being a *Black Box*, described as a device where something goes in and something comes out, but what happens inside the box cannot be seen. Soft Computing is not really a Black Box---researchers write the software code for it and know exactly what is inside the box. However, results emerge from this code in ways which often cannot be readily understood. The graphical SOM map in this research addresses that problem. Other characteristics of Soft Computing are tractability, robustness, low solution cost, and tolerance for imprecision and uncertainty [43]. Soft Computing has been further described as "aimed at an accommodation with the pervasive imprecision of the real world", as well as to "exploit the tolerance for imprecision, uncertainty and partial truth to achieve tractability, robustness, low solution cost and better rapport with reality", and "the role model for Soft Computing is the human mind" [44] (Page 1).

These characteristics of Soft Computing are summarized here as being the *Imprecision Principle*:

> Some labor intensive, error-prone, biological, evolutionary, incomplete, inconsistent, impossible, un-scalable, unsolvable,

> undecidable, complex, ambiguous, non-intuitive, and/or intractable problems are best resolved with imprecise, probabilistic, fuzzy, inexact, emergent, non-intuitive, uncertain, and partially true methods which are designed largely by the intuition, judgment, and experience of knowledge engineers.

Billions of network packets are continually being created on the Internet and it is impossible for the network security analyst to know the reason for every single one of these packets being sent by routers, switches, appliances, programs, and users from all over the world. This network traffic, from an analyst's point of view, contains imprecision, uncertainty, and partial truths, meaning that decisions must routinely be made from incomplete information. Soft Computing thus provides a coping mechanism for network security tasks that otherwise would be impossible.

Extensive research has been done on using Soft Computing for intrusion detection and [45] examines the state of the art in this field. Ten general Soft Computing components stand out in intrusion detection research: Artificial Immune Systems (AIS), Artificial Neural Networks (ANN), Bayes Reasoning, Decision Trees (DT), Dempster-Shafer (D-S), Evolutionary Computing (EC), Fuzzy reasoning, Hidden Markov Model (HMM), Self-Organizing Maps (SOM), and Swarm Intelligence.

Soft Computing generally refers to the hybridization of these components, for example ANN with Fuzzy. Three general types of hybridization are possible: consecutive, ensemble, and interactive.

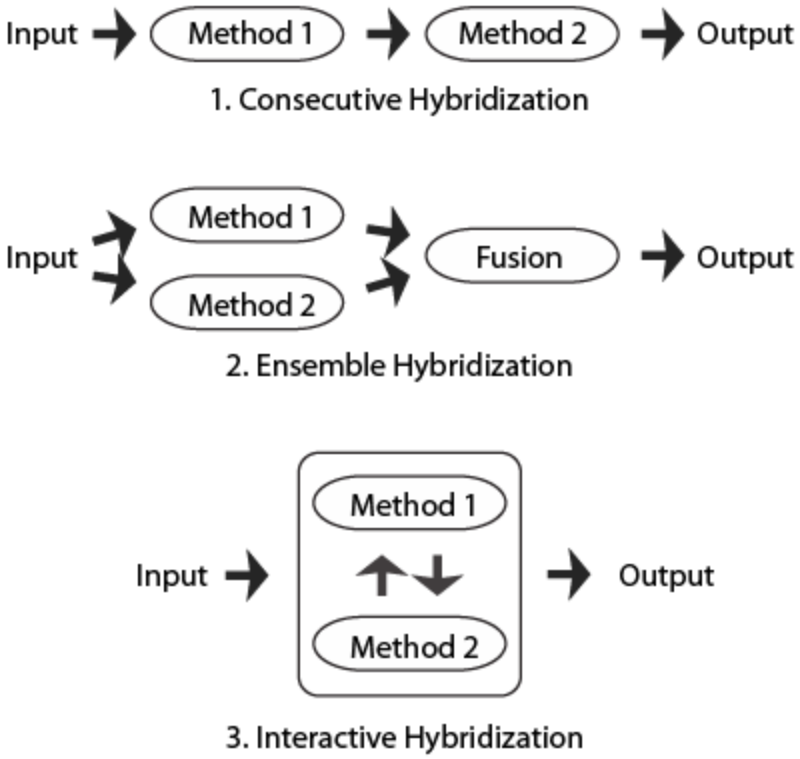


Figure 1, Types of Hybridization

Figure 1 illustrates three types of hybridization. The first type is consecutive in which the output of one method is the input of the next method. This can continue in a series larger than two methods. The second type of hybridization is ensemble in which two or more methods are implemented in parallel and the results are fused. The third type of hybridization is interactive in which two or more methods interact with each other in some way, such as a loop, before they produce a single output. Complex hybridizations with combinations of these types are also possible.

These methods of data fusion have been researched for ensemble hybridization. Averaging, voting, or using the maximum value was suggested by [22]. A positive result if any of the interim results is positive [46]. These methods

were proposed by [47]: Bayes Average, Bayes Product, Dempster-Shafer, Recognition, Substitution, and Rejection Rates (RSR), the Predictive Rate Method (PRM), and Rogova's Class Level Method. This method was proposed by [48] for four interim results: Let the four interim results be $a_n, b_n, c_n, \text{and } d_n$ for each of the four parallel methods. Then, let x_n be the final result which is to be determined. Find x_n such that $|a_n - x_n| + |b_n - x_n| + |c_n - x_n| + |d_n - x_n|$ is minimized. Although not called *fusion*, a subjective logic method of correlation of intrusion alarms was explained by [49] and the Dendritic Cell Algorithm (DCA) fuses a series of four inputs into a contextual output [50].

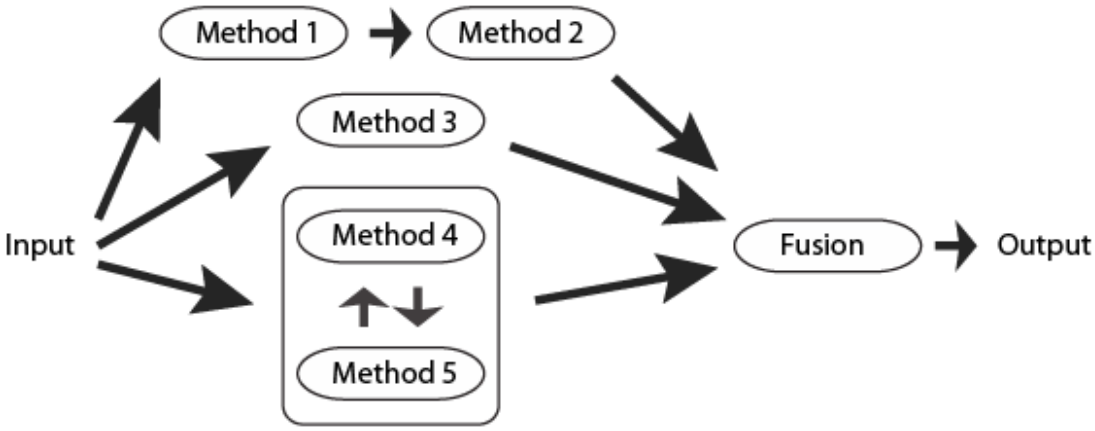


Figure 2, Complex Hybridization

Figure 2 illustrates an example of complex hybridization. The overall layout is an ensemble which consists of consecutive methods 1 and 2, Method 3, and interactive methods 4 and 5. The three interim results in Figure 2 are fused for a single output. The complex nature of intrusion detection favors hybridized methods because a single method cannot cover all of the possibilities and intricacies involved in discovering all kinds of intrusion attempts.

A study of existing Soft Computing intrusion detection strategies was published in [45]. Numerous examples of hybridized combinations of methods were noted in this paper, but comprehensive direct comparisons of methods are not feasible because of the many different kinds of intrusions, criteria, and variables which were used in setting up test environments. Not all researchers used the same variations of methods—many different types of ANN were evaluated, for example. The statistical relevance of the results of the comparisons is not known. One clear winner in testing, though, was the ensemble hybridization method [51] because it can use diverse approaches in parallel. Compare this with medical tests in which the patient might get blood tests, an electrocardiogram, x-rays, and a urinalysis *in parallel* with the results from these tests all going to a physician who *fuses* the data into a diagnosis.

Self-Organizing Maps (SOM)

The research for this paper began as an experiment with SOM to see if it would cluster Storm Worm intrusions from firewall logs. The experiment was successful and was published in [52] and [53], (see for details). This SOM was later called 1D ANNaBell, for a one dimensional ANN that *rang a bell* for alerts. A SOM is a type of Artificial Neural Network (ANN), but the structure of a SOM is significantly different from a feed forward and back propagation ANN. Both have nodes (*neurons*), but in a classical ANN the nodes are connected in such a way that data are manipulated while conceptually flowing through the node structure resulting in an answer, whereas in a SOM the nodes represent clusters in space which are conceptually pulled like rubber over the data, which can result in a

representational visual display of the data. The original ANN was created by McCulloch and Pitts in 1943 [40]. SOM was conceived by Kohonen in 1982 [54] and is sometimes called a *Kohonen Network*. A SOM can represent multidimensional data in a smaller dimensional space, like shining light on a three-dimensional object to create a two-dimensional shadow. Hexagonal SOM maps have been used to display high dimensional data in a more human understandable format.

A SOM primarily clusters the data, but it can also classify data by finding the nearest node in space, called the *Best Matching Node* (*BMN*). A visual display can be produced, but is not always necessary. The visual display for 1D ANNaBell is not very useful, but the node data of 1D ANNaBell can be used by scripts to produce intrusion detection alerts. The graphics were improved with a newer version called *3D ANNaBell* which was published in [55], [56], and [57], which see for detailed information.

Other researchers have also studied SOM for intrusion detection. Hoglund and Hatonen in 1998 [58] constructed a SOM prototype for visualization of anomaly detection with a hex map based on user account logs (CPU times, characters transmitted, and blocks read). A grey-scale U-matrix scheme showed the relative distances between the nodes and highlighted the clusters of data. (See Ultsch and Siemon [59] for a description of U-matrix.)

Lichodzijewski, Zincir-Heywood, and Heywood in 2002 [60] described how to do a hierarchy SOM with time series data, and Kayacik, Zincir-Heywood, and Heywood in 2003 [61] created a hierarchical hex SOM with TCP data. A 475-

node SOM was produced by Ramadas, Ostermann, and Tjaden in 2003 [62] based on network traffic.

Excellent demonstrative graphics were provided by Vicente and Vellido in 2004 [63] for a growing hierarchical SOM; Tauriainen in 2005 [64] described a robust SOM for detecting P2P activity; and, Wetmore, Zincir-Heywood, and Heywood in 2005 [65] proposed dynamic subset selection in order to train SOMs much more quickly.

Kayacik and Zincir-Heywood in 2006 [66] described a method of labeling nodes in a U-matrix hex map to clarify the meanings of resulting maps.

Payl Over SOM for Intrusion DetectiON (POSEIDON) finessed an IDS by using SOM instead of payload length for preprocessing classification as explained by Bolzoni, Etalle, and Hartel in 2006 [67] and Bolzoni and Etalle in 2008 [68].

Fuzzy Inference Systems (FIS)

Fuzzy inference is a classifier that helps to cope with inexact descriptions of intrusions where the Imprecision Principle applies. Network indications of a P2P botnet might be, for example, that a local computer has contacts with a large number of external IP addresses, the packet size entropy is high, a wide range of destination ports is used with many high-numbered ports, and the UDP ratio is high.

Luo in 2000 [69] refined an algorithm for mining fuzzy association rules, defined the concept of fuzzy frequency episodes, and presented an original algorithm for mining fuzzy frequency episodes, noting that security itself includes

fuzziness. Degrees of attack guilt were outlined by Noel, Wijesekera, and Youman in 2002 [70] which can be used for Fuzzy inference: Absolute innocence, probable innocence, possible innocence, possible guilt, probable guilt, and provable guilt.

Fuzzy IDS (FIDS) was proposed by Tillapart, Thumthawatworn, and Santiprabhob in 2002 [71] as a framework for network intrusions, including SYN and UDP floods, Ping of Death, E-mail Bomb, FTP and Telnet password guessing, and port scanning. Numerous example rules are provided in the paper.

Cougarr-based IDS (CIDS) utilized a Fuzzy Inference System in the Decision Agent [72]. Cougarr stands for cognitive agent architecture and is open source software available at http://www.cougarr.org (8/2/2009).

Rule-Based Fuzzy Cognitive Maps (RBFCM) were developed by Wang and Daniels in 2008 [73] to deal with causal relations for evidence graphs and hierarchical reasoning in network forensics.

Su, Yu, and Lin in 2009 [74] proposed using fuzzy association rules for incremental mining so that a real-time IDS can be implemented with this method.

In Association Based Classification (ABC), Tajbakhsh, Rahmati, and Mirzaei in 2009 [75] used fuzzy c-means for clustering and fuzzy association rules for classification.

Artificial Immune Systems Danger Theory (AISDT)

Using the generation of T cells in the Biological Immune System (BIS) as a basis for detecting computer viruses was proposed by [10] in 1994. Strings were randomly generated, some of which matched protected data and

represented *self*. Other strings did not match protected data and represented *non-self*. The strings which matched protected data (self) were dropped, while the remaining strings were used somewhat like T cells and were compared with the protected data. In a process called *negative selection*, a match of a non-self string with protected data indicated that a change in the protected data had occurred, indicating an intrusion. This would later be called an Artificial Immune System (AIS).

After evaluating AIS for network intrusion detection, Kim [76] in 2001 noted that the size of data which defines self and non-self is enormous; the system could not manage to generate a single valid detector after one day; over 600,000 detectors would be required; and, it would take over 1,000 years to generate that many detectors. He concluded that AIS had a severe scaling problem.

Matzinger in 1994 [77] had proposed a new viewpoint of the human immune system called *Danger Theory*, which emphasized the recognition of danger instead of self/non-self. Examples of danger in this context are tissue destruction, temperature, and an abnormally released molecule from a cell.

Danger Theory was applied to AIS by Aickelin [78] in 2002 with a suggestion to use it for computer security. He noted that negative selection is imperfect resulting in false positives being inevitable; that the self/non-self boundary is blurred; and, that self changes over time. With the qualification that negative selection is important, he noted the following seven considerations for AIS Danger Theory:

1. An Antigen Presenting Cell (APC) is required which can present a danger signal.
2. *Danger* in AIS Danger Theory just means something interesting, such as in data mining, and does not necessarily refer to actual danger.
3. A Danger Signal can be positive or negative (no danger).
4. A spatial Danger Zone needs a measure of proximity, such as distance or time.
5. A Danger Signal itself should not lead to further Danger Signals.
6. Priming killer cells via APCs in spatially distributed models might be relevant.
7. Examples of other considerations are how many antibodies should receive signals from a given APC; and, Danger Theory relies on concentrations, not binary matching.

He also noted that Danger Theory had quite a number of elements and might need to be altered for AIS.

In relation to computer security Anomaly Detection, Aickelin [78] noted that when a detector (*T cell*) is activated, it is reported to a human operator who decides if there is a true anomaly. If so, then the detector becomes a memory (persistent) detector. He noted that scaling is a problem in AIS: *…it becomes more and more problematic to find a set of detectors that provides adequate coverage whilst being computationally efficient.* He adds that Danger Theory assists in the scaling issue: *It restricts the domain of non-self to a manageable*

size, removes the need to screen against all self, and deals adaptively with scenarios where self (or non-self) changes over time. He suggested these as possible Danger Signals in computer security:

- Too low or too high memory usage
- Inappropriate disk activity
- Unexpected frequency of file changes as measured for example by checksums or file size
- SIGABRT signal from abnormally terminated UNIX processes.
- Presence of non-self.

Powers and He in 2008 [79] proposed a hybrid AIS/SOM system, but their AIS component was without Danger Theory. The AIS results were fed into a SOM for classification.

Fu in 2008 [80] used a clustering algorithm of the data to represent tissue in an AIS system so that changes in this tissue could produce danger signals. Each cluster represented a cell, which had mass, age, and a location.

A method inspired by dendritic cells was proposed by [81] in 2008 suggesting excessive CPU load, frequency of file changes, bandwidth saturation, and abnormal rates of e-mail communications as sources of danger signals.

Dasgupta and Niño wrote a comprehensive review of AIS, including Danger Theory, in 2009 with a section on computer security [82]. Wu and Banzhaf reviewed AIS, including Danger Theory, and other computationally intelligent methods in 2010 [83] noting that most of the algorithms were tested on benchmark datasets, but that real-world environments are far more complicated;

that the scaling problem needs to be overcome; and, that current AIS algorithms oversimplify their counterparts in immunology.

Kulis, et al, in 2011 [84] proposed a fuzzy dendritic cell algorithm with access to memory as an adaptation to AIS Danger Theory to reduce the amount of antigens sampled in order to improve runtime costs.

CHAPTER 3 – METHODOLOGY AND COMPONENTS

A comprehensive methodology was developed which overhauled concepts of intrusion detection including a new model of intrusion detection types created in order to facilitate analytical research in this area; a computational model created in order to lay a foundation for data analysis; and, a hybrid system composed of an Invisible Mobile Network Bridge (IMNB), a Self-Organizing Map (SOM), a Fuzzy Inference System (FIS), Svensson and Josang (SJ) Fusion, and Artificial Immune System Danger Theory (AISDT). A subsection for each of these concepts follows.

LLNIDS TYPES OF INTRUSION DETECTION

Historical descriptions of types of intrusion detection have been inconsistent and have not favored an analytical discussion of network intrusion detection. Not all of the historical labels of types are necessary, they are not mutually exclusive, and as individual groups they have not been demonstrated as being complete. Rather than arbitrate which of these previous labels should be used and how they should be defined, new labels have been created to describe types of local network intrusion detection in a manner which favors an analytical environment.

These new types are explained below, but first some terminology needs to be stated in order to later describe the types. An *Intrusion Detection System (IDS)* is software or an appliance that detects intrusions. A *Network Intrusion Detection System (NIDS)* is an appliance that detects an intrusion on a network.

In this research, *network* means a landline network. *Local* network intrusion detection refers to the instant case of network intrusion detection.

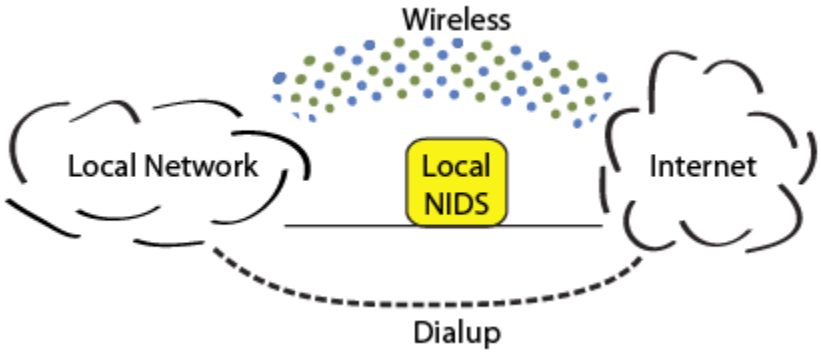


Figure 3, A Local Landline NIDS

Figure 3 illustrates the location (in yellow) of a Local Landline Network Intrusion Detection System (LLNIDS) as used in this research. It is an IDS on a landline between a local network and the Internet. The point of view of this research is from inside the LLNIDS. Users on the local network may have other ways of accessing the Internet that bypass the LLNIDS, such as wireless and dialup. This research is restricted to the LLNIDS as described here.

Examples of detection which are not Local Landline Network Intrusion Detection (LLNID) include detection on the host computer, detection by someone else out on the Internet, or detection by someone out in the world, such as a perpetrator bragging in a bar. This research concerns LLNID and the new types described in this paper refer to LLNID. A network intrusion in this context means one or more transmissions across the network that involves an intrusion. A single Internet transmission is often called a *packet*. Therefore, using this terminology, the physical manifestation of an intrusion on a network is one or more packets, and intrusion detection is the detection of these packets that

constitute intrusions. Intrusion detection research needs a model of types of intrusions and types of intrusion detection that benefits analysis of methods. This research focuses only on LLNID. These are the proposed types of intrusions for the special case of local landline network intrusion detection that facilitate intrusion detection research analysis in the LLNID context:

- **Type 1Intrusion:** An intrusion which can be positively detected in one or more packets in transit on the local network in a given time period.
- **Type 2 Intrusion:** An intrusion for which one or more symptoms (only) can be detected in one or more packets in transit on the local network in a given time period.
- **Type 3 Intrusion:** An intrusion which cannot be detected in packets in transit on the network in a given time period.

These three types of intrusions are necessary for analytical research in order to indicate and compare kinds of intrusions. A positive intrusion is different than only a symptom of an intrusion because immediate action can be taken on the first whereas further analysis should be taken on the second. Both of these are different than intrusions which have been missed by an LLNIDS. To show that these three types are mutually exclusive and are complete for a given time period, consider all of the intrusions for a given time period, such as a 24-hour day. The intrusions which were positively identified by the LLNIDS are Type1 intrusions. Of the remaining intrusions, the ones for which the LLNIDS found symptoms are Type 2. Here the hypothesis is that the LLNIDS can only find an

intrusion positively or only one or more symptoms are found. No other results can be returned by the LLNIDS. Therefore, the remaining intrusions are Type 3, which are intrusions not detected by the LLNIDS. No other types of intrusions in this context are possible.

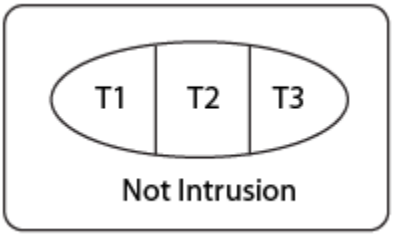


Figure 4, Types of Intrusions for LLNIDS

Figure 4 is a diagram that illustrates the types of intrusions as described above. An intrusion is either Type 1, Type 2, Type 3, or it is not an intrusion.

Those were the types of intrusions. Next are the types of intrusion detection. There are three types of network intrusion detection that correspond to the three types of intrusions in the LLNID context:

- **Type 1Network Intrusion Detection:** A Type 1 Intrusion is detected in a given time period.
- **Type 2 Network Intrusion Detection:** One or more symptoms (only) of a Type 2 Intrusion are detected in a given time period.
- **Type 3 Network Intrusion Detection:** No intrusion is detected in a given time period.

Admittedly, Type 3 is not a detection but the lack of detection. It is included because these three types of detection correspond to the three types of intrusions and Type 3 Intrusion Detection facilitates analysis of intrusion

detection methods. Examples of Type 3 Intrusion Detection are nothing was detected; no attempt was made at detection, an intrusion occurred but was not detected by the LLNIDS; and, no intrusion occurred. All of these have the same result: there was no detection of an intrusion by the LLNIDS.

Each of the three network intrusion detection types is necessary to describe all of the types of intrusion detection. A positive detection of an intrusion is different than just a symptom of an intrusion because a positive detection can be immediately acted upon while a symptom indicates that further analysis is needed. Both of these are different than intrusions that are missed by network intrusion detection. To show that these types are mutually exclusive and complete for a given time period, consider an LLNIDS looking at network packets for a given time period, say a 24-hour day. For all packets that the LLNIDS determines positively indicates an intrusion the LLNIDS has accomplished Type 1 intrusion detection. Of the remaining packets, for each packet that the LLNIDS determines is a symptom of an intrusion the LLNIDS has accomplished Type 2 intrusion detection. The remaining packets represent Type 3 intrusion detection. These three types of network intrusion detection are complete in this context because they cover all possibilities of intrusion detection. In common language, Type 1 is a certainty, Type 2 is a symptom, and Type 3 is an unknown.

Those were types of intrusion detection. Next are types of methods and alerts. LLNID methods can be defined in terms of the three intrusion types:

- **Type 1NID Method/Alert:** A method that detects a Type 1 Intrusion and an alert that indicates a Type 1Intrusion.

- **Type 2 NID Method/Alert:** A method that detects a symptom of a Type 2 Intrusion and an alert that indicates a symptom (only) of a Type 2 Intrusion.
- **Type 3 NID Method/Alert:** A method that does not exist, thus there is no alert.

These types of methods and alerts are necessary to differentiate that some methods are positively correct, other methods only indicate symptoms of intrusions, and some methods do not exist. They are mutually exclusive because a local method either positively indicates an intrusion (Type 1), it only detects a symptom of an intrusion (Type 2), or it does not exist (Type 3). They are complete because there are no other types of methods in this context.

Those were types of methods and alerts. Next are types of false positives. The term *false positive* generally has meant that an intrusion detection system has sent a false alarm. False positives are generally undesirable because the false positive rate of intrusion detection systems can be high and can use up a lot of seemingly unnecessary, and limited, resources. However, with these new types, the concept of a false positive is different for different intrusion detection types in the LLNIDS context.

- **Type 1False Positive:** A Type 1 Method produces an alarm in the absence of an intrusion.
- **Type 2 False Positive:** A Type 2 method produces an alarm in the absence of an intrusion.
- **Type 3 False Positive:** Does not exist because no alarm is

produced.

A Type 1 False Positive indicates a problem with the type 1 method which should be corrected. Type 2 False Positives are expected because Type 2 Methods do not positively detect intrusions, they only detect symptoms of intrusions. There is no Type 3 False Positive because no detections and alerts are produced for Type 3 Intrusion Detections. These types of false positive are necessary because they each indicate separate network intrusion detection issues. Type 1 is a network intrusion detection problem which needs to be corrected and Type 2 is expected. The two types of false positive are mutually exclusive and complete because only Type 1 Network Intrusion Detection can produce a Type 1 False Positive and only Type 2 Network Intrusion Detection can produce a Type 2 False Positive. No other types of false positives in this context are possible. Since Type 1 and Type 2 of local network intrusion detection methods are mutually exclusive, these are also mutually exclusive.

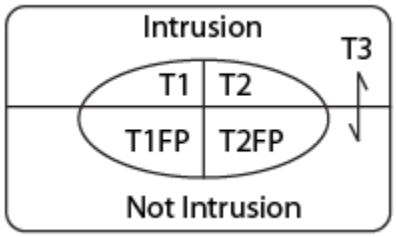


Figure 5, Types of Intrusion Detection for LLNID

Figure 5 is a Venn diagram which illustrates types of intrusion detection in the LLNIDS context. The horizontal line separates intrusions at the top from non-intrusions at the bottom. A Type 1 detection is in the upper left of the circle if it is actually an intrusion or it is in the lower left of the circle if it is a false positive. A

Type 2 detection is in the upper right of the circle if it is actually an intrusion or it is in the lower right of the circle if it is a false positive. Everything outside of the circle is Type 3 detection whether it is an intrusion or not.

This typing system allows illustration that empirically most intrusion detection is not Type 1 (positive detections), but Type 2 (symptoms of detections), and Type 3 (missed detections). This differentiation is essential in proceeding in a scientific way for improved intrusion detection.

Previously labeled types of intrusion detection do not fit neatly into these three new types. Misuse detection, for example, in some cases could indicate a definite intrusion and would then be Type 1, or it could indicate only symptoms of intrusions in other cases and would then be Type 2. The comparison of false positives of different methods of Misuse Detection is an invalid technique unless Type 1 methods are compared only with Type 1 methods and Type 2 methods are compared only with Type 2 methods. Anomaly detection, for example, would tend to be Type 2, but some anomalies could clearly indicate intrusions and would be Type 1. Type 1 and Type 2 methods of Anomaly Detection should be separated before making any comparisons. Likewise with intrusion detection labels based on activity, appearance, authentication analysis, behavior, knowledge, models, profiles, rules, signature, static analysis, statistics, and thresholds. These are still useful as descriptive terms, but they are not as useful in analyzing methods of determining whether or not an intrusion has occurred because they allow the comparisons of *apples and oranges* in numerous ways. The labels Type 1 and Type 2 give us more analytical information: either an

intrusion has occurred or else only a symptom of an intrusion has occurred. Type 3 intrusions tell us that we should find out why an intrusion was not detected in the network traffic so that we can create new rules to find more intrusions in the future. Previously labeled types of intrusion detection do not give us as much analytical information as do types 1, 2, and 3.

Using this system, one can clearly state objectives of LLNID research in a new way which was previously only implied. The significance of *given time period* is apparent in the descriptive of these objectives because the objectives are stated in terms of progress from one time period to another time period. Here are specifics for LLNID research:

- **Type 3 NID Research:** Find ways of detecting intrusions that are currently not being detected, moving it up to type 2 or 1 intrusion detection.
- **Type 2 NID Research:** Improve Type 2 Intrusion Detection with the goal of moving it up to Type 1 Intrusion Detection.
- **Type 1NID Research:** Improve Type 1 Intrusion Detection so that it is faster, uses fewer resources, and has fewer false positives.

Each of these types of research are necessary because finding new methods of intrusion detection is different than improving symptom detection which is different than making Type 1 Intrusion Detection more efficient. They are also complete because there are no other types of intrusion detection research in this context.

Table 1, Summary of LLNID Types

	Type 1	**Type 2**	**Type 3**
Intrusion	This can be positively detected by LLNIDS	A symptom of this can be detected by LLNIDS	This is not detected by LLNIDS
Intrusion Detection	This positively detects an intrusion	This detects one or more symptoms (only) of an intrusion	An intrusion is not detected
Method	How to positively detect an intrusion	How to positively detect a symptom of an intrusion	An intrusion is not detected
Alert	This positively signifies an intrusion	This signifies a symptom of an intrusion	This does not occur
False Positive	An alert *positively* signifies an intrusion, but there is no intrusion	An alert signifies a symptom of an intrusion, but there is no intrusion	An alert does not occur
Research	Improve Type 1 Intrusion Detection, such as by increasing the speed of detection, using less resources, and having fewer false positives	Improve Type 2 Intrusion Detection so that it becomes Type 1 Intrusion Detection	Detect Type 3 intrusions so that they become Type 2 or Type 1

Table 1 summarizes the types discussed in this section. These are some ways of how researchers can use these types: research that compares false positive rates of Type 1 methods with false positive rates of Type 2 methods is not valid because Type 1 methods are not supposed to have false positives whereas Type 2 methods are expected to have false positives. Discounting Type 3 intrusion detection because of the amount of time taken may be irrelevant if otherwise the intrusion would not be found, at all. Proposing that intrusion prevention will replace intrusion detection is a false claim so long as types 2 and

3 intrusions continue to exist. Rather than disregarding Type 2 methods, research should attempt to fuse the results of Type 2 methods in order to move them up to Type 1.

THE LLNIDS COMPUTATIONAL MODEL

The proposed Local Landline Network Intrusion Detection System (LLNIDS) Computational Model covers intrusion detection data from packet analysis to sophisticated Soft Computing methods. The LLNIDS Computational Model begins with a transmission of digital network traffic and proceeds stepwise to higher concepts. The terminology for the input data changes depending upon the level of the concept. The lowest level concept in this research is the network transmission, which is a series of bits called a *frame* or a *packet*. *Frame* refers to a type of protocol, such as Media Access Control (MAC), which is used between two neighboring devices, where the series of bits are framed by a header at the start and a particular sequence of bits at the end. *Packet* refers to many types of protocols, such as Internet Message Control Protocol (ICMP), User Datagram Protocol (UDP), and Transmission Control Protocol (TCP). A packet is used for hops between numerous devices, such as Internet traffic. The length of the series of bits in a packet is often indicated at certain locations in the headers of the packets. A frame passes a packet between two neighboring devices, where another frame passes the same packet between the next two devices, and subsequent frames keep passing the packet forward until the journey of the packet is concluded. Since frames and packets are variable lengths, they are represented by a set of objects which represent the various elements of

information inside the frame or packet.

A Transmission (T) consists of a set of objects (o) representing elements of information in that transmission.

$$T = \{o_1, o_2, o_3, \ldots, o_{n_T}\} \quad (1)$$

where $n_T > 0$. Examples of objects in a transmission are the source MAC address, IP address, and port; the destination MAC address, IP address, and port, the direction of the traffic, protocols used, flags set, sequence numbers, checksums, type of service, time to live, fragmentation information, and the content being sent.

```
20:51:40.885767 IP (tos 0x0, ttl  62, id 40857, offset 0, flags [none],
proto: UDP (17), length: 144) 431.240.64.213.16402 > 438.87.208.113.16402:
[udp sum ok] UDP, length 116
        0x0000:  4500 0090 9f99 0000 3e11 bd3f 83e6 40d5  E.......>..?..@.
        0x0010:  8a57 d071 4012 4012 007c f873 81c8 000c  .W.q@.@..|.s....
        0x0020:  de8c 6c0a ce79 c2bd a91f 1800 0081 ea3f  ..l..y.........?
        0x0030:  0000 85ba 01fd d766 6edd a1ce 2a00 09a1  .......fn...*...
        0x0040:  0001 1d17 0000 0573 20dc 08fd 0000 c38d  .......s........
        0x0050:  81ca 000c de8c 6c0a 0127 6865 6374 6f72  ......l..'rector
        0x0060:  7265 6761 6c61 646f 2d63 6f75 7479 2d32  delgados-conty-2
        0x0070:  3330 4031 3331 2e32 3330 2e36 342e 3231  31@431.240.64.21
        0x0080:  3300 0000 80c1 0002 de8c 6c0a 1d15 0000  3.........l.....
```

Figure 6, A Sample Packet

Figure 6 is a sample packet as displayed by tcpdump [85]. Header information extracted from the packet is displayed across the top. The leftmost column is the byte count in hexadecimal. The packet itself is displayed in hexadecimal in columns in the middle. Character representations of the hexadecimal code, when possible, are shown on the right. The packet is a transmission set, T, with variable length objects as elements. Example object elements for this set are the protocol, UDP, and the destination port, 16402, both

of which have been extracted from the packet code.

If an intrusion occurs on a local landline, it occurs in one or more T, so LLNID means inspecting T's for intrusions. Not all of the available data in T has equal relevance to intrusion detection and the reduction of the amount of data is desirable in order to reduce the resources needed for analysis. This process has been called *feature deduction* [86], *feature reduction* [86], *feature ranking* [87], or *feature selection* [86]. The first feature selection must be done manually by a knowledge engineer, after that the features can be ranked and/or reduced computationally. Soft Computing methods often use data structures of n-tuple formats, such as one-dimensional arrays, sets, vectors, and/or points in space. Since sets can be used as a basis to describe these data structures, the next step in the computational model is to convert features of T into higher levels of sets which can be further manipulated for data analysis. The next set to be considered is an Event (E) which consists of a set of elements (e) obtained from the objects of T, and which changes the concept level from a transmission of objects to a set of elements:

$$E = \{e_1, e_2, e_3, \dots, e_{n_E}\} \tag{2}$$

where $n_E > 0$ and the following condition is also met:

$$1 \leq i \leq n_E, \forall e_i \in E, e_i \in T \tag{3}$$

How to construct e_i from the objects of T is feature selection--elements should be selected which can detect intrusions. The Imprecision Principle applies to feature selection and experimentation is appropriate. An example of possible elements for an event is the source IP address, the destination IP

address, the source and destination ports, the protocol, and the size of a packet crossing the network.

Table 2, A Sample Event

UDP	231.240.64.213	238.87.208.113	16402

Table 2 shows a sample event with the following elements: The protocol is UDP, the source IP address is 231.240.64.213, the destination IP address is 238.87.208.113, and the destination port is 16402. These elements were object elements in the sample transmission set shown above. The process of pulling data objects from a packet and saving them as Event elements is called *parsing the data*.

The next step is to add Meta-data (M), if appropriate, about the event consisting of meta-data elements (m):

$$M = \{m_1, m_2, m_3, , m_{n_M}\} \quad (4)$$

where $n_M \geq 0$. Meta-data is data about data. In this context, it means data about the transmission that is not inside the transmission, itself. Examples of meta-data are the time when a packet crossed the network, the device which detected the packet, the alert level from the device, the direction the packet was travelling, and the reason the packet was detected. The concept level has changed from a set of elements to a set of meta-data about the set of elements.

Table 3, Sample Meta-Data

20100916	00:14:54	FW

Table 3 shows sample meta-data for an event. The meta-data in this table

is the date, 20100916, and the time, 00:14:54, at which an appliance detected the transmission, and a label for the appliance that detected the packet, FW.

A Record (R) of the event includes both the event, itself, plus the meta-data:

$$R = M \cup E \tag{5}$$

An example of a record is an entry in a normalized firewall log. The concept level has changed from a set of meta-data to a set that includes both the elements and meta-data about those elements. In practice, the meta-data typically occurs in R before the elements to which the meta-data refers.

Table 4, A Sample Record

20100916	00:14:54	FW	UDP	231.240.64.213	238.87.208.113	16402

Table 4 is a sample record, which consists of meta-data and elements from the previous examples for M and E. Before proceeding to the next step, the attributes of R for a given analysis should be in a fixed order because they can later become coordinates in a location vector. Processing the data into fixed orders of attributes is called *normalizing the data*.

A Log (L) of records is a partially ordered set:

$$L = \{R_i\}_{i \in \mathbb{N}} \tag{6}$$

An example of a log is a file containing normalized firewall log entries. An infinite-like log could be live streaming data.

Table 5, A Sample Log

20100916	00:14:54	FW	UDP	231.240.64.213	238.87.208.113	16402
20100916	00:14:56	FW	TCP	216.162.156.85	198.18.147.222	40833
20100916	11:14:57	FW	ICMP	90.29.214.20	198.18.147.221	41170

Table 5 shows a sample log. It is like the sample record, above, except there are three entries instead of just one entry. The concept level has changed from a set of meta-data and elements to a collection of sets of meta-data and elements. L can be considered to be a set of vectors; L can also be considered to be a matrix. If L is a text file, each line of the file is one location vector and the entire file is a matrix, changing the concept level to a matrix.

If the features have been selected successfully, an intrusion, or one or more symptoms of it, should be able to be detectable in L. Therefore, LLNIDS intrusions and intrusion detection can be defined in terms of R and L. Let $\mathscr{R}$ be the universal set of R and let I_1 represent a set of $\mathscr{R}$ that describe a Type 1 Intrusion. Then I_1 is the set:

$$I_1 = \{R | R \in \mathscr{R}, R \text{ involves a Type 1 Intrusion}\} \qquad (7)$$

Formula 7 formulates a Type 1 Intrusion. Examples of Type 1 intrusions are a Ping of Death and a get request to a known malicious web site. These intrusions can potentially be prevented. I_1 has the same attributes as L in that it can be considered to be a set of location vectors or it can be considered to be a matrix. As matrices, the number of columns in I_1 and L for an analysis must be the same, but the number of rows in I_1 and L can be different. For reference below, let $\mathscr{I}_1$ be the universal set of all Type 1 intrusions. The concept level for $\mathscr{I}_1$ has changed from a matrix to a set of matrices. That was about intrusions. Now here is the function for Type 1 Intrusion Detection, I_1^D:

$$I_1^D(L) = \begin{cases} \text{True}, & \exists I_1 \in \mathcal{I}_1 \colon\ I_1 \subseteq L \\ \text{False}, & \text{otherwise} \end{cases} \qquad (8)$$

Formula 8 is the function for Type 1 Intrusion Detection, which returns True if an intrusion has been detected, otherwise it returns False. Next is Type 2 intrusions and intrusion detection. In most cases, one or more events occur which makes the security technician suspicious that an intrusion has occurred, but more investigation is necessary in order to reach a conclusion. This scenario, which is Type 2 Intrusion Detection, is similar to a patient going to a physician, who looks for symptoms and then makes a decision about whether or not the patient has a medical problem. The security technician also looks for symptoms and then makes a decision about whether or not an intrusion has occurred. Let $\mathcal{R}$ be the universal set of R and let I_2 represent a set of R that describes one or more symptoms of a Type 2 Intrusion. Then I_2 is the set:

$$I_2 = \{R | R \in \mathcal{R}, R \text{ involves a Type 2 Intrusion}\} \qquad (9)$$

Formula 9 formulates a Type 2 Intrusion. Let $\mathcal{I}_2$ be the universal set of all Type 2 intrusions. Now here is a formula for Type 2 Intrusion Detection, I_2^D:

$$I_2^D(L) = \begin{cases} \text{True}, & \exists I_2 \in \mathcal{I}_2 \colon\ I_2 \subseteq L \\ \text{False}, & \text{otherwise} \end{cases} \qquad (10)$$

The $I_2^D(L)$ function returns True if a symptom of an intrusion has been detected; otherwise it returns False. Possible examples of Type 2 intrusions are the following: The set of records consisting of a single local source IP address and numerous unique destination addresses all with a destination port of 445; the set of records consisting of a local IP address sending numerous e-mails during non-working hours; and, the set of records consisting of high volumes of UDP

traffic on high destination ports to a single local IP address matching criteria set by a Self-Organizing Map. Like a cough does not necessarily indicate a cold, the detection of an intrusion symptom does not always indicate an intrusion.

That was Type 2 intrusions and intrusion detection. Next is Type 3 intrusions, which are not detected in a given time period. Let $\mathscr{R}$ be the universal set of R and let I_3 represent a set of R that describes a Type 3 Intrusion. Then I_3 is the set:

$$I_3 = \{R | R \in \mathscr{R}, R \text{ involves a Type 3 Intrusion}\} \qquad (11)$$

$\mathscr{I}_3$ is the universal set of all I_3. A Type 3 Intrusion is not detected:

$$I_3^D(L) = \text{ False} \qquad (12)$$

As a summary, compare these three types of intrusion detection in a medical context to typhoid fever, which is spread by infected feces. Type 1 intrusion detection (prevention) is to wash one's hands after using the toilet; Type 2 intrusion detection is to recognize the symptoms, such as fever, stomach ache, and diarrhea; Type 3 detection is represented by Typhoid Mary, who had no readily recognizable symptoms.

The next step involves changing the data formats from R and L into forms which can be directly manipulated by analysis software. (Packet analysis can already occur directly on T.) This involves converting records into vectors and logs into matrices. This conversion is straightforward with a Detailed Input Data Vector, V_D, which starts as a set and is then used later as a location vector:

$$V_D \subseteq R \qquad (13)$$

More feature reduction can occur at this step. If the order of each element

in the set is fixed, i.e., if the order of the attributes of the set are fixed, then the set can become a location vector. An example of V_D as a set is {1280093999, 10.3.4.10, 10.3.4.12, 445, TCP} which could indicate a time stamp in seconds, a source IP address, a destination IP address, a destination port, and a protocol. Converting IP addresses to numerical formats, and assigning a numerical label to TCP, the same example of V_D as a location vector could be (1280093999, 167969802, 167969804, 445, 6).

Aggregate elements are also possible for a given time period, such as aggregate data for each local IP address for a day. Examples of such aggregate elements are the total number of R for the local IP address, the count of unique source IP addresses communicating with the local IP address, and the percentage of TCP network traffic for the local IP address. Many other types of aggregate elements are possible. The Imprecision Principle applies and experimentation is appropriate. These aggregate elements can be converted to an Aggregate Input Data Vector, V_A, with $f()$ being an aggregation function:

$$V_A = \{f_1(L), f_2(L), f_3(L), \ldots, f_{n_V}(L)\} \qquad (14)$$

where $n_V > 0$. Again, the order of the attributes of the set are fixed so that the set can become a location vector. An example of V_A as a set is {20100725, 428, 10.3.4.10, 48, 0.89} which could indicate that on 7/25/2010 428 unique source IP addresses attempted to contact destination IP address 10.3.4.10 on 48 unique destination ports with the TCP protocol being used 89 percent of the time. The date and IP address become a label for the location vector when the location vector is created. From the same example above, the location vector for IP

address 10.3.4.10 on 7/25/2010 is (428, 48, 0.89).

Both of these types of sets/vectors can be generalized as a General Input Data Vector, V:

$$V = V_D \text{ or } V = V_A \tag{15}$$

The next concept level is to generalize V so that it can be used as input to a wide variety of Soft Computer and other methods. The generalized elements of V are be represented by e. V is an n-tuple of real numbers which can be perceived, depending upon how it is intended as being used, as being a set, a location vector, or a matrix:

Set: $$V = \{e_1, e_2, e_3, \dots, e_{n_V}\} \tag{16}$$

Vector: $$V = (e_1, e_2, e_3, \dots, e_{n_V}) \tag{17}$$

Matrix: $$V = [e_1 \quad e_2 \quad e_3 \quad \cdots \quad e_{n_V}] \tag{18}$$

where $n_V \in \mathbb{N}$. For example, if the elements of V are an n-tuple of the real numbers 0.6, 0.5, 0.4, 0.3, 0.2, and 0.1, then V can be perceived as being a set, a vector or a matrix:

Set: $$V = \{0.6, 0.5, 0.4, 0.3, 0.2, 0.1\} \tag{19}$$

Vector: $$V = (0.6, 0.5, 0.4, 0.3, 0.2, 0.1) \tag{20}$$

Matrix: $$V = [0.6 \quad 0.5 \quad 0.4 \quad 0.3 \quad 0.2 \quad 0.1] \tag{21}$$

An Input Data Matrix, D, is a collection of similar types of V. Here D is represented as a set of V:

$$D = \{V_1, V_2, V_3, \dots, V_{n_D}\} \tag{22}$$

where $n_D \in \mathbb{N}$. D is on the same concept level as L—each can be considered to be sets of location vectors or a matrix. Here is how D can be represented as a

matrix:

$$D = \begin{bmatrix} V_{1,1} & \cdots & V_{1,n_V} \\ \vdots & \ddots & \vdots \\ V_{n_D,1} & \cdots & V_{n_D,n_V} \end{bmatrix} \tag{23}$$

where $n_D \in \mathbb{N}$ and $n_V \in \mathbb{N}$.

For example, given these three location vectors, each represented as a matrix,

$$V_1 = [0.6 \quad 0.5 \quad 0.4 \quad 0.3 \quad 0.2 \quad 0.1] \tag{24}$$

$$V_2 = [0.1 \quad 0.2 \quad 0.3 \quad 0.4 \quad 0.5 \quad 0.6] \tag{25}$$

$$V_3 = [0.9 \quad 0.8 \quad 0.7 \quad 0.6 \quad 0.5 \quad 0.4] \tag{26}$$

D would be represented this way as a matrix:

$$D = \begin{bmatrix} 0.6 & 0.5 & 0.4 & 0.3 & 0.2 & 0.1 \\ 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 \\ 0.9 & 0.8 & 0.7 & 0.6 & 0.5 & 0.4 \end{bmatrix} \tag{27}$$

D_D can refer to an Input Data Matrix consisting of V_D and D_A can refer to an Input Data Matrix consisting of V_A. D can also be one of these three types:

1. D_{Train} refers to a data set which is used to train the software intelligence.
2. D_{Test} refers to a data set which is used to test the software intelligence.
3. D_{Real} refers to feral data.

D can be used in virtually an infinite variety of analysis methods, from spreadsheet methods to statistics and data mining, to machine learning methods. For example, D_{Train} can be used by clustering software which, after testing, would then classify D_{Real} for intrusion detection.

The LLNIDS Computation Method more accurately defines information security concepts and scientifically ties components of information security together with structured and uniform data structures. The LLNIDS can be extended to describe existing and potential methodologies of analysis methods including statistics, data mining, AIS, NeuroFuzzy, Swarm Intelligence, and SOM, as well as Bayes Theory, Decision Trees, Dempster-Shafer Theory, Evolutionary Computing, Hidden Markov Models, and many other types of analysis.

DESIGNING THE HYBRID

Soft Computing components can be combined in virtually an infinite number of ways to create hybrid methods. The historical work of researchers has shown that different methods are better for different types of intrusions—there is no one *winner take all* method. The Imprecision Principle applies so that experimentation and intuition is appropriate. A general ensemble design which applies numerous methods at once appears to be the best solution. The LLNIDS Types of Intrusion help to focus the design on specific objectives, and the LLNIDS Computational Model provides a methodology of processing the data.

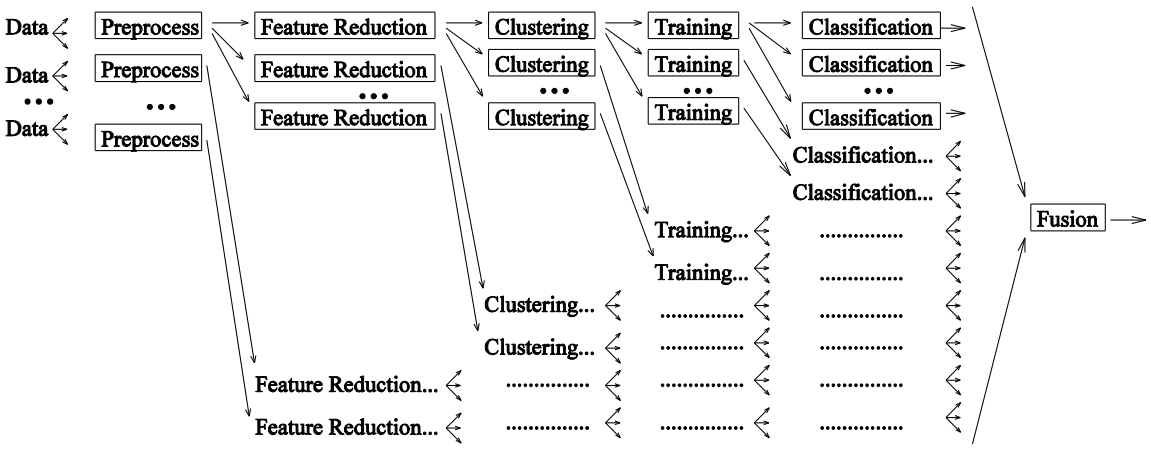


Figure 7, Hybrid Design

Figure 7 illustrates a framework of some of the uncountable possibilities of hybridization. Starting from the left, input data can come from one or numerous sources, including firewall logs, system logs, network flow data, packet analysis systems, and tcpdump. For each data source the data can be preprocessed (normalized) many different ways for the initial analysis. For each type of normalization, many alternative methods are possible for each additional step in the process from feature reduction to clustering, to training, and to classification. Several different methods are also possible for fusing the data. The overall number of possibilities is too great to try them all at once.

THE INVISIBLE MOBILE NETWORK BRIDGE (IMNB)

The Invisible Mobile Network Bridge (IMNB) was selected for data collection because it provided more information than firewall logs but was not as overwhelming as network flow data. ANNaBell used firewall logs as input with 20 million entries per day being typical. Network flow data would contain more information, but would be based on approximately 2 billion entries per day, an

amount more difficult to process. Host data cannot be trusted because experience has shown that current malware can exist outside of the operating system and can control intrusion detection software on the host. This research proposes a new source of network data produced from a device that is close to, but not located on the host, and that produces complete, but not an overwhelming amount, of information: an Invisible Mobile Network Bridge (IMNB).

The illustrations below show a laptop computer configured as an IMNB. *Invisible* means that the laptop has no IP address and is invisible on Layer 3 of the network. *Mobile* means that any computer or subnet traffic can be redirected through the laptop in real time without rebooting or reconfiguring any computers, routers, or switches. The laptop bridge can also remain running while Ethernet cables are physically switched and network traffic is redirected through it. The laptop bridge can then use various kinds of packet analysis software, such as PF [88], tcpdump [85], Snort [89], and Wireshark [90] to analyze the network traffic being redirected through it. This data can also be normalized as input vectors for more sophisticated analysis methods such as ANNaBell.

Figure 8, IMNB Desktop Insertion

Figure 8 shows the basic bridge setup with a laptop configured as the invisible mobile bridge. A blue cable is plugged into an extra NIC in the side of the bridge. The computer to be analyzed is the desktop which is connected to the Internet and running a YouTube [91] video. The Internet cable is unplugged from the desktop and plugged into the back of the mobile network bridge. The unplugged end of the blue cable is then plugged into the back of the desktop. The network connection is maintained and all of the desktop's network traffic is now travelling through the laptop configured as the invisible mobile network bridge. The YouTube video keeps running as though nothing has changed. The laptop can now capture network data and, after the intelligent software has been trained, act as an Intrusion Detection System (IDS) doing all types of analysis on the data, from packet sniffing to statistics, data mining, computational intelligence, and Soft Computing. Think of it as being like a heart monitor which is taking an EKG to show the health of the network connection.

Figure 9, IMNB Subnet Insertion

Figure 9 shows how the mobile bridge can analyze an entire subnet by rearranging switch cables. The pink cable (circled in left photo) originally goes from the switch out to the Internet. By plugging this pink cable into the back of

the laptop, and by plugging the blue cable from the laptop into the switch, network traffic for the entire subnet is redirected through the laptop acting as a network bridge. A YouTube video runs uninterrupted during the process, and users of the subnet need not even be aware that a change has been made. The laptop bridge can programmatically watch only certain computers on the network, if desired, or can also act as an IDS. Think of it as being like a heart monitor taking the EKG of all of the computers on the subnet at the same time. The bridge can be placed in network areas with the greatest potential impact.

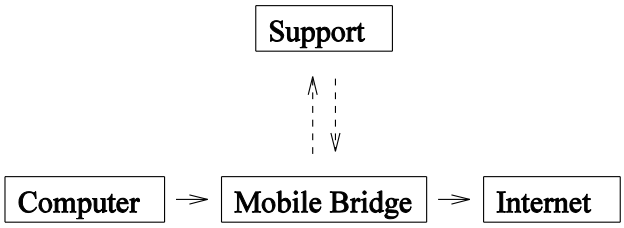


Figure 10, Basic IMNB Scenario

Many scenarios for using the mobile bridge are possible, and Figure 10 shows the main idea. A mobile bridge is placed as a *man in the middle* between a computer and the Internet, where it collects network data on the computer. This data can be observed on the mobile bridge in real time as it is being saved. See Figure 8 for a photograph of this scenario. The Mobile Bridge is *mobile* because this can be done many times with different computers with various scenarios (see more scenarios below), all without turning the mobile bridge or any of the computers off. The collected data is later transferred, by any of various ways, to a support computer, which then uses the data to train and test one or more modules containing various methods of computational intelligence and/or Soft Computing. The trained and tested modules are then transferred

back to mobile bridges, which then become intrusion detection systems using the trained and tested modules.

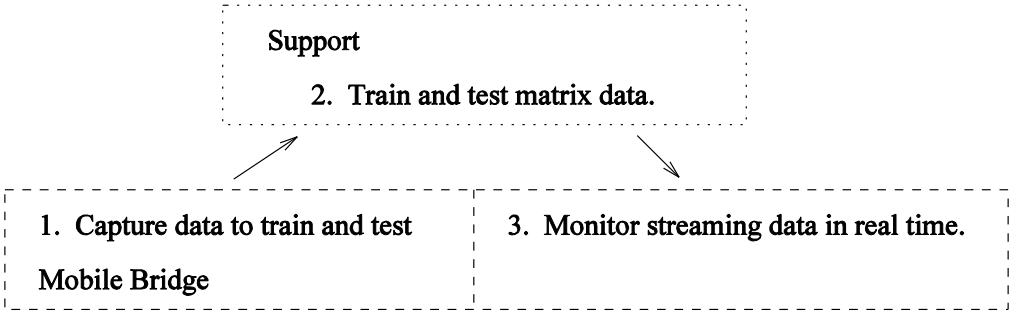


Figure 11, IMNB Symbolic Division of Labor

A summary of this division of labor is shown in Figure 11 as three simple steps: (1) one or more mobile bridges capture the data; (2) one or more support computers use the data to train and test intelligent software modules; and, (3) the modules are then used on mobile bridges for intrusion detection. The reason that separate support computers are needed is because the training can be lengthy and computationally intensive. Once the intelligent software modules are trained, though, they can be quickly and easily used by mobile bridges for real-time intrusion detection.

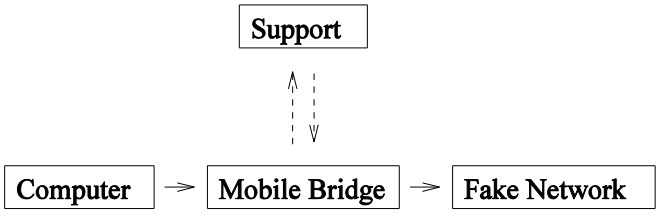


Figure 12, IMNB Forensics Scenario

Live on-the-spot forensics can be quickly accomplished simply by switching cables. The mobile bridge can also be inserted between a known infected computer and a simulated network as in Figure 12 for two major accomplishments: (1) the mobile bridge can capture network information from

known infected computers without endangering other computers on the Internet; and, (2) the mobile bridge can examine the network traffic of infected computers for forensics value. The first accomplishment is particularly noteworthy because the packet analysis can be used to create rules for Type 1 and/or Type 2 intrusion detection, and also because the network traffic of known infections can be used to train and test the intelligent software modules.

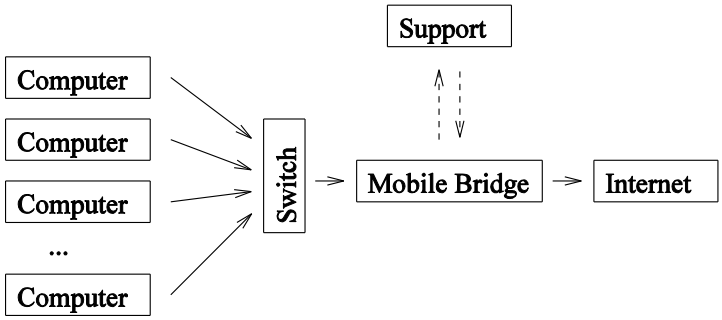


Figure 13, IMNB Subnet Scenario

The mobile bridge can also be placed behind a network switch (or hub) in order to collect data on, and monitor, an entire subnet as illustrated in Figure 13. The mobile bridge can be inserted live while the network is active simply by re-plugging cables. The mobile bridge could alternatively take a sample of each computer in turn to take a closer look at each one.

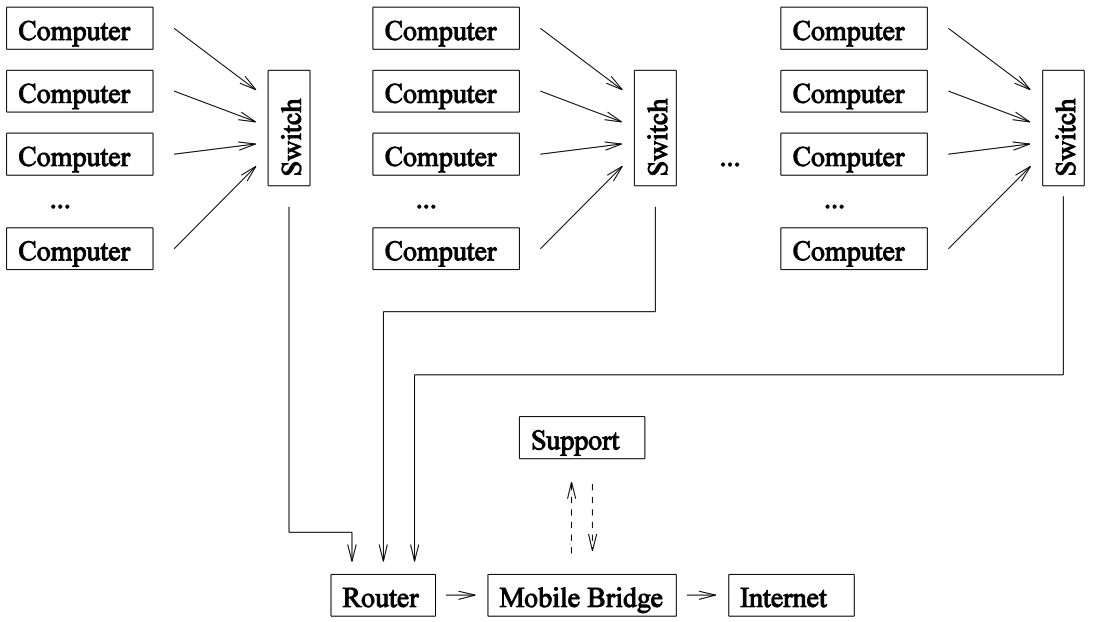


Figure 14, IMNB WAN Scenario

The mobile bridge can also be placed in a Wide Area Network (WAN) as shown in Figure 14. The only limitation is the hardware capability of the mobile bridge in terms of bandwidth, storage, and processing capability. The mobile bridge could alternatively cycle through each subnet or each computer for more detailed analysis.

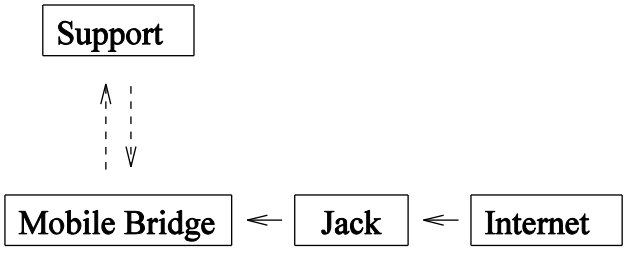


Figure 15, IMNB Jack Scenario

The mobile bridge can capture and monitor the traffic visible on a jack (an Ethernet wall plugin) as shown in Figure 15. Among other things, this scenario can tell if a switch port for a jack is properly limiting the traffic to that jack.

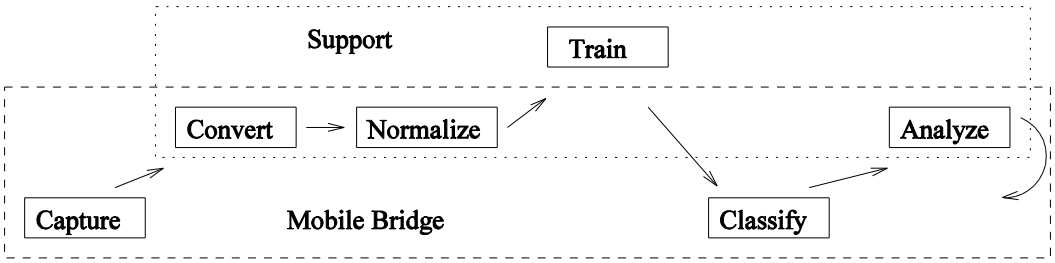


Figure 16, IMNB Task Distribution

The tasks are distributed as in Figure 16. Capturing must be done by a mobile bridge at the location of a computer or subnet. Converting and normalizing the data can be accomplished by either a mobile bridge or a support computer. The processing power of a support computer must be used to train the intelligent software modules. Classification of new data, i.e., *intrusion detection*, must be done by a mobile bridge on site. Further various types of analysis, such as fusion of intrusion alerts, can be done by either a mobile bridge or a support computer.

THE SOM COMPONENT

Since past history can be an indication of future performance, this research used a SOM based on ANNaBell as one of the components to be hybridized. The Imprecision Principle applies and intuition and experimentation are appropriate. Neural computing such as SOM is appropriate for data, such as network traffic, which is noisy and ill-defined [54].

SOM uses Hebbian [54] competitive learning which uses neighborhoods of nodes. The SOM input training data is not labeled, so the learning is unsupervised---the SOM clusters the data with no indication of what the data represents.

The SOM nodes have location vectors which must have the same number of dimensions as the input data. As previously discussed, the input data can be perceived on many levels, from a network transmission, to a set of elements, to a set of meta-data about the set of elements, to both the set of elements and meta-data about those elements, to a collection of sets of meta-data and elements, to vectors, to sets of vectors, to a matrix. Referring to the LLNIDS Computation Model discussed above, the input data is D_{I} and for SOM D is perceived as being a matrix. Each row of D is a V and every V will have the same size n_V, which is also the number of columns in D. Note that each V is commonly called a *vector* even though each V is also conceptually a row in the matrix D.

Each node in the SOM has the same structure as V in the input data--specifically, each node has n_V elements each of which is in the set of real numbers--and all of the nodes together in the SOM form a matrix which has the same number of columns as D. Since one of the objectives of SOM is to reduce the representation of the data space in the input data, the number of rows in the SOM matrix should be less than the number of rows in D. Let V_{N} represent a location vector for a node in the SOM so that V_{N} is analogous to V and let D_{N} represent the data matrix of all of the node location vectors in the SOM so that D_{N} is analogous to D. To easily differentiate between input data and node data, input data is designated D_{I} and node data is designated D_{N}. Likewise, an input vector is V_{I} and a node vector is V_{N}. The comparison of D_{I} and D_{N} is as follows:

$$D_{\mathrm{I}} = \begin{bmatrix} V_{\mathrm{I}_{1,1}} & \cdots & V_{\mathrm{I}_{1,n_V}} \\ \vdots & \ddots & \vdots \\ V_{\mathrm{I}_{n_D,1}} & \cdots & V_{\mathrm{I}_{n_D,n_V}} \end{bmatrix} \quad (28)$$

$$D_{\mathrm{N}} = \begin{bmatrix} V_{\mathrm{N}_{1,1}} & \cdots & V_{\mathrm{N}_{1,n_V}} \\ \vdots & \ddots & \vdots \\ V_{\mathrm{N}_{n_{\mathrm{N}},1}} & \cdots & V_{\mathrm{N}_{n_{\mathrm{N}},n_V}} \end{bmatrix} \quad (29)$$

where $n_D \in \mathbb{N}$, $n_V \in \mathbb{N}$, $n_{\mathrm{N}} \in \mathbb{N}$, and $n_{\mathrm{N}} < n_D$. Each node location vector in the SOM (V_N) may be initially created randomly or pseudo-randomly from $\mathbb{R}$ or from the domain of V from D_{TRAIN}. *Initiating the SOM* means to assign these initial values to the nodes.

BMN and Distances

The SOM methodology has three phases: training, testing, and production. The training phase uses nodes to cluster the training data; the testing phase tests the clusters with test data; and, the production phase classifies feral data using the clusters. The concept of a Best Matching Node (BMN) is central to SOM, both for clustering and classification. The Best Matching Node is sometimes called the *Best Matching Unit* or the *image*. BMN is also sometimes called *winner take all*, except this is not technically correct because neighbors of the *winner* also share in the *take*. This is explained further below.

For BMN, the concept level changes to a multidimensional solution space and each of the location vectors indicate a point in this solution space. The training phase of the SOM clusters node vector points in multidimensional space with each node typically representing a cluster. The nodes are typically also

clustered, resulting in clusters of clusters. The classification phase finds the BMN for a new input vector, thus indicating the cluster in multidimensional space which is closest to the new input location vector.

A distance measure is needed to determine the clusters and BMN in multidimensional space. Examples of possible distance measures are Euclidean distance and Manhattan distance. This research uses Euclidean distance, represented in this formula by d:

$$d_{V_I,V_N} = \sqrt{(V_I[1] - V_N[1])^2 + (V_I[2] - V_N[2])^2 + \cdots + (V_I[x] - V_N[x])^2} \qquad (30)$$

where x is the number of dimensions. The next formula uses the function $d(\,)$ as a distance measure to determine the BMN, V_{BMN}, for V_I:

Another kind of distance is also used in SOM which designates a neighborhood of nodes. Both types of distance are necessary in training the SOM. The SOM neighborhood is a topological configuration that is determined by the knowledge engineer before the SOM is trained. The Imprecision Principle applies to the determination of the configuration, which relies on the judgment of the knowledge engineer. The rational for the specific configuration for this research is explained further below.

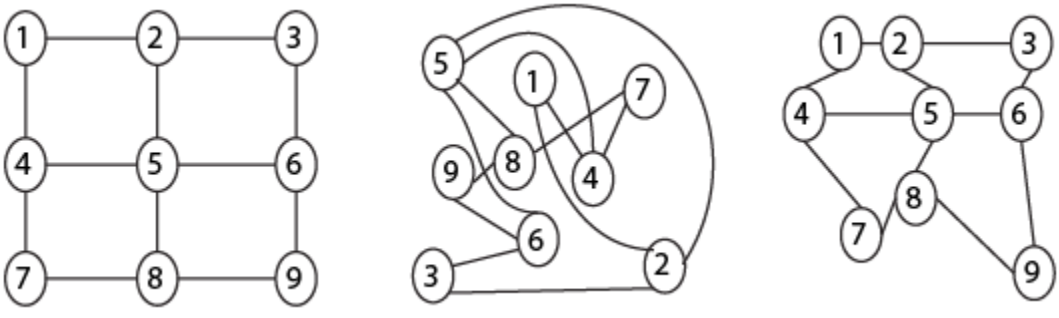


Figure 17, Neighborhood of Nodes

Figure 17 illustrates the SOM neighborhood for a sample configuration as a graph in three different ways. The concept level has changed from points in multidimensional space to a topological graph. The vertices of the graph represent the nodes of the SOM and the edges of the graph represent neighbors of the nodes. The first illustration, on the left, shows the topological configuration of 9 nodes in a rectangular format. Node 1, for example, has the neighbors 2 and 4. Node 5 has the neighbors 2, 4, 6, and 8. A neighborhood distance in a SOM is the number of edges between two nodes in the topological configuration. It is the number of *hops*, or *hop count*, using the edges between nodes. The neighborhood distance in the topological graph is different than the vector point distances in multidimensional space. Both of these two different types of distances are needed for the training of a SOM.

Node	1	2	3	4	5	6	7	8	9
1	0	1	2	1	2	3	2	3	4
2		0	1	2	1	2	3	2	3
3			0	3	2	1	4	3	2
4				0	1	2	1	2	3
5					0	1	2	1	2
6						0	3	2	1
7							0	1	2
8								0	1
9									0

Figure 18, Neighborhood Distances

Figure 18 shows the neighborhood distances between all of the nodes depicted in Figure 17. This can easily be verified by counting the edges between the nodes in the left graph of Figure 17. The middle graph of Figure 17 uses the same topological configuration as the left graph, but physically places the nodes

in two-dimensional space as though the node location vectors were created at random. It is possible in the middle graph of Figure 17, and in an actual SOM, for the vector points of two neighboring nodes to be far apart in the solution space. Nodes 2 and 5 in the middle graph of Figure 17, for example, are neighbors, yet their vector points are the furthest apart in space. Node location vectors change during training, and the right graph in Figure 17 shows an example representation of how a SOM neighborhood might appear after training: while not evenly spaced, the vector points of neighbor nodes tend to be relatively near each other. Note that the neighborhood distances in all three graphs in Figure 17 are the same, even though the vector points of the nodes have moved in space. Many other kinds of neighborhood configurations are possible besides the one shown in Figure 17. *The neighborhood of the BMN* means all of the nodes, including the BMN, within a given neighborhood distance of the BMN. *The neighborhood size* refers to the given neighborhood distance. For example, a neighborhood size of 2 means all of the nodes 2 edges or fewer away from the BMN, including the BMN, itself. A neighborhood size of 0 would consist only of the BMN. A neighborhood size begins large and shrinks during the training of the SOM.

The SOM is trained by repeatedly adjusting the locations of the node vectors as the neighborhood size gradually becomes smaller. Each V_{I} in D_{TRAIN} is considered in turn and all of the nodes in the current neighborhood of the BMN for the instant V_{N} are moved in vector space towards that V_{I}. The distance each node is moved, Δ, for each iteration of training is determined by the distance

measure, $d_{V_{\mathrm{I}},V_{\mathrm{N}}}$, and a training factor, α, which is explained below. The formula for how far to move a node for each input vector for each iteration of training is as follows:

$$\Delta V_{\mathrm{N}_j} = \begin{cases} \alpha d_{V_{\mathrm{I}_i},V_{\mathrm{N}_j}}, & \text{if } V_{\mathrm{N}_i} \text{ is in the neighborhood} \\ 0, & \text{if } V_{\mathrm{N}_i} \text{ is not in the neighborhood} \end{cases} \quad (31)$$

BMI and the Training Factor

Training factors can be determined many ways [54]. In this research, the training factor is determined at times by the number of Best Matching Inputs (BMI) in the current neighborhoods. BMI are the counterpoint to a BMN. If the BMN for a V_{I} is V_{N}, then that V_{I} is one of the BMI for that V_{N}. Note that BMN is referred to in this paper as being singular because there is only one BMN at a time, while BMI are referred to as being plural because there can be more than one best matching input at a time, even though the actual count of BMI might be 0 or 1. BMI have a many-to-one correspondence with a BMN.

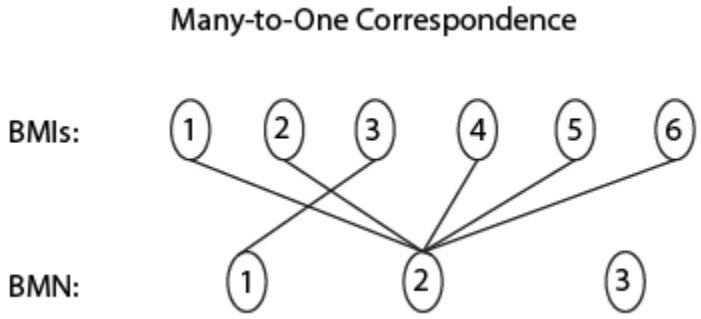


Figure 19, Best Matching Inputs

Figure 19 illustrates the many-to-one correspondence in an example with six input vectors (BMI) and three node vectors (BMN). An input vector always

has 1 BMN, but a node can have 0, 1, or more BMI. Here are the Best Matching Nodes for each input vector and the Best Matching Inputs for each node based on the Figure 19 example:

Table 6, BMN and BMI

Input Vector	BMN
1	2
2	2
3	1
4	2
5	2
6	2

Node	BMI
1	3
2	1,2,4,5, and 6
3	(None)

Table 6 shows the Best Matching Nodes and Best Matching Inputs from the example in Figure 19. Referring to the table, the Best Matching Node for input vector 1 is 2, while the Best Matching Inputs for Node 2 are 1, 2, 4, 5, and 6. The number of BMI for a node is important in the calculation of α. Let $|V_{N_i}|$ represent the count of BMI for Node *i*. Then in the example in Table 6, $\left|V_{N_1}\right| = 1$, $\left|V_{N_2}\right| = 5$, and $\left|V_{N_3}\right| = 0$.

The number of BMI for a neighborhood is also important. The previous illustration was for the number of BMI for a node, the next illustration is for the number of BMI for a neighborhood. First, the neighborhood for the example needs to be configured.

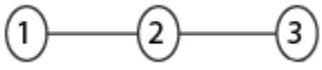


Figure 20, Neighborhood Example

Figure 20 shows a possible neighborhood configuration in order to continue the example from Figure 19. In this configuration, nodes 1 and 2 have a neighborhood distance of 1; nodes 2 and 3 have a neighborhood distance of 1; and, nodes 1 and 3 have a neighborhood distance of 2. Let $V_{\mathrm{N}_i\mathrm{NBH}_{s,t}}$ represent the neighborhood of Node i with a neighborhood size of s for training iteration (time) t. (Since multiple training iterations can be used for each neighborhood size, and the count of BMI can change between iterations, the iteration also needs to be indicated.) Then $|V_{\mathrm{N}_i\mathrm{NBH}_{s,t}}|$ represents the count of all of the BMI in that neighborhood for that iteration. For example, continuing the example from figures 19 and 20, for the current iteration $|V_{\mathrm{N}_3\mathrm{NBH}_1}| = 5$ because the neighborhood for Node 3 with a neighborhood size of 1 includes nodes 2 and 3 and the total number of BMI for nodes 2 and 3 is 5.

Table 7, Counts of Neighborhood BMI

	$d_{\mathrm{N}} = 0$	$d_{\mathrm{N}} = 1$	$d_{\mathrm{N}} = 2$
Node 1	1	6	6
Node 2	5	6	6
Node 3	0	5	6

Table 7 continues the example from figures 19 and 20 by showing all of the counts for all of the possible neighborhood sizes for each of the nodes. This table shows, for example, that the count for Node 3 with a neighborhood size of 0 is 0, with a neighborhood size of 1 is 5, and with a neighborhood size of 2 is 6.

The next important measure used in the calculation of α is the maximum count of BMI in the neighborhoods for a given neighborhood size. Let $|V_{\mathrm{N}_{\mathrm{MAX}}\mathrm{NBH}_{s,t}}|$ represent this measurement for the neighborhood size s and

training iteration t. Referring to Table 7, the maximum count for a neighborhood distance of 0 is 5, the maximum count for a neighborhood distance of 1 is 6, and the maximum count for a neighborhood distance of 2 is 6.

$$|V_{\mathrm{N_{MAX}NBH}_{s,t}}| = \arg\max_{i}\{|V_{\mathrm{N_i NBH}_{s,t}}|\} \tag{32}$$

The training factor α is calculated from $|V_{\mathrm{N_{MAX}NBH}_{s,t}}|$ and is as follows:

$$\alpha_{s,t} = \frac{1}{|V_{\mathrm{N_{MAX}NBH}_{s,t}}|} \tag{33}$$

A characteristic of determining the training factor this way is that BMN and BMI assignments must be determined during or prior to the first training iteration. One way of doing this is to have the initial neighborhood size include all of the nodes in the SOM. Keeping track of each BMN and the count of the BMI is also a way of tracking the progress of the training of the SOM: a slowing rate of change in these measurements is an indication of progress in the training of the SOM. Again, this is how the training factor is used in the SOM:

$$\Delta V_{\mathrm{N}_j} = \begin{cases} \alpha d_{V_{\mathrm{I}_i}, V_{\mathrm{N}_j}}, & \text{if } V_{\mathrm{N}_i} \text{ is in the neighborhood} \\ 0, & \text{if } V_{\mathrm{N}_i} \text{ is not in the neighborhood} \end{cases} \tag{34}$$

And this is how the change in the node vector is used in the training for one input vector for one iteration of training:

$$V_{\mathrm{N}\,j,(t+1)} = V_{\mathrm{N}\,j,t} + \Delta V_{\mathrm{N}_j} \tag{35}$$

An *epoch* is one iteration of training for the SOM. An epoch consists of looking at each of input vectors V_{I}, finding the BMN V_{BMN} for that vector, and adjusting the neighborhood node vectors $V_{\mathrm{BMN,NBH}}$ appropriately as described in the formulas above. If there are 65,536 input vectors, for example, as was the

case in ANNaBell, then one epoch consists of finding the BMN and adjusting the BMN neighborhood vectors for each of the 65,536 input vectors.

The knowledge engineer must determine how many epochs should be used to train the SOM, which is an Imprecision Principle issue. This can be decided in advance or it can be determined dynamically by monitoring changes in the movement of the nodes during the training. Deciding this in advance for large datasets is risky for a couple of reasons: 1) the training can last for days and one may not know when the training will end; and, 2) the nodes can move back and forth during training without converging on a solution.

Neighborhood Sizes

Another issue is determining what the neighborhood sizes will be for each epoch. The knowledge engineer determines the original (largest) neighborhood size for the first epoch, then the neighborhood sizes can get progressively smaller down to a neighborhood size of 0 for the last epoch. This can be done at a fixed rate for a fixed number of epochs, or it can be done dynamically. The dynamic method monitors node movement for successive epochs until the nodes acceptably converge. Then, the neighborhood size is reduced, and the further epochs are run until the nodes acceptably converge, again. This is repeated until the nodes acceptably converge at a neighborhood size of 0. What constitutes acceptable convergence is subject to the Imprecision Principle. Indications of convergence are the average amount of movement of the nodes for an epoch, the largest single movement of a node during an epoch, and the amount of change in BMI between two successive epochs. An issue to look for in

convergence is nodes that repeatedly move back and forth without converging.

1D ANNaBell and 3D ANNaBell started with a neighborhood size consisting of all of the nodes in the SOM. They could run a given number of epochs at the same neighborhood size, saving the state of each epoch, and stop, allowing the knowledge engineer to compare the states of successive epochs for convergence. More epochs could be run at the same neighborhood size or at a reduced neighborhood size, depending upon the appearance of convergence. This continues until the knowledge engineer is satisfied of convergence at a neighborhood size of 0. Here is the pseudo-code:

```
set epochs_to_run
set neighborhood_size
load previous state
for each epoch
  calculate the training factor
  for each input vector
    find the BMN
    move the BMN
    move the BMN neighbors
  save the state
```

Interpreting the Results

After training, the output of the SOM is in D_{N}, which contains the vector locations of all of the SOM nodes. The interpretation of this output data is the next major issue and this interpretation is subject to the Imprecision Principle. Many ways exist of interpreting the SOM and in a large sense understanding the SOM output is a data mining problem. One way is to graphically plot the distances between neighboring nodes resulting in a display which shows clusters of nodes. Another way is to call each dimension of the vectors a *feature*, and to data mine relationships between features in the nodes of the SOM. The concept

level has changed to features representing different dimensions of the vectors. Yet another way is to label the types of input BMI per node and look for patterns of these BMI. Each node can be labeled this way depending upon the types of BMI that it has. In this research, for example, each input vector represents a single local IP address. These input vectors can therefore be labeled with a type of network user or usage, such as student, faculty, and staff users or desktop, wireless, or server usage. Subnets representing departments can also be labeled, as can individual computers. The result can be a fingerprint of the kinds of activity for that department.

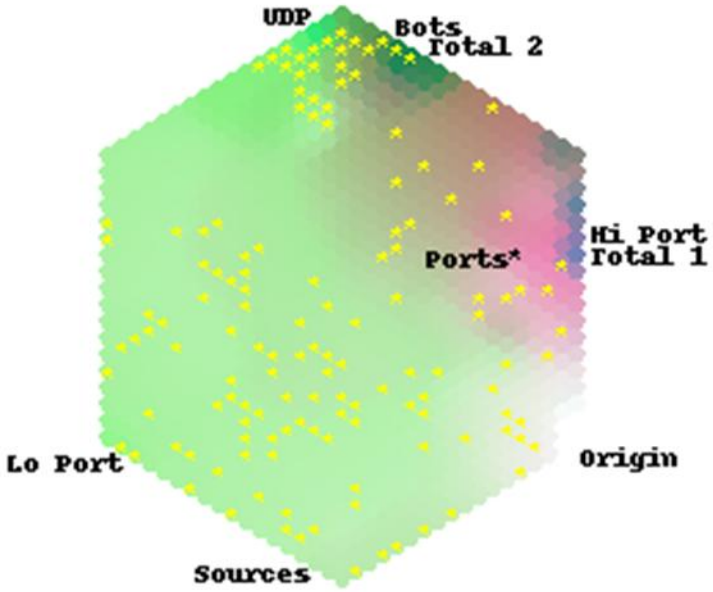


Figure 21, Fingerprint of a Department

Figure 21 plots yellow asterisks for IP addresses for a department on the BMN vectors on the SOM topological map resulting in a fingerprint of the network activity for that department. This profiling can be a model of risk or used as

symptoms or danger signals. The labeled features on this map include the percentage of UDP traffic, the total number of entries in a log file, the highest port accessed, the lowest port accessed, the number of unique ports accessed, and the number of unique source IP addresses in the traffic. The origin is the location of the BMN for IP addresses with no log entries. The background color is a blend of red, green, and blue colors for the intensities of some of the features.

From a node point of view, the BMI of a node might primarily represent student computer activity, so that node could be labeled *student*. From a usage point of view, the BMN vector can be located for the vector of each IP address in a department.

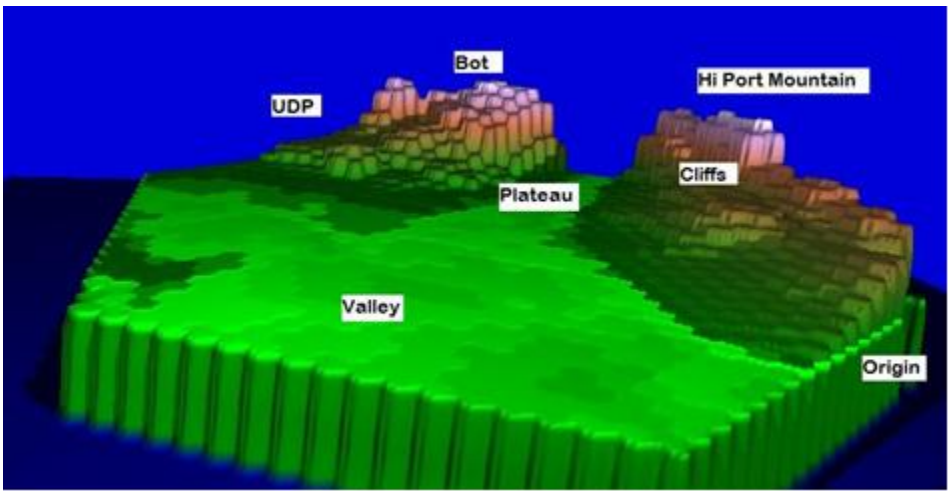


Figure 22, ANNaBell Island

Figure 22 is an example of a higher level feature map, created with Art of Illusion [92], from 3D ANNaBell. Landscape features such as valley, plateau, and mountain, developed from the SOM nodes, represent types of network traffic. The concept level has changed from features of vector dimensions to landscape features representing groups of vector dimensions. The valley

landscape feature, for example, consists of nodes with vector dimensions of very low total number of log entries, low ratios of unique source IP addresses, very high ratios of port ratios, medium low ports, high high ports and low UDP ratios. Each small area of the feature map represents a node of the SOM based on the topological map, and any IP address can be plotted on this map by locating its BMN. The neighbors on the feature map are based on the topological map, while the colors and elevations are based on intensities of vector elements. An IP address with a BMN in the valley area, for example, indicates normal office network traffic for that IP address. An IP address with a BMN in a mountainous area, depending upon the specific location, might warn of dangerous network traffic.

The testing of the SOM is a classification problem. New location vectors from D_{TEST} are used to find the corresponding BMN for each and plot the location on the feature map. Then, the kind of network traffic of the test vector is compared with the kind of network traffic for the BMI of the BMN to see if they are similar, or not. A test vector from a sample of malware, for example, should have a BMN in a *mountainous* area, while a test vector from an office computer should have a BMN in the *valley*. The results of the test should give some indication of the reliability of the SOM. Since testing only compares each input vector with each node once, it can be done extremely fast.

The SOM in production is also a classification problem. Vectors from feral network traffic are matched with the corresponding BMN for each. If the BMN for an input vector is in a dangerous area of the feature map, then an alarm, or

danger signal, is produced. Since a SOM in production only compares each input vector with each node once, it can be done extremely fast.

The 1D ANNaBell had a single node, 996, which indicated malicious or vulnerable traffic, and which made alarms and follow up response straightforward. That node is outdated, though, now, and so is no longer useful for two reasons: the network traffic of Storm Worm changed, and other computers with this characteristic were handled by incident responders until there were none left. 3D ANNaBell had ambiguous areas of the feature map which indicated danger areas, and so the alerts were not as straightforward. This ambiguity led to a Fuzzy Inference System (FIS).

THE FIS COMPONENT

A Fuzzy Inference System (FIS) is a classifier that helps to cope with inexact descriptions of data. This research adapts FIS to send danger signals based on SOM results. A visual examination of ANNaBell Island shows 7 general areas on the map, but many of the boundaries of these areas are indistinct.

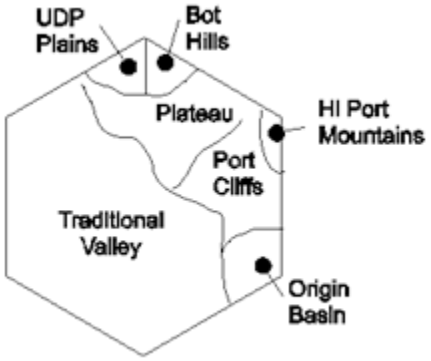


Figure 23, Fuzzy Divisions of SOM Topological Map

Figure 23 is a drawing of the 7 general landscape areas of ANNaBell Island. These 7 areas represent different mixes of network activity involving number of log entries, number of source IP addresses, ratio of UDP traffic and other indicators. The IP addresses with malicious traffic, for example, have tended to have a BMN in the Bot Hills area. The UDP Plains tend to represent student computers—the IP address of any office computer with a BMN in the UDP Plains area is suspect. Locating the BMN of an IP address on the map is helpful in profiling computers, but the areas of the map are indistinct, so FIS becomes beneficial in analyzing SOM output. 3D ANNaBell was converted into FIS by observing the colorized nodes in the map to determine general areas, and then by examining and comparing the intensities of the node vector values for each dimension. The concept level has changed from landscape features to fuzzy inference. The fuzzy values for the Traditional Valley, for example are total entries, very low; ratio of unique source addresses, high; ratio of unique destination ports, medium; lowest port, mixed; highest port, low; and, UDP ratio, medium.

Table 8, 3D ANNaBell Fuzzy Values

	Total Normalized	Source Ratio	Port Ratio	Lowest Port	Highest Port	UDP Ratio
UDP	Mixed	Low	Low	Somewhat High	Low	Very High
Bot	High	Low	Low	Medium	High	High
Plateau	Very Low	High	Medium	High	High	Somewhat High
Hi Port	Extremely High	Low	Low	Medium	Very High	Low
Origin	Very Low	Very Low	Very Low	Very Low	Very Low	Very Low
Traditional	Very Low	High	Medium	Mixed	Low	Medium
Port	Very Low	Low	Very High	Medium	High	Low

Table 8 shows all of the fuzzy values for all seven landscape feature areas of ANNaBell Island. These were determined by looking at the feature maps,

getting the location vectors for the nodes in each area, and assigning fuzzy values to the intensities of the values for each dimension for the nodes in each area. If network traffic for an IP address had a high amount of entries, a low ratio of source IP addresses, a low ratio of unique ports, a medium lowest port, a high highest port, and a high UDP ratio, then that traffic would have a fuzzy match with the traffic for BMI in the Bot Hills area of the feature map. In other words, the network traffic had similarities to malware traffic.

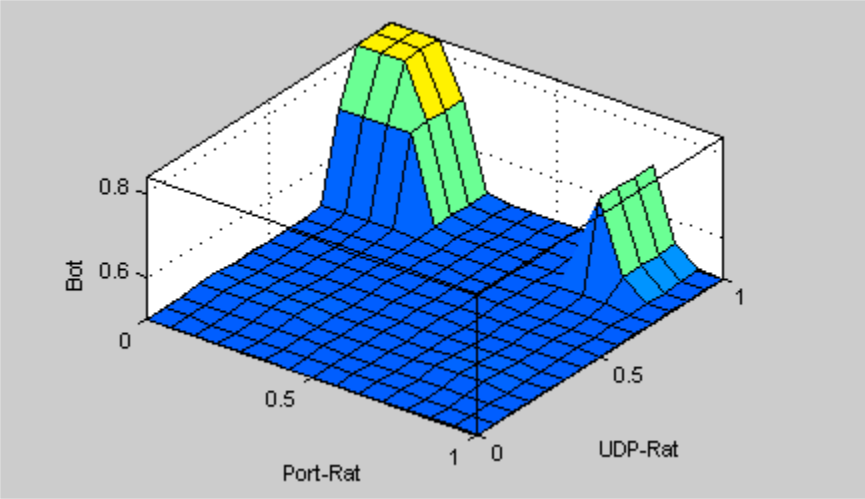


Figure 24, Matlab Illustration

Figure 24 is a Matlab illustration of the fuzzy values of the ratio of unique ports and the UDP ratio compared to the Bot Hills area of the feature map.

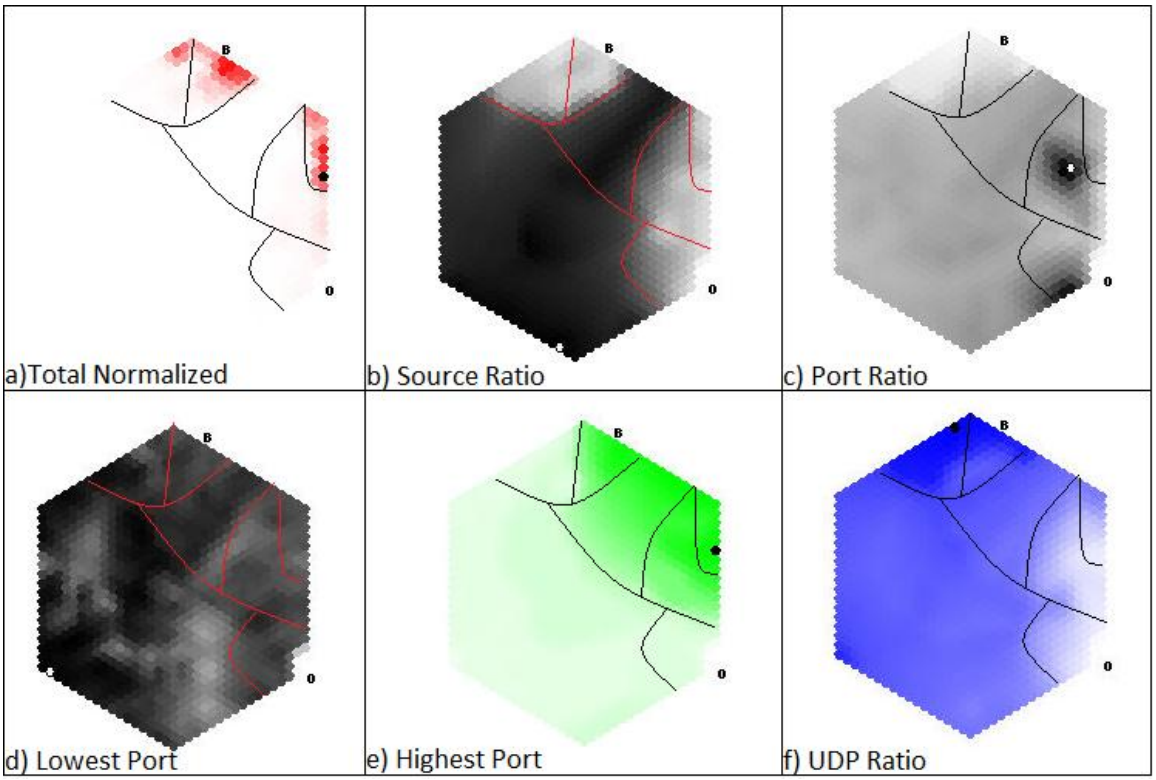


Figure 25, Intensities of Features

Figure 25 shows the intensities for each of the dimension values for the 3D ANNaBell location vectors which were used to create Table 8. The landscape feature overlays are derived from Figure 21 and are similar to the drawing in Figure 23. Looking at the Traditional Valley area of each part of Figure 25, for example, one can see that the total normalized is very low; the source ratio is high; the port ratio is medium; the lowest port is mixed; the highest port is low, and the UDP ratio is medium. Judgments of boundaries and intensities may vary somewhat because of the Imprecision Principle. The splotchiness of the intensities for d) Lowest Port indicate that this feature was not very useful in determining relevant types of network traffic.

Fuzzy rules can be created from this information. Here is an example for determining if a type of network traffic is similar to the type of traffic represented

by the Bot Hills, in which case an alert should be produced:

```
IF   Total Normalized is High
AND  Source Ratio is Low
AND  Port Ratio is Low
AND  Lowest Port is Medium
AND  Highest Port is High
AND  UDP Ratio is High
THEN Alert is High
```

Crisp output can be derived from the fuzzy inferences using Table 8. Suppose, for example, that network traffic for a certain local IP address for a 24-hour period had 1,574, 557 firewall log entries. Using the procedures for 3D ANNaBell, this would be normalized to 0.779349187. Suppose these entries came from 1,512,381 unique source addresses, which would be a 0.96051214 source ratio. Also, these entries targeted 956,961 unique ports, for a port ratio of 0.607765017. The lowest port attempted was 24,439 (0.372914079 normalized) and the highest port attempted was 43,940 (0.670486862 normalized). The UDP ratio was 0.546359486. The linguistic variables from Table 8 are Low (L), Very Low (VL), Medium (M), Somewhat High (SH), High (H), Very High (VH), and Extremely High (XH). Below are graphs illustrating how to get crisp output for the data from this example. Sugeno-style inference is used in this example for computational efficiency [93].

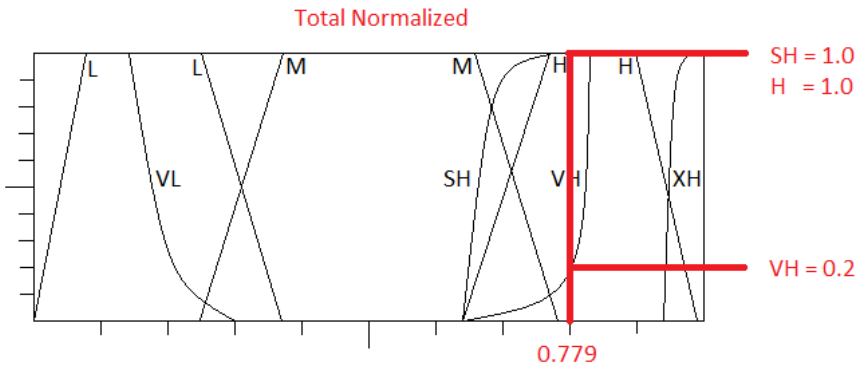


Figure 26, Total Normalized Graph

Figure 26 shows the conversion to linguistic variables for the total normalized dimension. The input is shown rounded to 0.779, which converts to a fuzzy membership of 1.0 for Somewhat High; a fuzzy membership of 1.0 for High; and, a fuzzy membership of 0.2 for Very High. The zero memberships are not shown in the figure: Low, 0; Very Low, 0; Medium, 0; and, Extra High, 0. These output values will be used further below.

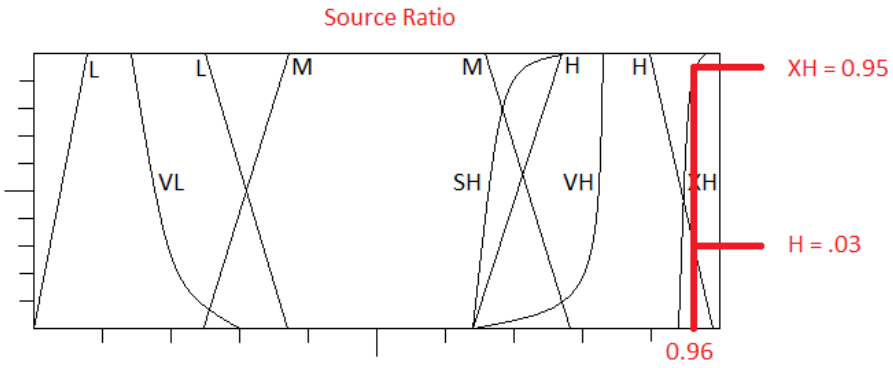


Figure 27, Source Ratio Graph

Figure 27 shows the conversion for the source ratio dimension. The input is shown rounded to 0.96. The non-zero outputs are Extra High, 0.95; and, High, 0.03.

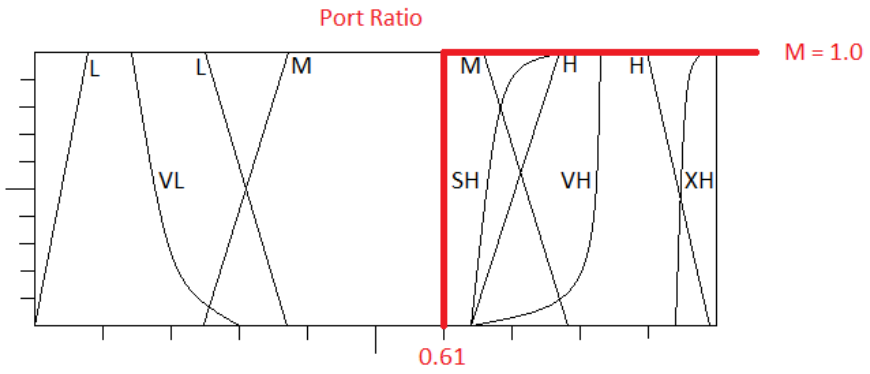


Figure 28, Port Ratio Graph

Figure 28 shows the conversion for the port ratio dimension with the input shown rounded to 0.61. The single non-zero output is Medium, 1.0.

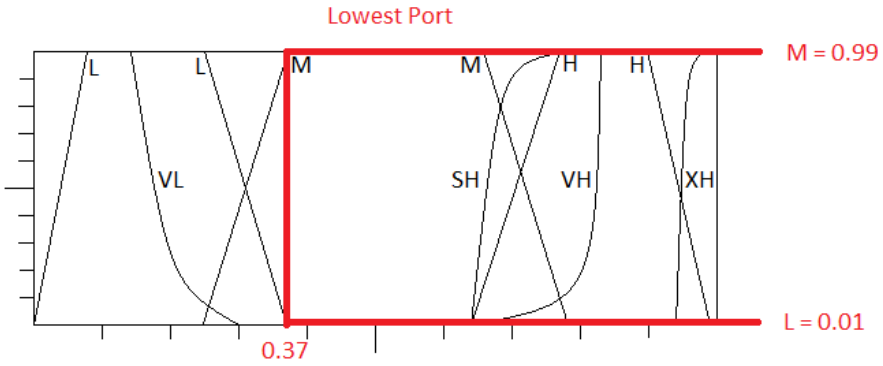


Figure 29, Lowest Port Graph

Figure 29 shows the conversion for the lowest port dimension with the input shown rounded to 0.37. The non-zero outputs are Medium, 0.99; and, Low, 0.01.

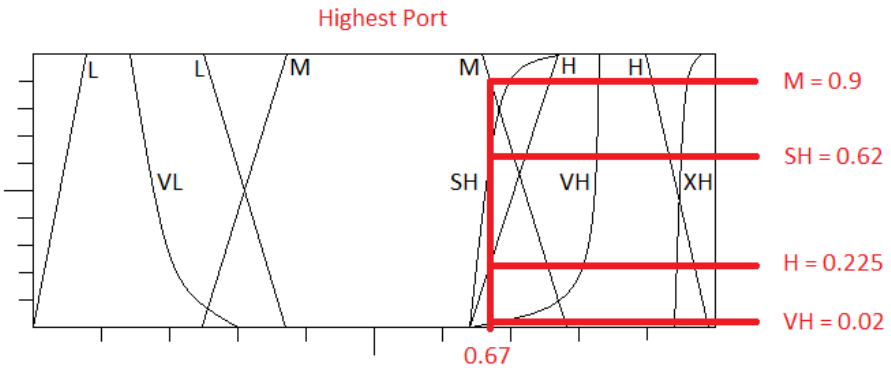


Figure 30, Highest Port Graph

Figure 30 shows the conversion for the highest port dimension with the input shown rounded to 0.67. The non-zero outputs are Medium, 0.9; Somewhat High, 0.62; High, 0.225; and, Very High, 0.02.

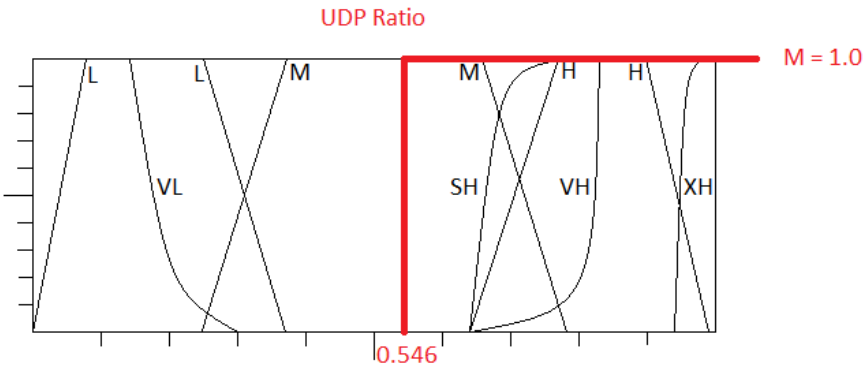


Figure 31, UDP Ratio Graph

Figure 31 shows the conversion for the UDP ratio dimension with the input shown rounded to 0.546. The only non-zero output is Medium, 1.0.

Table 9, Crisp Outputs

	Total Normalized	Source Ratio	Port Ratio	Lowest Port	Highest Port	UDP Ratio	Crisp Output
UDP Plains	1.000	0.000	0.000	0.000	0.000	0.000	0.167
Bot Hills	1.000	0.000	0.000	0.990	0.225	0.000	0.369
Plateau	0.000	0.030	1.000	0.000	0.225	0.000	0.209
Hi Port Mountains	0.000	0.000	0.000	0.990	0.020	0.000	0.168
Origin Basin	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Traditional Valley	0.000	0.030	1.000	0.000	0.000	1.000	0.338
Port Cliffs	0.000	0.000	0.000	0.990	0.225	0.000	0.203

Table 9 orders the outputs from the six previous graphs and displays the calculated crisp outputs for each of the landscape features. Notice the similarity of Table 9 with Table 8: they are the same except fuzzy memberships are substituted for linguistic variables. Table 9 adds an additional column on the right which shows the calculated crisp outputs. These crisp outputs are determined with Sugeno-style inference where all of the weights are 1.0 for the simplicity of illustration, which results in each crisp output simply being the average of the values for each row. Since the linguistic variables overlap for the various landscape features, the total of the crisp outputs do not necessarily add up to 1.

The danger signals come from the crisp outputs, each of which can range from 0 to 1. Since the Bot Hills is a known bad area, the crisp output for that area, 0.69, is a danger signal. Safe signals are also possible. Since the Origin Basin and the Traditional Valley are known safe areas, their crisp outputs of 0.000 and 0.338 are safe signals. The danger signals of the other landscape areas depend upon the type of computer being evaluated. The crisp output of

0.167 for the UDP plains, for example, would be a danger signal for an office computer, but would not be relevant for a student computer. Ways of fusing these danger signals to obtain a consensus are the subjects of the next two topics: SJ Fusion and AISDT.

THE SJ FUSION COMPONENT

The most robust method for fusing intrusion alarms found in the literature was a form of subjective logic explained by Svensson and Josang [49]. A portion of that work concerns obtaining a consensus of possibly conflicting and uncertain opinions. In order to maintain linguistic consistency, this consensus is called *fusion* and this particular type of fusion is called *SJ Fusion*, for the authors of this paper. All of the formulas in this section are from that source. Characteristics of SJ Fusion are that it is based on belief theory operating on uncertain beliefs about crisp propositions. It is appropriate for intrusion detection because an intrusion either has taken place or it has not (it is crisp), and beliefs about an intrusion can have varying degrees of certainty. In the current context, danger signals have varying degrees of certainty. The concept level is changing from crisp output from FIS, to uncertain danger signals of an intrusion. SJ Fusion will be described, then the example from Table 9 will continue.

Let $\Theta = \{x, \bar{x}\}$ be a state space containing x and its complement $\bar{x}$. In the current context, *x* represents an intrusion and $\bar{x}$ represents the lack of an intrusion. These variable names are pertinent in SJ Fusion:

- b Belief. b_x Is the belief in x, from 0 to 1.
- d Disbelief. d_x Is the disbelief in x, from 0 to 1.

- u Uncertainty. u_x Is the uncertainty in the belief of x, from 0 to 1.
- a Atomicity. a_x Is the atomicity setting for x. Atomicity is a variable to allow the knowledge engineer to manipulate uncertainty in the output, from 0 to 1. The neutral setting is 0.5.

This restriction applies:

$$b_x + d_x + u_x = 1 \tag{36}$$

The combination of these variables is called the *opinion* of x and is the tuple:

$$\omega_x \equiv (b_x, d_x, u_x, a_x) \tag{37}$$

The probability expectation for the truth of x is:

$$E(\omega_x) \equiv b_x + a_x u_x \tag{38}$$

One can see in the above formula how the atomicity a splits the probability for uncertainty u. An atomicity value of 0.5 evenly splits the uncertainty probability. The knowledge engineer sets the atomicity value and a value of 0.5 is used in this research.

Some of the crisp outputs for the landscape features in Table 9 support the truth of an intrusion (danger) and some support the untruth of an intrusion (safety). In all cases, The Bot Hills support the truth of an intrusion and the Traditional Valley and Origin Basin support the untruth of an intrusion. The other landscape features depend upon the type of computer, such as a student computer, or an office computer, which is being evaluated. For an office computer, the UDP Plains, Plateau, Hi Port Mountains, and Port Cliffs also

support the truth of an intrusion. This information sets up the following SJ Fusion opinions with numbers taken from the crisp output of Table 9:

Table 10, SJ Fusion Opinions

	b	d	u	a
UDP Plains	0.167	0.000	0.833	0.500
Bot Hills	0.369	0.000	0.631	0.500
Plateau	0.209	0.000	0.791	0.500
Hi Port Mountains	0.168	0.000	0.832	0.500
Origin Basin	0.000	0.000	1.000	0.500
Traditional Valley	0.000	0.338	0.662	0.500
Port Cliffs	0.203	0.000	0.797	0.500

These varying opinions in Table 10 can not only be fused into a single opinion, but can be done so with discounting. Suppose that for office computers the UDP Plains, the Plateau, the Hi Port Mountains and the Port Cliffs are indicators of danger, but not to the same extent as the Bot Hills are an indicator of danger. These opinions can be discounted by opinions of opinions. Let A and B be two agents where $\omega_B^A = (b_B^A, d_B^A, u_B^A, a_B^A)$ is A's opinion of B as an advice provider and $\omega_x^B = (b_x^B, d_x^B, u_x^B, a_x^B)$ is B's opinion of x. Then, $\omega_x^{AB} = (b_x^{AB}, d_x^{AB}, u_x^{AB}, a_x^{AB})$ is the discounted opinion such that:

1. $b_x^{AB} = b_B^A b_x^B$
2. $d_x^{AB} = b_B^A d_x^B$
3. $u_x^{AB} = d_B^A + u_B^A + b_B^A u_x^B$
4. $a_x^{AB} = a_x^B$

The symbol $\otimes$ represents this operation so that $\omega_x^{AB} = \omega_B^A \otimes \omega_x^B$. This

operation $\otimes$ is associative but not commutative so that in a chain of opinions, the discounting can start at either end, but the order of opinions is significant. To continue the example from Table 9, if the knowledge engineer's opinion of the danger signals of the UDP Plains, the Plateau, the Hi Port Mountains and the Port Cliffs is (0.5, 0, 0.5, 0.5), then the discounted danger signals of these landscape features are:

Table 11, Discounted Opinions

	b	*d*	*u*	*a*
UDP Plains	0.084	0.000	0.917	0.500
Plateau	0.105	0.000	0.896	0.500
Hi Port Mountains	0.084	0.000	0.916	0.500
Port Cliffs	0.102	0.000	0.899	0.500

A fused consensus can be made of the opinions using discounted opinions when appropriate. Let $\omega_x^A = (b_x^A, d_x^A, u_x^A, a_x^A)$ and $\omega_x^B = (b_x^B, d_x^B, u_x^B, a_x^B)$ be opinions respectively held by agents A and B and let $k = u_x^A + u_x^B - u_x^A u_x^B$. Let $\omega_x^{A,B} = (b_x^{A,B}, d_x^{A,B}, u_x^{A,B}, a_x^{A,B})$ be the opinion such that:

$$1.\ b_x^{A,B} = \begin{cases} \frac{b_x^A + b_x^B}{2}, & \text{if k=0} \\ \frac{b_x^A u_x^B + b_x^B u_x^A}{k}, & \text{if k}\neq 0 \end{cases}$$

$$2.\ d_x^{A,B} = \begin{cases} \frac{d_x^A + d_x^B}{2}, & \text{if k=0} \\ \frac{d_x^A u_x^B + d_x^B u_x^A}{k}, & \text{if k}\neq 0 \end{cases}$$

$$3.\ u_x^{A,B} = \begin{cases} 0, & \text{if k=0} \\ \frac{u_x^A u_x^B}{k}, & \text{if k}\neq 0 \end{cases}$$

$$4.\ a_x^{A,B} = \begin{cases} \frac{a_x^A + a_x^B}{2}, & \text{if k=0} \\ \frac{a_x^B u_x^A + a_x^A u_x^B - (a_x^A + a_x^B) u_x^A u_x^B}{k}, & \text{if k}\neq 0 \end{cases}$$

Then, $\omega_x^{A,B}$ is the consensus between ω_x^A and ω_x^B representing an imaginary agent *[A,B]*'s opinion as if that agent represented both A and B. By using the symbol $\oplus$ to designate this operator, $\omega_x^{A,B} = \omega_x^A \oplus \omega_x^B$. This consensus operator is both commutative and associative. The fusion/consensus is done by starting with two of the landscape feature opinions to create an interim consensus. A third landscape feature opinion is then fused with the interim consensus to create a new interim consensus. This is repeated with each additional appropriate landscape feature until a final consensus is reached.

Table 12, Office Computer Consensus

	b	*d*	*u*	*a*	*k*
UDP Plains dis.	0.084	0.000	0.917	0.500	
Bot Hills	0.369	0.000	0.631	0.500	0.969
Interim A	**0.404**	0.000	0.597	0.500	
Plateau dis.	0.105	0.000	0.896	0.500	0.958
Interim B	**0.443**	0.000	0.558	0.500	
Hi Port Mtn. dis.	0.084	0.000	0.916	0.500	0.963
Interim C	**0.470**	0.000	0.531	0.500	
Origin Basin	0.000	0.000	1.000	0.500	1.000
Interim D	**0.470**	0.000	0.531	0.500	
Traditional Valley	0.000	0.338	0.662	0.500	0.841
Interim E	**0.370**	0.213	0.418	0.500	
Port Cliffs dis.	0.102	0.000	0.899	0.500	0.941
Consensus	**0.399**	0.204	0.399	0.500	

Table 12 shows the line-by-line calculated consensus for an office computer for the example from Table 9, using discounted opinions where appropriate. The significance of the *office computer* IP address designation is that the opinions/danger signals from the UDP Plains discounted, Plateau discounted, Hi Port Mountains discounted, and Port Cliffs discounted are included in the consensus, which would not be the case for a student computer IP address. The calculations were performed two lines at a time from top to bottom. For example, the consensus for UDP Plains discounted and Bot Hills is Interim A; the consensus for Interim A and Plateau discounted is Interim B; and so forth, until the overall consensus is shown on the bottom line. One can follow the changes as the interim danger signals, highlighted in bold red, are calculated from the top down: 0.404, 0.443, 0.470, 0.470, and 0.370. The result for the final consensus/danger signal is on the bottom line: 0.399—this is the fused danger signal which is the output that is passed on to the Artificial Immune System (AIS), covered in the next section. The calculation for a student computer does not include the opinions of all of the landscape features because students are allowed peer-to-peer activity, online gaming, and other activities which are not appropriate for most office computers.

Table 13, Student Computer Consensus

	b	*d*	*u*	*a*	*k*
Traditional Valley	0.000	0.338	0.662	0.500	
Bot Hills	0.369	0.000	0.631	0.500	0.875
Consensus	**0.279**	0.244	0.477	0.500	

Table 13 shows the consensus for the same example as though it were for

the IP address of a student computer. The Origin Basin opinion/danger signal was omitted because it had no effect in this example, having values of zero for both belief and disbelief. Compare the final consensus/danger signal for the office computer, 0.399 with the final consensus/danger signal for the student computer, 0.279. They are different even though the same values were used because student computers are expected to have more variety of network activity than office computers.

The terminology changes somewhat going into the next section. *Danger Signal* is used exclusively instead of *consensus*. The Danger Signal is the result of the consensus for the beliefs of an intrusion, derived from the opinions taken from the values for the landscape features. For clarification, the SJ fused Danger Signal for the example of the office computer is 0.399 and the SJ fused Danger Signal for the example of the student computer is 0.279. Whether these numbers represent low or high levels of danger is not readily apparent. One way of resolving this would be to derive many danger signals and compare them. Any possible significance of the disbelief in danger in these examples is not apparent. Also, suppose that Snort fired a low alert on the same IP address involved with these danger signals. How would this Snort alarm be figured in? The Dendritic Cell Algorithm (DCA) explained further below addresses these issues.

THE AIS DANGER THEORY COMPONENT

Danger signals from the SOM and FIS components integrate naturally into Artificial Immune System Danger Theory (AISDT). AIS is based on the exceedingly complex human immune system, which is summarized here only to

the extent to provide some context for AISDT. Immunology is a developing and controversial field of study in which experts sometimes define the same terms somewhat differently. This paper seeks only to summarize and paraphrase key biological concepts as they are related to AIS Danger Theory, so see immunology texts for differing immunology viewpoints and more technical information than what is presented in this paper.

Human Immunity

The human immune system has two descriptive systems, innate and adaptive, that sometimes act together [94]. The innate immune system is non-specific, fast, has no memory, and usually is of short duration. The adaptive immune system is specific, slower in development, has memory, and is long lasting. Both systems have mechanisms for distinguishing self from non-self. [94]

Objects associated with the innate immune system, but which may also interact with the adaptive system, include skin, cytokines, natural killer cells, macrophages, and dendritic cells [94]. Dendritic cells have been known since 1973 but their importance in both innate and adaptive immunity has only recently been more clearly defined. Dendritic cells develop in the bone marrow and then circulate in the blood and some tissues where they play a role in surveillance. (They are named for their branched projections which are similar to dendrites of neurons, but a dendritic cell only has this appearance and is not a neuron.) When dendritic cells are activated they become mature and are the only cells capable of activating T-cells. Dendritic cells are activated upon exposure to

danger signals, such as cytokines, and Pathogen-Associated Molecular Patterns (PAMPs). When activated, dendritic cells migrate to lymph nodes where they present the associated antigen to T-cells and help to define the T-cell response. [94]

The adaptive immune system has two general systems: humoral immunity and cell-mediated immunity [94]. Humoral immunity is found in extra-cellular body fluid or serum, and is mediated by antibodies produced by B-cells. The complement of an antibody is an antigen, which is a substance or molecule that is a possible invader and that triggers the production of antibodies. This is sometimes referred to as a self/non-self system with antibodies being a part of the *self* which looks for antigens which are conceptually the *non-self*. [95] Antibodies find antigens by binding with them. Binding (also called *matching* or *fitting*) occurs at a binding site, which is complementary in size, shape, charge, and hydrophobic or hydrophilic character [96]. The part of the antigen which matches an antibody is called the *epitope*. Self/non-self immunity protection has a direct comparison with anomaly detection in information security: determine what is normal (self), and then look for what is not normal (non-self). [94]

Cell-mediated immunity involves T-cells, which originate in bone marrow and develop in the thymus, after which they enter the blood stream. [94] T-cells are adept at identifying and killing cells that have been infected by pathogens. Helper T-cells regulate the humoral immune system and cytotoxic (killer) T-cells destroy infected cells.

Table 14, Comparison of Immune Systems

Innate	Adaptive	
	Humoral	Cell-Mediated
Static	Dynamic	Dynamic
Non-specific	Specific	Specific
Fast	Slow	Slow
No Memory	Memory	Memory
Short Duration	Long Duration	Long Duration
Self/Non-Self	Self/Non-Self	Self/Non-Self
Antigen	Antigen	Antigen
Skin		
Cytokines		
Macrophages		
Natural Killer Cells		Killer T-cells
Dendritic Cells		
PAMPs		
	B-cells	T-cells
	Antibodies	T-cell receptors
	Extra-cellular	Intra-cellular
		Helper T-cells

Table 14 shows some general comparisons between the types of immune systems. Not all experts agree on these divisions, so see immunology texts for more detailed information. The innate and adaptive immune systems interact with each other. A phagocyte is an *eating cell* that consumes pathogens or particles as part of immune activity. A macrophage is a type of phagocyte that digests the invader as part of the innate immune system. However, it sometimes *presents* part of the invader, the antigen, at its cell boundary. The part of the macrophage which presents the antigen is called the Major Histocompatibility Complex (MHC). A cell that presents an antigen, such as a macrophage, is called an Antigen Presenting Cell (APC). A dendritic cell is also a kind of APC.

Negative Selection in Immunity

The immune system uses a process which computer scientists have labeled a Negative Selection Algorithm (NSA)—rather than looking for *intruders*, the immune system looks for *non-self*. Parts of the following description are from [97]. B-cells produce many potential antibodies with each B-cell producing only a single type of potential antibody. The potential antibodies are compared with *self* and the ones that are similar to *self* are eliminated, so only antibodies that do not match with *self* remain. The result is that whatever the remaining antibodies match with is *non-self*. These remaining antibodies are then dispatched to look for matching antigens. When an antigen is found, the B-cells which make the corresponding antibody then make mass quantities of this particular antibody in order to find more of the same antigen. Helper T-cells regulate this B-cell activity. A B-cell whose antibody has matched an antigen is called *mature* or a *plasma cell*. Mutations of the antibody occur to improve the matching with the found antigen. Memory cells remember this antibody in order to continue looking for this antigen in the future. Since there is an almost infinite variety of possible foreign molecules, an enormous amount of antibodies are needed. This is done biologically by using chains of molecules in the antibody which can be shuffled for different combinations of matches. An estimated 24 million combinations are possible which are increased even further by other means. An immediate problem is apparent in imitating this biological activity on a computer: the biological activity is massively parallel with all of the numerous B-cells and antigens being processed at the same time, whereas on a computer, a Central Processing Unit (CPU) can only process the activity of one B-cell or antigen at a

time. If an antibody matches an antigen and is activated, then it is duplicated by clonal selection in order to broaden the search and find more of the same antigen. Any B-cells which make this antibody are signaled to start mutating in order to attempt to find a better match for this particular antigen. These activated B-cells also start tagging the antigens so that macrophages and neutrophils can identify them and destroy them. These activated B-cells also reproduce as a memory cell in case the same antigen appears in the future. Some future antibodies are positively selected (instead of NSA) to match remembered antigens.

The cell-mediated immune system T-cells also detect antigens using T-Cell Receptors on their cell walls using a negative selection method similar to antibodies [97]. However, antibodies can match whole foreign antigens, but T-cell receptors can only match digested fragments of antigens presented by a Major Histocompatibility Complex (MHC). Two kinds of T-Cells react two different ways when matching an antigen presented by an MHC: a helper T-Cell then proceeds to activate B-cells which make the corresponding antibody while a killer T-cell simply kills the host cell presenting the antigen. A killer T-cell is sometimes called a cytotoxic T-cell. Activated T-cells reproduce to save a memory of the antigen. Some future T-cells are positively selected (instead of NSA) to match remembered antigens. [97]

A Negative Selection Algorithm (NSA) is simple without the scientific microbiology terminology:

1. Generate random detectors

2. If a detector matches self, then eliminate it.
3. If a detector matches anything else, trigger a response.
4. Periodically recycle detectors, keeping the ones that have triggered a response.

Danger Theory in Immunity

Besides computation time, this algorithm has obvious problems in biology: bacteria in the intestines is foreign, for example, but is not dangerous; cancerous cells are self, but are dangerous. Likewise in network intrusion detection, an anomaly often turns out to be legitimate user behavior, and malicious behavior can be disguised as normal network traffic. Matzinger [77] addressed this problem in biology in 1994 noting that the immune system is far more concerned with danger and potential destruction than with the distinction between self and non-self. She questioned why some foreign objects elicit immune responses while silicone, well-boiled bone fragments, solitary haptens (parts of antigens), and food do not. As a result she proposed Danger Theory. The concept level with NSA is that the antigen is the indication of the intrusion: if one finds an antigen then one has found an intrusion. Examples of antigens in intrusion detection on this concept level are malicious types of network traffic or fingerprints of malware found in network packets. This concept level changes with Danger Theory in which the antigen is just a label that identifies something. Examples of an antigen in AIS Danger Theory for intrusion detection are an IP address or a network connection. The indicators of intrusions in Danger Theory are a Pathogenic Associated Molecular Pattern (PAMP), danger signals, and

safe signals. An example of a PAMP in AIS Danger Theory for intrusion detection is a network packet that matches a Snort rule. An example of a danger signal in biology is a chemical signal that a cell is dying as a result of an attack by a pathogen. An example of a safe signal in biology is a chemical signal that a cell is dying normally, called *apoptosis*. An example of a safe signal in intrusion detection is network traffic that is believed to be acceptable.

The Dendritic Cell Algorithm

Dendritic cells are the arbiters of Danger Theory, deciding whether or not to initiate a response to an intrusion. When a dendritic cell consumes a pathogen or particle, it travels to a lymph node where it becomes an Antigen Presenting Cell (APC) for T-cells and B-cells, presenting the antigen as being either safe or dangerous. The process of deciding whether an antigen is safe or dangerous is imitated with the Dendritic Cell Algorithm (DCA) [50] which is a method of fusing a series of four types of input into a single contextual output. The concept level has changed to a dendritic cell algorithm being a fusion method for intrusion detection. The four types of input are PAMP signals, danger signals, safe signals, and inflammation, which is just an indicator that may increase the other values. An example of a PAMP signal in AIS Danger Theory for intrusion detection would be the severity of a Snort alert with 0.33 representing low severity, 0.67 representing medium severity, and 1.0 representing high severity. PAMP and danger signals along with danger contexts could be considered to be symptoms of infection depending upon the circumstances. The algorithm maintains three interim values each of which are

aggregated: a co-stimulation signal (CSM), a semi-mature signal, and a mature signal. The co-stimulation signal represents the accumulation of evidence: when a given threshold of evidence is reached, called the *migration threshold*, then a decision is made. In biology, the dendritic cell then migrates to a lymph node and presents the antigen as being either safe or dangerous. The semi-mature signal represents evidence of safety, and the mature signal represents evidence of danger. Here is the general formula for the interim part of this fusion from [50], labeled in this paper as the Interim DCA Formula:

$$Output = \left(P_w \sum_i P_i + D_w \sum_i D_i + S_w \sum_i S_i \right) * (1 + I) \tag{39}$$

The input to the Interim DCA Formula, above, is a series of values for PAMP (P_i), Danger (D_i), and Safe (S_i), signals and an Inflammation (I) signal. Groups of these streaming values are summated with each type of input modified by the corresponding weight, P_w, D_w, or S_w. The knowledge engineer decides how large the groups of data are with the Imprecision Principal. This formula is repeated three times per calculation with varying weights for each group of data providing three separate outputs: one for the CMS signal, one for the semi-mature signal, and one for the mature signal. The two base weights, $W1$ and $W2$ in the table below, are provided by the knowledge engineer with the Imprecision Principal and the remaining weights are either static or else derived from the two base weights.

Table 15, Weights for Interim DCA Formula

Signal	PAMP	Danger	Safe
CSM	$W1$	$\frac{W1}{2}$	$W1 * 1.5$
Semi-Mature	0	0	1
Mature	$W2$	$\frac{W2}{2}$	$-W2 * 1.5$

Table 15 shows how the weights are determined for the Interim DCA Formula. When the Interim DCA Formula is used to calculate the CSM Signal, for example, $P_w = W1$, $D_w = \frac{W1}{2}$, and $S_w = W1 * 1.5$.

Table 16, Weight Examples

Signal\Weight	P_w	D_w	S_w
CSM	6.0	3.0	9.0
Semi-Mature	0.0	0.0	1.0
Mature	*1.0*	0.5	-1.5

Table 16 shows what the weights would be, using Table 15, for each of the three outputs of the Interim DCA Formula if the knowledge engineer set $W1 = 6$ and $W2 = 1$. When calculating the output for the mature signal, for example, $D_w = 0.5$. These sample weights and the Interim DCA Formula can be used to continue the example from the end of the previous section.

Table 17, Interim DCA Formula Output Example

	PAMP Signal	*Danger Signal*	*Safe Signal*		
UDP Plains dis.	0.000	0.084	0.000		
Bot Hills	0.000	0.369	0.000		
Plateau dis.	0.000	0.105	0.000		
Hi Port Mtn. dis.	0.000	0.084	0.000		
Origin Basin	0.000	0.000	0.000		
Traditional Valley	0.000	0.000	0.338		
Port Cliffs dis.	0.000	0.102	0.000		
Snort	0.330	0.000	0.000		
Sum of above	0.330	0.744	0.338		
CSM Signal	1.980	2.232	3.042	=	7.254
Semi-Mature Signal	0.000	0.000	0.338	=	**0.338**
Mature Signal	0.330	0.372	-0.507	=	**0.195**

$$\boldsymbol{Context \equiv 0.338 > 0.195 \equiv Safe}$$

Table 17 considers a Snort alarm along with values from Table 12 to calculate interim DCA signals and provide an example contextual output. The PAMP Signal column includes a value of 0.330 representing a low Snort alert. The sum of PAMP alerts is this one signal of 0.330. This PAMP signal summation of 0.330 is taken times weights of 6, 0, and 1 for values of 1.980, 0, and 0.330 for the CSM, semi-mature, and mature signals respectively. The values labeled *Belief* in Table 12 are labeled *Danger Signal* in Table 17 and their summation is shown as 0.744, which is converted by weights to 2.232, 0, and 0.372 for the CSM, semi-mature, and mature signals respectively. The values labeled *Disbelief* in Table 12 are labeled *Safe Signal* in Table 17 and their summation is shown as 0.338, which is converted by weights to 3.042, 0.338, and -0.507 for the CSM, semi-mature, and mature signals respectively. The interim outputs are aggregated across the rows with the sum for the CSM Signal

being 7.254, the sum for the semi-mature Signal being 0.338, and the sum for the mature Signal being 0.195. The concept level has changed to Snort and SOM outputs becoming *biological* inputs to artificial dendritic cells.

To obtain a contextual output for the example in Table 17, first consider the interim CSM Signal of 7.254. If that is enough stimulation, then the DCA is ready to determine the contextual output. If that is not enough stimulation, then the DCA needs more input. The knowledge engineer determines using the Imprecision Principal what constitutes enough stimulation to determine if enough information has been considered to produce an output. In a system with multiple artificial dendritic cells, the thresholds can be set randomly within a range. Suppose for this example that 7.254 is enough stimulation. Then, the semi-mature signal is compared to the mature signal. If the semi-mature signal is greater than the mature signal, then the contextual output is *safe*; otherwise the contextual output is *danger*. If the context is danger, then the dendritic cell presents the antigen as being danger and immune response (incident response) is initiated; otherwise, it is not initiated. In this example, since the semi-mature signal of 0.338 is greater than the mature signal of 0.195, the contextual output is *safe*, the artificial dendritic cell presents the antigen as being safe, and immune response (incident response) is not initiated. The DCA is intended for multiple artificial dendritic cells, each of which can consider different inputs in different time frames. With multiple dendritic cells the immune response depends upon the greater of semi-mature or mature contexts.

METHODOLOGY SUMMARY

This chapter explained methodologies for components of the system with examples from early research using firewall logs. The next chapter shows how these methodologies were implemented with data collected from the Invisible Mobil Network Bridge (IMNB) using new dynamic methods.

CHAPTER 4 – IMPLEMENTATION AND RESULTS

This chapter demonstrates the implementation and results of the methodology developed in the previous chapter using data collected by the Invisible Mobile Network Bridge (IMNB) instead of from firewall logs. This chapter also explains how the training factor was automatically created initially for each neighborhood size and changed dynamically during the training, as well as how the neighborhood sizes were changed dynamically during the training. A new training monitoring method is demonstrated. The newly trained SOM is analyzed with new color maps and a new 3D map being created, showing danger and safe zones. A method is shown of automatically applying FIS. New data is used to test the Danger Theory aspects of the new hybrid, and the potential of an evolving universal SOM is explained.

The IMNB made network traffic captures in 53 separate sessions including traffic involving desktops, laptops, lab computers, subnets, and computers known to be infected with malware. The network traffic included live connections to the Internet and simulated network traffic using INetSim [98]. Samples included boots, active directory and local logins, shutdowns, browsing, heavy usage, videos, pornography, banking, inactivity, both static and dynamic IP addresses, a variety of operating systems, and long and short duration captures. Malware traffic was related to a computer with multiple infections, a prober, a computer with TDSS/TDL4, and a computer with the Blackhole Exploit Pack. In order to distinguish this new ANNaBell from 1D ANNaBell and 3D ANNaBell, this new version is called LLNIDS ANNaBell.

Original captures for LLNIDS ANNaBell were saved to binary pcap files (creating sets of transmission, T) which were translated with tcpdump to text files (creating sets of events, E, with tcpdump adding metadata, M., producing records, R, and logs, L). A script obfuscated the local IP addresses to remove identifying information. These text files were processed by another script to create aggregate vectors (V_A) in 10-second increments. These increments were further aggregated into 60-second sliding windows, creating a potential data input matrix (D). Most of these vectors from D were saved in a file for training (D_{Train}) and the remaining were saved in a file for testing (D_{Test}).

Features for LLNIDS ANNaBell were selected primarily based on the previous successes of 1D ANNaBell and 3D ANNaBell. The words *subject* and *alien* are used in describing IP addresses in the features, indicating which side of the Invisible Mobile Network Bridge (IMNB) an IP address is active on. The *alien* side of the IMNB is the side connected with the Internet or the simulated network, such as INetSim. The *Subject* side is the other side of the IMNB which is connected to the one or more computers or subnet which is being analyzed. In network traffic coming from the subject computer and/or the subject subnet, the source IP addresses are the subject IP addresses. All other IP addresses are alien. While 1D ANNaBell and 3D ANNaBell were based on a Vulture Fest model of attempted, but denied, alien network traffic, LLNIDS ANNaBell is based on two-way actual (not denied) network traffic. For this reason, one-way alien traffic is ignored in LLNIDS ANNaBell. The features for LLNIDS ANNaBell are as follows:

- ip_count The number of subject IP addresses involved in the network traffic. Naively this would be one IP address, but some computers used multiple IP addresses and some of the traffic included numerous local IP addresses on a subnet. For traffic involving multiple subject IP addresses, such as for a subnet, vectors were created for each subject IP address and also for the subnet as a whole.
- tot_count The total count of records, R.
- uniq_aliens The number of unique alien IP addresses which are being communicated with by subject IP addresses. This would typically be non-local IP addresses out on the Internet, IP addresses on another subnet, or IP addresses spoofed by a fake network such as INetSim.
- uniq_ports The number of unique destination ports in the traffic.
- port_ave The average of the port numbers of the destination ports in the traffic.
- tcp_rat The ratio of TCP protocol in the traffic.
- bytes The total number of bytes involved in the traffic.

ip_count was included as a new feature because it helps to differentiate between subnets and individual computers and also because single computers might be using more than one IP address. tot_count was used successfully in 1D ANNaBell and 3D ANNaBell. uniq_aliens and uniq_ports are used similarly as in 1D ANNaBell and 3D ANNaBell, except here they are counts and not ratios.

port_ave is an indicator of how much high-numbered destination ports are used in the traffic. lo_port (for lowest destination port) was used in 1D ANNaBell but was not indicative of malware, so this feature was eliminated. hi_port (for highest destination port) was successfully used in 3D ANNaBell, but port_ave should be an even better indicator. tcp_rat replaces upd_rat (for UDP ratio) in 1D ANNaBell and 3D ANNaBell because more ICMP protocol network traffic appears in IMNB logs. Since UDP and ICMP traffic can be associated more with malicious network traffic, tcp_rat is used as a potential indicator of safe network traffic. bytes was added as a feature because the number of bytes transmitted in the network is readily available from IMNB logs and this should be a notable characteristic of types of network traffic. (The number of bytes transmitted was not available for 1D ANNaBell and 3D ANNaBell.)

A SOM size of 91 nodes was selected which provided for an average of approximately 31 inputs per node, slightly more detailed than previous versions of ANNaBell. The vector elements were each normalized to a range of [0, 1] during training so that each vector dimension would have a similar magnitude of effect on the training. This normalization can only be approximated because future high values of some of the count elements, such as the tot_count, cannot be anticipated. The training factor is called *eta* in some papers and *alpha* in others. Eta and alpha are interchangeable in this paper and both mean the training factor. The training software was written in Perl [99].

TRAINING THE LLNIDS SOM

Several enhancements were made to reduce the SOM training time

including how the nodes were set up, automatically determining the first training factor for each neighborhood distance, dynamically changing the training factor for each iteration, monitoring the progress of the training, and dynamically determining the number of iterations for each neighborhood distance.

The initial D_N was created less randomly as for 1D ANNaBell and 3D ANNaBell in order to reduce training time. In 1D ANNaBell and 3D ANNaBell, the only special vectors were the origins, (0, 0, 0, 0, 0, 0, 0, 0) and (0, 0, 0, 0, 0, 0). The remaining vectors were created randomly. In LLNIDS ANNaBell, all of the initial D_N vectors were D_I vectors. Ten of these vectors were special and the others were chosen randomly from D_I. These were the ten special vectors chosen for the initial D_N:

1. The origin (0, 0, 0, 0, 0, 0, 0).
2. The vector from D_I with the maximum ip_count value.
3. The vector from D_I with the maximum tot_count value.
4. The vector from D_I with the maximum uniq_aliens value.
5. The vector from D_I with the maximum uniq_ports value.
6. The vector from D_I with the maximum port_ave value.
7. A vector from D_I with the maximum tcp_rat value.
8. The vector from D_I with the maximum bytes value.
9. The vector from D_I with the most average values.
10. The vector from D_I which was the most distant from the origin.

Noting that the origin and most of the nodes (all except one) with the maximum values ended up on the edge of the meta-hexagon in 3D ANNaBell,

the origin and all of the vectors with the maximum values were assigned to edge nodes of the meta-hexagon in the initial D_N of LLNIDS ANNaBell. The vector with the most average values was assigned to Node 0 in the center of the meta-hexagon.

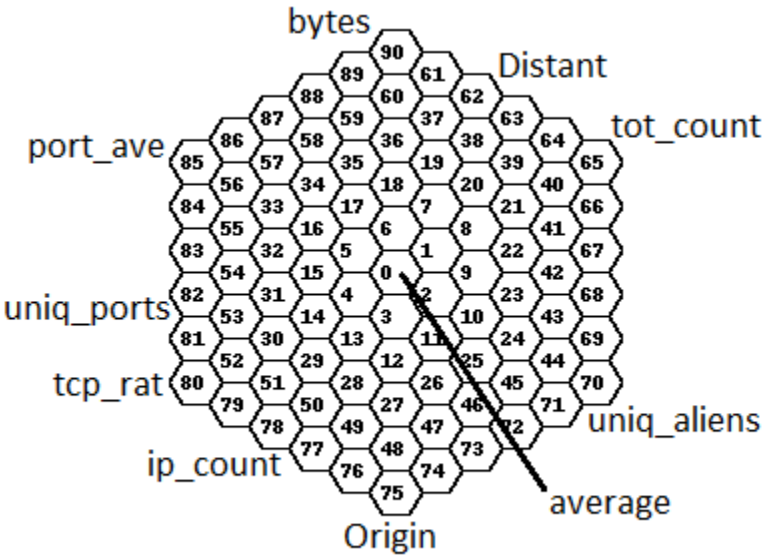


Figure 32, Initial SOM Layout

Figure 32 shows the initial node assignments with Node 0 in the center being assigned the most average vector. The other assignments in clockwise order from the upper right are Node 62, most distant; Node 65, maximum tot_count; Node 70 maximum uniq_aliens; Node 75, origin; Node 78, maximum ip_count; Node 80, maximum tcp_rat; Node 82, maximum uniq_ports; Node 85, maximum port_ave; and, Node 90 maximum bytes. All of the remaining node vectors were taken randomly from D_I. Figure 32 also shows the node numbering from Node 0 in the center spiraling out in a clockwise direction to Node 90 at the top, which is a numbering scheme similar to that used in 3D ANNaBell.

Automating the Initial Training Factor

$$\frac{1}{|V_{\mathrm{N_{MAX}NBH}_{s,t}}|} \quad (40)$$

Formula 40 is the training factor, explained earlier, for the first iteration of each new neighborhood size. It is the inverse of the maximum vector node neighborhood BMI count for a given neighborhood size, s, and iteration, t. This training factor was manually estimated in 3D ANNaBell, which stopped training between each neighborhood size. However, it was automatically calculated in LLNIDS ANNaBell, which continued training with the new training factor without stopping between neighborhood sizes.

Dynamically Changing the Training Factor

The training factor continued to be automatically adjusted between each iteration in order to reduce training time. This was done by keeping track of the average movement of the nodes during training iterations. If the average movement for the current iteration was less than the average movement for the previous iteration, then the training factor was increased by 3 percent, otherwise the training factor was reduced by 50 percent. The 3 percent increase was determined by experimentation. The 50 percent reduction was necessary to prevent thrashing back and forth between alternate sides of a target.

Figure 33, Training Factor Adjustments

Figure 33 illustrates the rational of the per-iteration training factor adjustment. The exact location of the target (star) is not necessarily known during the approach. Moving from left to right the training factor is increased by 3 percent per iteration as the target is approached in order to reduce the time in reaching the target. When the target is overshot, the training factor is reduced by 50 percent so that the target is continued to be approached without just thrashing back and forth between alternate sides of the target. In SOM training, a target is the theoretical location where a node best represents a cluster of input vectors. The SOM training factor is adjusted based on an average of all of the node vector changes. This average is determined by monitoring the SOM training.

Monitoring the Progress

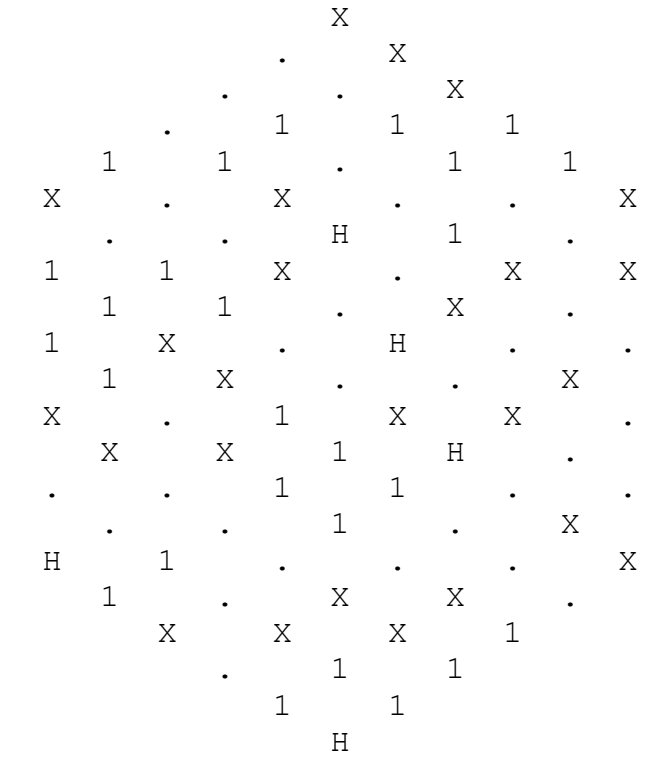
Below is the standard output report for a single iteration of LLNIDS ANNaBell:

```
Iteration 0 distance 10.
alpha 0.000353606789250354.
This iteration lasted 0.130956367651621 minutes.
Average change = 0.339842126153345.
BMN changes:  2828.  Max BMI:  2828

                  1
               .     X
            .     X     1
         X     X     X     1
      1     X     1     X     1
   X     .     H     X     X     1
      .     1     H     X     X
   X     1     X     X     X     X     Key, per node:
      1     X     1     X     X           .  No BMI
   1     1     .     X     X     X        1  1-9 BMI
      X     X     1     .     1           X  10-99 BMI
   1     .     X     X     X     .        H  100+ BMI
      H     X     X     H     X
   1     1     X     1     1     X
      .     X     X     .     X
   X     1     .     .     .     X
      .     .     H     .     .
         1     X     X     1
            1     X     X
               X     1
                  H
```

The above output provides the following information. This is the report for Iteration 0 with a neighborhood distance of 10. The training factor was 0.000353606789250354 and the processing time for this iteration was 0.130956367651621 minutes. There were 2,828 changes for a BMN and the maximum count of BMI for the largest neighborhood was 2,828. A text-based meta-hexagon then provides a summary of the distribution of the BMI for each node. A period (.) represents no BMI for a node; a 1 represents from 1-9 BMI for a node; an X represents from 10-99 BMI for a node; and, an H represents 100 or more BMI for a node. The next iteration was as follows:

```
Iteration 1 distance 10. alpha 0.000364214992927864.
This iteration lasted 0.115660949548086 minutes.
Average change = 0.127156481290656.
BMN changes:  1677.  Max BMI:  2828
```

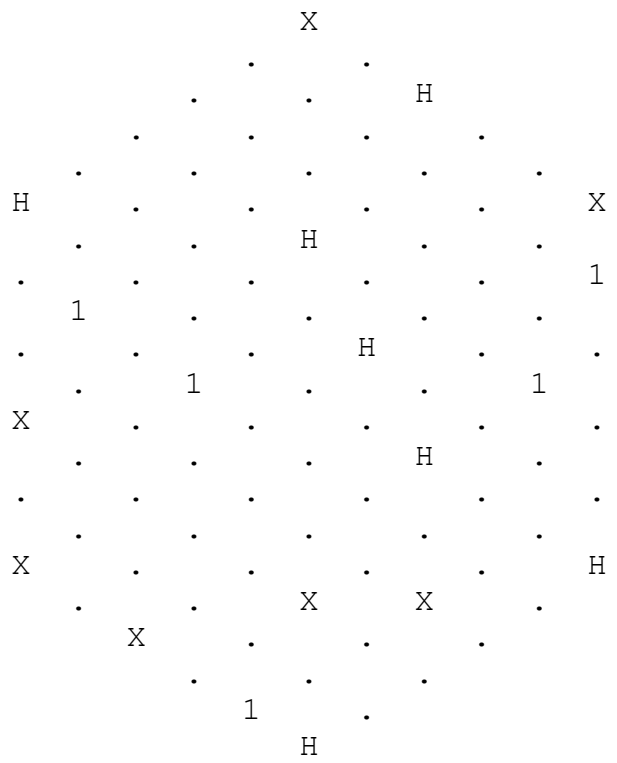


The above output indicates that this information is for Iteration 1. The neighborhood distance is still 10. The training factor (alpha) has increased by 3 percent. The average change is less than the previous iteration. The number of BMN changes has gone down to 1,677. The knowledge engineer can monitor this output in real time to verify that the SOM is making progress in training; otherwise, variables may need to be adjusted and the training restarted. The processing time should be monitored because training can last for days. Experience indicates that the training factor, the average change, and the number of BMN changes should all approach zero over numerous iterations.

Dynamically Changing the Neighborhood Size

One of the goals of this research was to determine a dynamic way of determining when to change the neighborhood size. Experimentation showed that BMN changes converged within a reasonable amount of time to zero for each neighborhood distance, so this became the determining factor for going to the next neighborhood size. However, a maximum limit on the number of iterations for each neighborhood size was placed in the code to prevent any potential infinite loops. Experimentation showed that after the BMN changes reached zero, they did not necessarily stabilize at zero---they could go back up in a future iteration. Even so, the first time BMN changes reached zero became the deciding factor to change neighborhood sizes because this represented some convergence, and the nodes would move again at the next neighborhood size unless it was zero. A *perfect* solution was not attempted because a goal of this SOM was to be perpetually dynamic with no final ending state. This will be explained more when testing of the SOM is covered further below.

```
Iteration 56 distance 10
alpha 2.42215458070362e-05
This iteration lasted 0.112553381919861 minutes.
Average change = 0.000119039384720208.
BMN changes:  0.  Max BMI:  2828
```



The above output shows the status after Iteration 56, which was the last iteration at a neighborhood distance of 10. Note that the distribution of the BMI has significantly changed so that most nodes have zero BMI. The BMI will be redistributed to the other nodes as the neighborhood size decreases in the training. Experience has shown that, in general, BMI are distributed to the outer nodes at first, and then redistributed to the inner nodes.

```
Iteration 309 distance 5
alpha 1.36307815988189e-05.
This iteration lasted 0.0593816002209981 minutes.
Average change = 6.91557543954126e-05.
BMN changes:  0.  Max BMI:  2828

                  H
               X     .
            1     .     .
         1     1     .     .
      X     .     .     .     .
   H     X     1     .     .     1
      X     1     .     .     .
   X     X     1     .     .     H
      1     X     .     .     X
   1     .     X     X     X     1
      .     X     1     X     X
   1     1     X     1     .     X
      1     1     1     .     1
   1     1     1     1     .     1
      1     .     1     .     .
   H     .     .     1     .     .
      H     .     .     1     1
         H     .     .     1
            X     .     .
               X     H
                  X
```

The above output shows the status at the last iteration for a neighborhood distance of 5. Note that the training factor and the average change are both minute. The above output was to the screen for real-time monitoring. A log file was also created

```
Iteration 476 distance 0
alpha 0.000826913829579694.
This iteration lasted 0.0147421836853027 minutes.
Average change = 0.000338512549388827.
BMN changes:  0.  Max BMI:  326

                  X
               X     X
            X     X     X
         X     X     X     X
      X     X     X     X     .
   X     X     X     X     X     X
      X     X     X     .     1
   1     X     1     X     .     X
      X     X     .     X     X
   X     X     X     .     1     X
      1     X     1     X     1
   X     X     X     X     X     X
      X     X     X     X     X
   X     1     X     X     X     X
      X     X     1     X     X
   X     X     X     1     1     X
      X     X     X     .     .
         X     X     X     H
            X     X     X
               1     X
                  H
Elapsed time:  31.7587482492129 minutes.
```

The above output shows the status when the SOM training stopped. Note the total elapsed time of approximately 32 minutes which is substantially less than the 6-day training time of 1D ANNaBell and the 2-day training time of 3D ANNaBell. The speed is attributed to a more focused data set, enhanced automatic training monitoring, optimized code, and better hardware. The training time could probably be further reduced by using a compiled programming language such as C instead of Perl. A log file was also created during the training with a somewhat different focus on the progress of the training.

```
     359 distance 3, Eta 0.000397011669280822:
0.00697095083557402 average change, (10 biggest at
0.03585063519038).
     360 distance 3, Eta 0.000408922019359247:
0.00572748632108736 average change, (10 biggest at
0.0296745465177702).
     361 distance 3, Eta 0.000204461009679623:
0.00684703722906379 average change, (44 biggest at
0.0623563975424632).
```

Above is a three-line example from the log file output. The first line shows that for iteration 359, the neighborhood distance was 3 and the training factor was 0.000397011669280822. The average change was 0.00697095083557402 and Node 10 had the biggest change at 0.03585063519038. One can see that the average change increased from Iteration 360 to Iteration 361 so the training factor was reduced by 50 percent, at which time Node 44 had the biggest change instead of Node 10. This information can be used by the knowledge engineer to monitor the status of the training.

EXAMINING THE SOM

Once the SOM has been trained, it needs to be examined in order to see the results of the training. What changes during the SOM training is the location of the nodes, V_N, in multidimensional space moving towards the locations of the input vectors, V_I, in the same multidimensional space. The closest node to an input vector is the BMN. Similarly, the closest input vectors to a node are the BMI. At the end of the training, various nodes are in proximity to various clusters of input vectors, while other nodes may be in between clusters of input vectors.

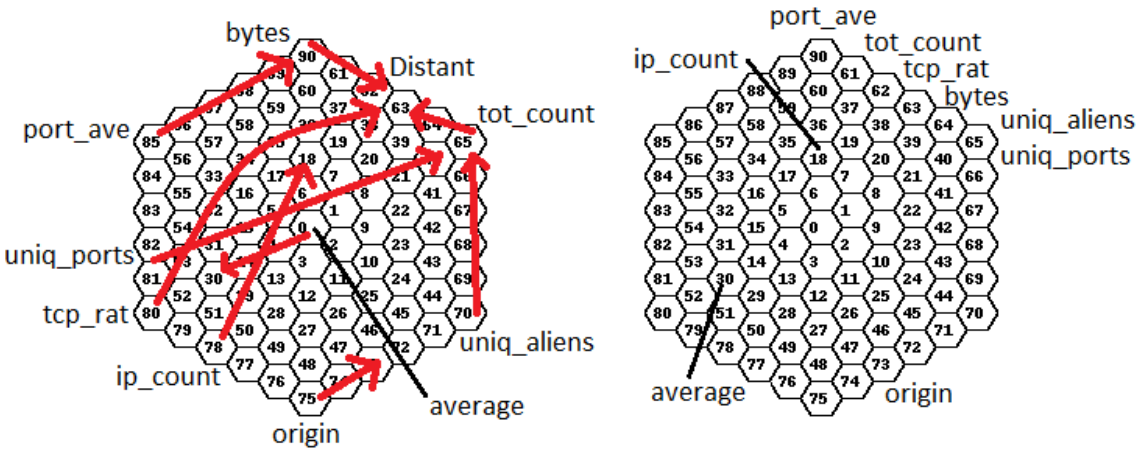


Figure 34, Node Training Movement

Figure 34 shows how the nodes moved to accommodate the 10 special vectors during training. The left meta-hexagon uses arrows to indicate movement from the starting positions. The right meta-hexagon shows the labels for the ending positions after the training. The most extreme vectors were in the upper right of the meta-hexagon after the training. Although it appears that the maximum value for tcp_rat, for example, moved from Node 80 to Node 63, it was actually nodes 80 and 63 (and all of the other nodes) which did the moving in multidimensional space. The data in the trained SOM can be analyzed many different ways.

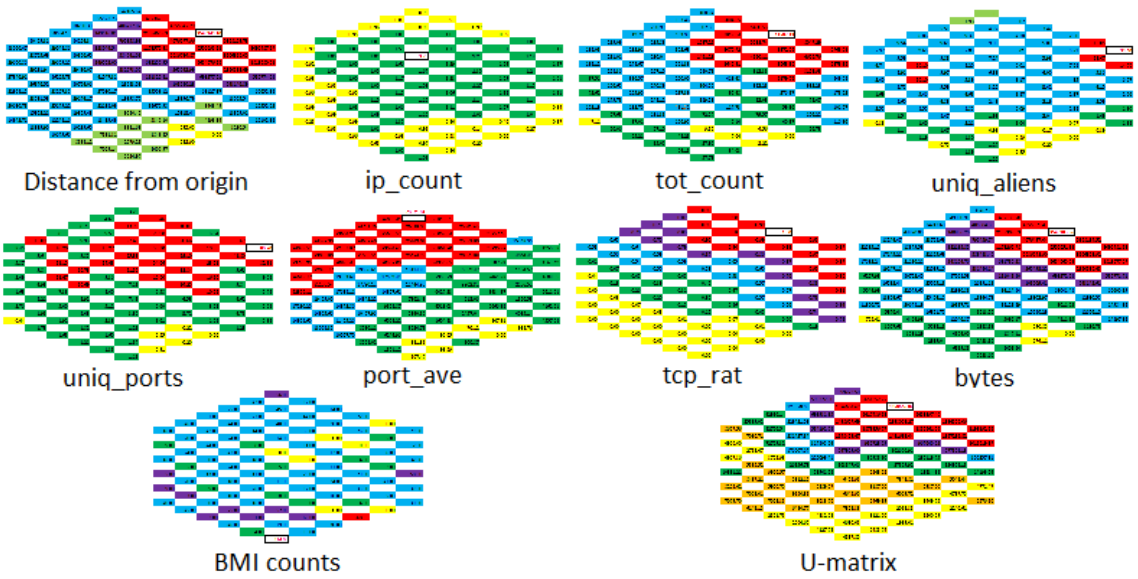


Figure 35, Preliminary SOM Analysis

Figure 35 shows a variety of preliminary analyses on the SOM results which were created using a spreadsheet. Yellow represents the lowest values, with green, blue, purple, and red representing increasingly higher values. The node with the highest value in each case is boxed in black. The *Distance from origin* meta-hexagon shows the origin in yellow on the lower right with vector spaces then located off to the left and then up to the right, with the furthest node from the origin being in the upper right. The ip_count graphic generally shows lower values around the edges moving in to the highest value in the upper center. The tot_count, uniq_ports, tcp_rat, and bytes graphics generally show high values in the upper right. The uniq_aliens graphic shows high points in the upper right, but also in the middle left. The port_ave graphic shows higher values in the upper left. *BMI counts* shows where most of the input vectors are located in the map: generally on the lower left side. *U-matrix* indicates average distances between neighboring nodes: the nodes on the bottom and the left are

closer together while the nodes in the upper right are furthest apart. For a full-color graphic, uniq_aliens, port_ave, and tcp_rat appear to have varying and distinctive patterns which blended together should produce a map showing higher level patterns.

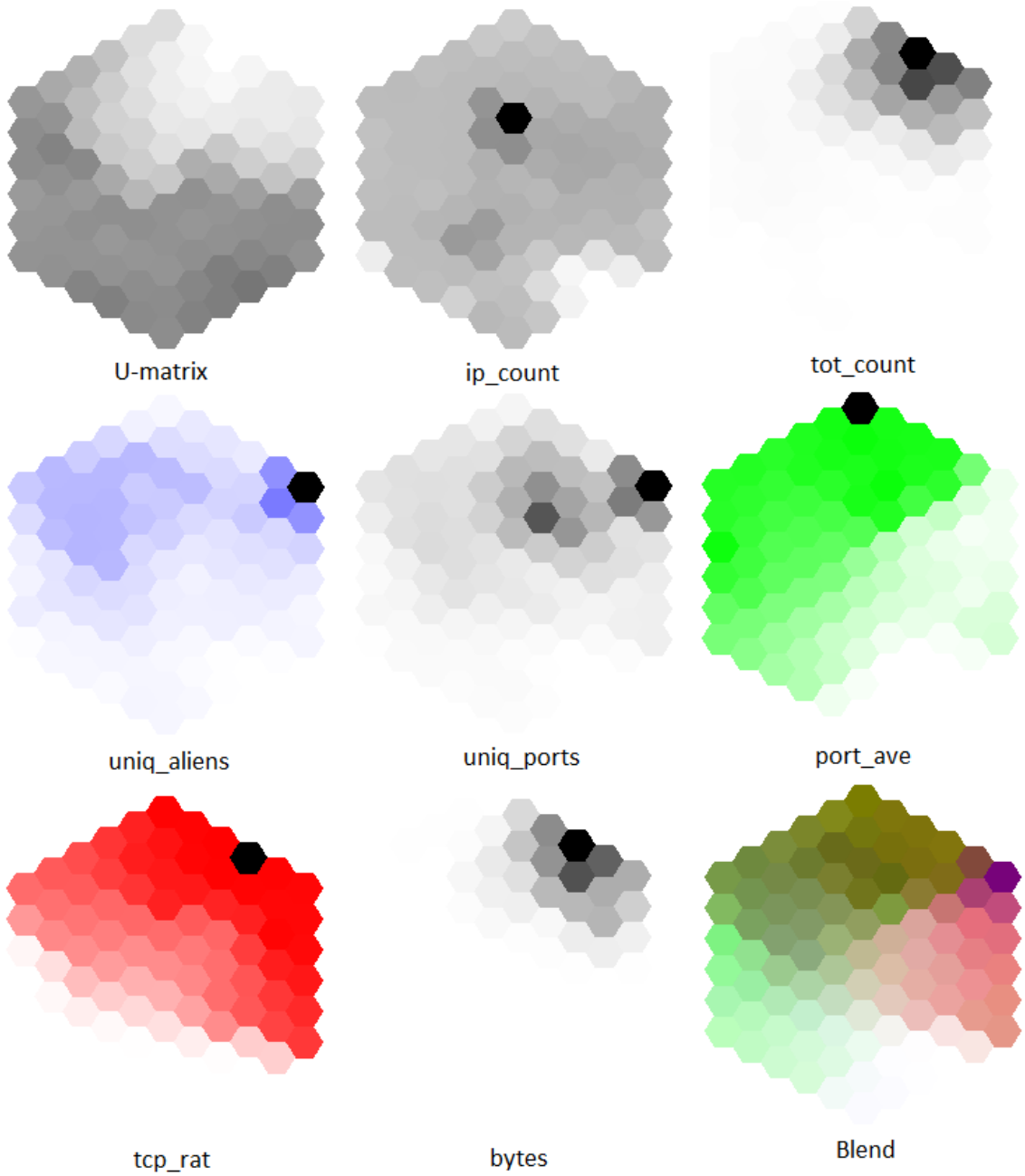


Figure 36, SOM Maps

Figure 36 shows a variety of SOM maps from this training which were created with custom Python [100] script. These maps take longer to create, but provide more detail than the spreadsheet graphics shown previously. Logarithmic values were used in the U-matrix meta-hexagon because otherwise only one hexagon could be seen. The U-matrix graphic generally indicates that nodes are close together around the bottom and left edges and are extremely far apart in the upper right. The next seven meta-hexagons show the intensities of the values for each of the seven dimensions in the vectors. Solid black hexagons represent the locations of the maximum values. The primary colors of red, green, and blue are used for tcp_rat, port_ave, and uniq_aliens because this helps to visualize how the values of these dimensions are blended to create the *Blend* meta-hexagon, which shows the most information of any of the graphics in this group.

Figure 37, Alternate Blends

Figure 37 shows three example alternate ways of blending the data for full-color display. Alternate Blend A used uniq_aliens as red, port_ave as green, and tcp_rat as blue. Alternate Blend B used port_ave as red, tcp_rat as green,

and uniq_aliens as blue. Alternate Blend C used tot_count as red, port_ave as green, and uniq_ports as blue. The objective with trying these different blends is to find the blend that best illustrates the distribution of the data by the SOM.

DANGER AND SAFE ZONES

Self-training in the context of this research means that the SOM was unaware of what input vectors originated from known infected computers and what input vectors originated from computers believed to be uninfected. The SOM trained itself with raw unlabeled data not knowing what data represented *good* network traffic and what data represented *bad* network traffic. In order to be useful for information security, it is necessary to determine if the SOM successfully distinguished between pertinent types of network traffic. Unlike 1D ANNaBell and 3D ANNaBell where each IP address was represented by a single node, LLNIDS ANNaBell was trained with moving windows of temporal network traffic, so numerous nodes can represent a single IP address, making visualization and analysis more complex.

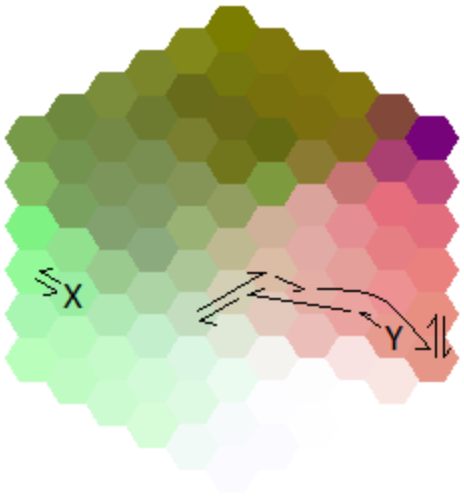


Figure 38, Blackhole Exploit Pack Example

Figure 38 is a an example comparing an office desktop known to be infected with the Blackhole Exploit Pack with another office desktop believed to be uninfected. The samples were taken while each computer was connected to INetSim, and each sample is approximately two minutes long. The BMN for the *uninfected* computer started with Node 53, marked with an *X*, moved to Node 82 as marked with an arrow, and then alternated between nodes 53 and 82. (Refer to a previous graphic to review the node numbering system, if desired.) The infected computer began with Node 44, marked with a *Y*, and then moved in sequence to nodes 24, 3, 13, 2, 10, 70, 69, and 70.

Other infected computers matched other nodes and other *uninfected* computers also matched other nodes, with some infected and *uninfected* computers sometimes matching the same nodes. An enormous number of variables are involved such as the type of operating system, whether the computer is on a domain, if a computer is doing automatic updating, how the computer has been used in the past, how the computer is currently being used,

what other software is installed on the computer, what else is happening on a network, how a computer is configured, and the available bandwidth. An infected computer should be doing much of the same network activity as an uninfected computer, but will also be communicating with its controller and should also be participating in additional activity such as scanning the network, attacking other computers, sending spam, uploading information, downloading additional malware, or other activities as instructed by its controller.

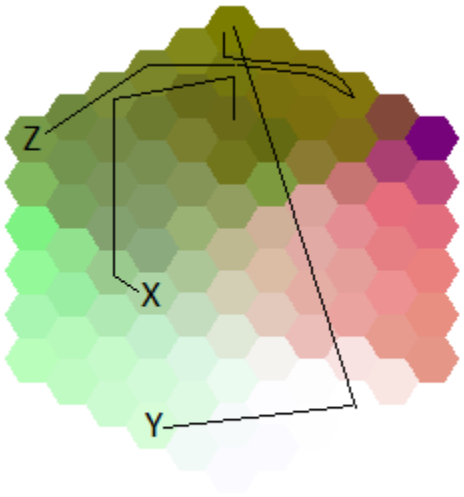


Figure 39, Normal Usage Examples

Figure 39 shows examples of normal usage of computers connected to the Internet. *X* is a student laptop booting, browsing, and shutting down. *Y* is an office desktop idling. *Z* is an office desktop being heavily used. In this graphic, continuous lines are shown instead of arrows; the timelines start at the *X*, *Y*, and *Z* labels.

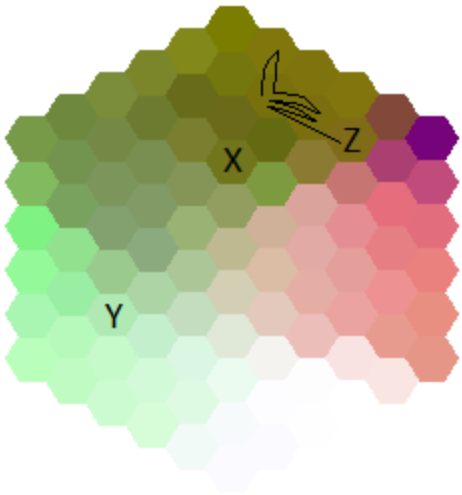


Figure 40, Special Examples

In Figure 40, *X* is the subnet of a computer lab connected to the Internet; *Y* is a computer running a YouTube video; and *Z* is surfing of a pornography web site (which was considered to be *uninfected* traffic because the computer used was not believed to be infected). *X* and *Y* in these cases are examples of when a BMN did not change over time.

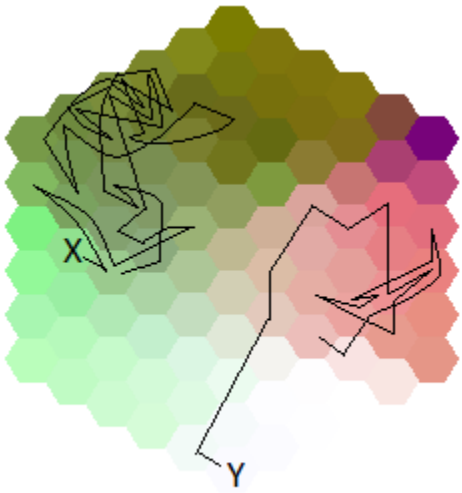


Figure 41, Malicious Examples

In Figure 41, X was a laptop which set off an alarm for probing the network

with network discovery software, a borderline situation of malicious behavior. Y was a computer infected with TDSS/TDL4. Since an infected computer will also have *normal* network traffic, the nodes of the SOM representing only infected or only uninfected traffic are not readily apparent by looking at the figures displayed, so far. In order to more accurately determine the nature of the traffic depicted by the various nodes, a histogram was created of the BMN counts for both infected and *uninfected* computer traffic. In some cases, a node represented only infected traffic or *uninfected* traffic. In the remaining cases a ratio was created of the amount of infected traffic that a node represented.

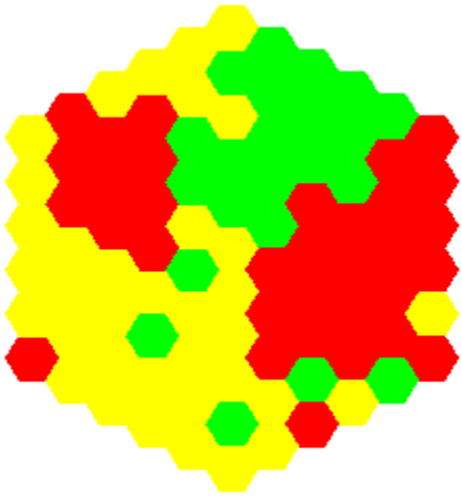


Figure 42, Danger/Safe Zones

Figure 42 displays the danger and safe zone of the map according to the type of traffic that the nodes represent. The red nodes (for *stop*) have BMI of traffic of only infected computers, the green nodes (for *go*) have BMI of only traffic of uninfected computers, and the yellow nodes (for *caution*) have BMI of both types of traffic. These zones can be used to show symptoms of infections.

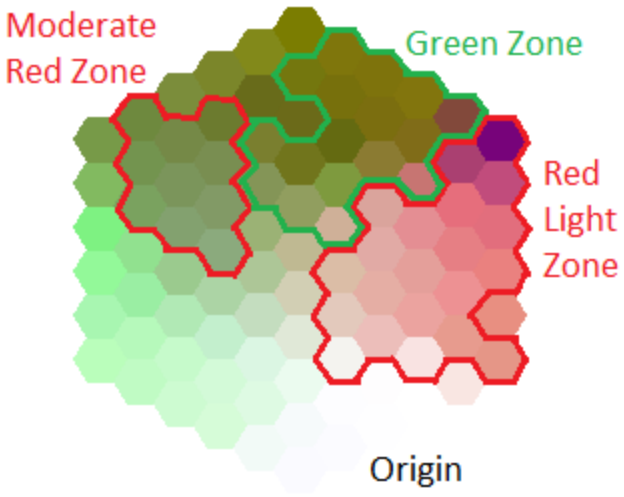


Figure 43, Major Red/Green Zones

Figure 43 shows the three major zones superimposed on the blended map. The Green Zone is the largest contiguous area where the traffic of infected computers did not have a BMN. The Red Light Zone is the largest danger zone which also has the most extreme vector values. The Moderate Red Zone has less extreme vector values. The blended color scheme and zones indicate that the node vectors in these areas can be interpreted both for fuzzy inference and also for danger/safe signals.

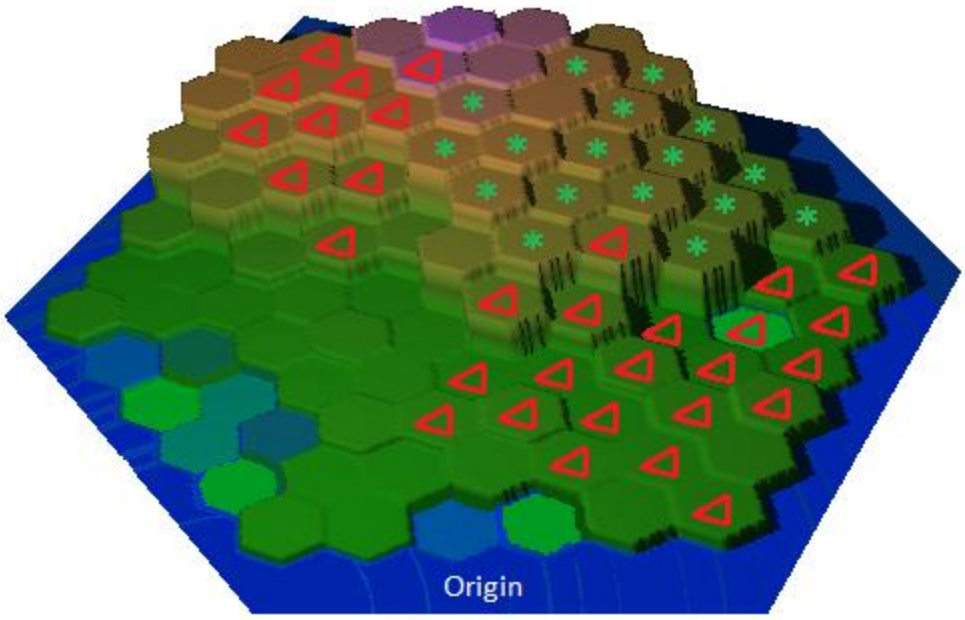


Figure 44, 3D Full-Color SOM with Major Zones

Figure 44 shows a 3D full color representation of the SOM, created with Art of Illusion [92], with the nodes in the red zones indicated by red triangles and nodes in the Green Zone indicated by green asterisks. The heights are logarithms of the U-matrix values, meaning that the vectors of the nodes represented by lower hexagons are close together and the vectors of the nodes represented by higher hexagons are extremely far apart from each other in multidimensional space. The colors of the hexagons are based on their heights, generally with shades of green indicating lower elevations and shades of brown indicating higher elevations. The area around the meta-hexagon is blue.

EXAMINING THE LLNIDS FIS

The Fuzzy Inference System (FIS) was based on these ranges for the dimensions: ip_count, 0 to 3.756307952; tot_count, 0 to 11592.18606; uniq_aliens, 0 to 34.40384547; uniq_ports, o to 80.90343139; port_ave, 0 to 28071.80605; tcp_rat, 0 to 0.996178223; and, bytes, 0 to 9596203.858 (values extremely close to 0 were rounded to 0). A goal of this research was to automate the creation of the fuzzy categories instead of creating them manually as was done with 3D ANNaBell. Experience has shown that extreme values are important in categorizing network activity, therefore, these equally spaced categories were chosen including very low and very high values: very low (VL), low, medium (med), high, and very high (VH).

Table 18, LLNIDS ANNaBell Fuzzy Values

	ip_count	tot_count	uniq_aliens	uniq_ports	port_ave	tcp_rat	bytes
Green Zone	Low to VH	Mixed	VL to Low	VL to High	Low to VH	Med to VH	Mixed
Moderate Red Zone	Low	VL	Low	VL	High to VH	Low to High	VL
Red Light Zone	Low	VL to Med	Mixed	Mixed	VL	Mixed	VL to Low

Table 18 shows the FIS results. The Green Zone, for example, had ip_count values from low to very high; the tot_count values were mixed; the uniq_aliens counts were from very low to low; the uniq_ports counts were from very low to high; the port_ave values were from low to very high; the tcp_rat values were from medium to very high; and, the byte counts were mixed.

APPLYING LLNIDS AIS DANGER THEORY

Separate IMNB data put aside for D_{TEST} was used to test the new system. D_{TEST} included sample network traffic from the computer used in a previous

example which set off an alarm for probing the network. The network discovery software which was found on this computer was disabled and additional input data was created from the resulting network traffic in order to compare it with the previous traffic from the same computer while it was known to be probing.

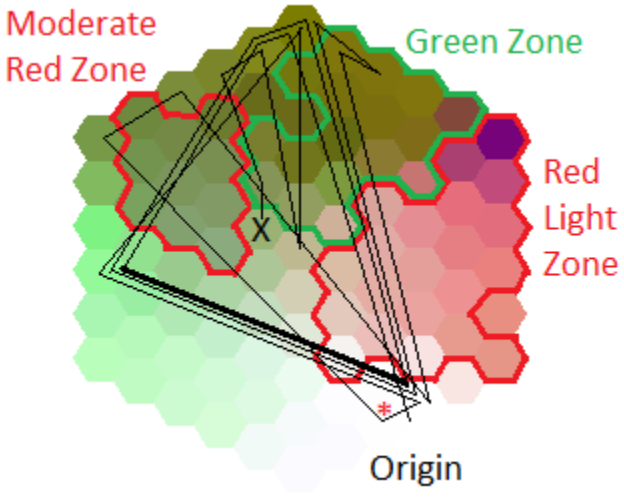


Figure 45, Example Test Map

Figure 45 displays the network traffic node pattern of the probing computer after the network discovery software was disabled. The previous pattern was in the vicinity of the Moderate Red Zone and this has changed. The new pattern includes nodes mostly outside of the main zones, but also includes a single *red* node (marked with a red asterisk (*)) and two nodes in the Green Zone. This sample represents approximately 22 minutes of network activity.

The Fuzzy Inference System (FIS) shown previously for 3D ANNaBell along with SJ Fusion could be used provide an indication of the danger, if any, of this test data, but this would be considerably more complex with multiple nodes involved (with 3D ANNaBell, only a single node was involved in the FIS calculation). Instead, using the ratio of infected traffic that each node represents

in order to create danger and safe signals as input to the Dendritic Cell Algorithm (DCA) from Artificial Immune System Danger Theory (AISDT) is a much more direct method of getting an indication of any possible danger.

Table 19, Danger/Safe Signals over Time

Time Series	BMN	Danger Signal	Safe Signal		CSM	Danger	Safe	Alert
1	5	0.00	0.94		1.41	-1.41	0.94	
2	17	0.00	1.00		2.91	-2.91	1.94	
3	88	0.00	0.77		4.07	-4.07	2.71	
4	89	0.00	0.73		5.16	-5.16	3.44	
5	89	0.00	0.73		6.26	-6.26	4.17	
6	89	0.00	0.73		7.35	-7.35	4.90	
7	0	0.80	0.00		7.75	-6.95	4.90	
8	90	0.00	0.86		9.04	-8.24	5.76	
9	90	0.00	0.86		10.33	-9.53	6.62	Safe
10	82	0.65	0.00		0.33	0.33	0.00	
11	82	0.65	0.00		0.65	0.65	0.00	
12	82	0.65	0.00		0.98	0.98	0.00	
13	82	0.65	0.00		1.30	1.30	0.00	
14	82	0.65	0.00		1.63	1.63	0.00	
15	82	0.65	0.00		1.95	1.95	0.00	
16	72	0.89	0.00		2.40	2.40	0.00	
17	73	1.00	0.00		2.90	2.90	0.00	
18	73	1.00	0.00		3.40	3.40	0.00	
19	73	1.00	0.00		3.90	3.90	0.00	
20	73	1.00	0.00		4.40	4.40	0.00	
21	85	0.69	0.00		4.74	4.74	0.00	
22	85	0.69	0.00		5.09	5.09	0.00	
23	87	0.61	0.00		5.39	5.39	0.00	
24	87	0.61	0.00		5.70	5.70	0.00	
25	87	0.61	0.00		6.00	6.00	0.00	
26	87	0.61	0.00		6.31	6.31	0.00	
27	72	0.89	0.00		6.75	6.75	0.00	
28	72	0.89	0.00		7.20	7.20	0.00	
29	72	0.89	0.00		7.64	7.64	0.00	
30	72	0.89	0.00		8.09	8.09	0.00	
31	72	0.89	0.00		8.53	8.53	0.00	
32	72	0.89	0.00		8.98	8.98	0.00	

Table 19 (Continued)

33	72	0.89	0.00		9.42	9.42	0.00	
34	61	0.00	1.00		10.92	7.92	1.00	**Danger**
35	62	0.00	1.00		1.50	-1.50	1.00	
36	62	0.00	1.00		3.00	-3.00	2.00	
37	62	0.00	1.00		4.50	-4.50	3.00	
38	62	0.00	1.00		6.00	-6.00	4.00	
39	62	0.00	1.00		7.50	-7.50	5.00	
40	61	0.00	1.00		9.00	-9.00	6.00	
41	90	0.00	0.86		10.29	-10.29	6.86	**Safe**
42	72	0.89	0.00		0.45	0.45	0.00	
43	72	0.89	0.00		0.89	0.89	0.00	
44	72	0.89	0.00		1.34	1.34	0.00	
45	72	0.89	0.00		1.78	1.78	0.00	
46	72	0.89	0.00		2.23	2.23	0.00	
47	72	0.89	0.00		2.67	2.67	0.00	
48	72	0.89	0.00		3.12	3.12	0.00	
49	72	0.89	0.00		3.56	3.56	0.00	
50	72	0.89	0.00		4.01	4.01	0.00	
51	72	0.89	0.00		4.45	4.45	0.00	
52	72	0.89	0.00		4.90	4.90	0.00	
53	72	0.89	0.00		5.34	5.34	0.00	
54	72	0.89	0.00		5.79	5.79	0.00	
55	72	0.89	0.00		6.23	6.23	0.00	
56	72	0.89	0.00		6.68	6.68	0.00	
57	72	0.89	0.00		7.12	7.12	0.00	
58	72	0.89	0.00		7.57	7.57	0.00	
59	72	0.89	0.00		8.01	8.01	0.00	
60	82	0.65	0.00		8.34	8.34	0.00	
61	82	0.65	0.00		8.66	8.66	0.00	
62	82	0.65	0.00		8.99	8.99	0.00	
63	89	0.00	1.00		10.49	7.49	1.00	**Danger**
64	89	0.00	0.73		1.10	-1.10	0.73	
65	89	0.00	0.73		2.19	-2.19	1.46	
66	89	0.00	0.73		3.29	-3.29	2.19	
67	89	0.00	0.73		4.38	-4.38	2.92	
68	89	0.00	0.73		5.48	-5.48	3.65	
69	90	0.00	0.86		6.77	-6.77	4.51	
70	90	0.00	0.86		8.06	-8.06	5.37	

Table 19 (Continued)

71	90	0.00	0.86		9.35	-9.35	6.23	
72	72	0.89	0.00		9.79	-8.90	6.23	**Safe**
73	72	0.89	0.00		0.45	0.45	0.00	
74	72	0.89	0.00		0.89	0.89	0.00	
75	72	0.89	0.00		1.34	1.34	0.00	
76	72	0.89	0.00		1.78	1.78	0.00	
77	72	0.89	0.00		2.23	2.23	0.00	
78	72	0.89	0.00		2.67	2.67	0.00	
79	72	0.89	0.00		3.12	3.12	0.00	
80	72	0.89	0.00		3.56	3.56	0.00	
81	72	0.89	0.00		4.01	4.01	0.00	
82	72	0.89	0.00		4.45	4.45	0.00	
83	72	0.89	0.00		4.90	4.90	0.00	
84	82	0.65	0.00		5.22	5.22	0.00	
85	82	0.65	0.00		5.55	5.55	0.00	
86	82	0.65	0.00		5.87	5.87	0.00	
87	82	0.65	0.00		6.20	6.20	0.00	
88	82	0.65	0.00		6.52	6.52	0.00	
89	82	0.65	0.00		6.85	6.85	0.00	
90	72	0.89	0.00		7.29	7.29	0.00	
91	72	0.89	0.00		7.74	7.74	0.00	
92	72	0.89	0.00		8.18	8.18	0.00	
93	72	0.89	0.00		8.63	8.63	0.00	
94	72	0.89	0.00		9.07	9.07	0.00	
95	72	0.89	0.00		9.52	9.52	0.00	
96	72	0.89	0.00		9.96	9.96	0.00	
97	72	0.89	0.00		10.41	10.41	0.00	**Danger**
98	72	0.89	0.00		0.45	0.45	0.00	
99	72	0.89	0.00		0.89	0.89	0.00	
100	72	0.89	0.00		1.34	1.34	0.00	
101	72	0.89	0.00		1.78	1.78	0.00	
102	72	0.89	0.00		2.23	2.23	0.00	
103	72	0.89	0.00		2.67	2.67	0.00	
104	72	0.89	0.00		3.12	3.12	0.00	
105	72	0.89	0.00		3.56	3.56	0.00	
106	72	0.89	0.00		4.01	4.01	0.00	
107	72	0.89	0.00		4.45	4.45	0.00	
108	72	0.89	0.00		4.90	4.90	0.00	
109	82	0.65	0.00		5.22	5.22	0.00	

Table 19, (Continued)

110	89	0.00	0.73		6.32	4.13	0.73	
111	89	0.00	0.73		7.41	3.03	1.46	
112	89	0.00	0.73		8.51	1.94	2.19	
113	89	0.00	0.73		9.60	0.84	2.92	
114	89	0.00	0.73		10.70	-0.25	3.65	**Safe**
115	89	0.00	0.73		1.10	-1.10	0.73	
116	89	0.00	0.73		2.19	-2.19	1.46	
117	89	0.00	0.73		3.29	-3.29	2.19	
118	90	0.00	0.86		4.58	-4.58	3.05	
119	90	0.00	0.86		5.87	-5.87	3.91	
120	90	0.00	0.86		7.16	-7.16	4.77	
121	90	0.00	0.86		8.45	-8.45	5.63	
122	90	0.00	0.86		9.74	-9.74	6.49	
123	72	0.89	0.00		10.18	-9.29	6.49	**Safe**
124	72	0.89	0.00		0.45	0.45	0.00	
125	72	0.89	0.00		0.89	0.89	0.00	
126	72	0.89	0.00		1.34	1.34	0.00	
127	72	0.89	0.00		1.78	1.78	0.00	
128	72	0.89	0.00		2.23	2.23	0.00	
129	72	0.89	0.00		2.67	2.67	0.00	
130	72	0.89	0.00		3.12	3.12	0.00	
131	72	0.89	0.00		3.56	3.56	0.00	

Table 19 shows the derivations of the danger and safe signals for the example over time. Each line is a one-minute moving window of network traffic and the lines are spaced 10 seconds apart. The first column is the sequence number of the time series. The second column shows the BMN for that time slot. The next two columns show the danger signal or the safe signal for the corresponding BMN based on the ratio of infected/*uninfected* BMI for that node. If the ratio is higher for infected traffic, then the ratio becomes the danger signal for that time slot and the cell is highlighted in red; if the ratio is higher for uninfected traffic, then the ratio become the safe signal for that time slot and the

cell is highlighted in green. These calculations ignore any possible *inflammation* and Pathogen-Associated Molecular Pattern (PAMP) inputs (Snort alerts). The column labeled *CSM* is for the co-stimulation signal and keeps track of the amount of stimulus for an alert. When the CSM reaches a threshold, 10 in this example, then an alert is produced. The columns labeled *Danger* and *Safe* accumulate the danger and safe signals until the threshold is reached. The last column, labeled *Alert*, shows when an alert is issued and whether the alert is for danger or safety. The calculations highlighted in yellow represent a single artificial dendritic cell accumulating danger and safe signals, reaching a threshold, and then reporting the status (safe in this case) to the *body*. This table shows the calculations of 9 such artificial dendritic cells in sequential order, 5 reporting safe conditions, 3 reporting dangerous conditions, and one (the last one) not yet reporting. Multiple CSM/Danger/Alert calculations could be occurring in parallel, with a new set being started with each new line. The formulas for these calculations were provided earlier in this paper. This AIS Danger Theory analysis indicates that the subject computer was alternating between safe activity and suspicious activity, indicating that a follow up would be appropriate to see if this computer had an additional problem besides the network discovery software. This analysis also indicates the times during the traffic when a forensics analyst should look for the additional problematic activity. The computer was no longer available, however, when this was determined. An added benefit of this method is that the analysis can be done in real time indicating when the suspicious network traffic is occurring, which would allow a

technician to observe via Task Manager or other software what program is active while the suspicious traffic is taking place.

Time	BMN	Danger Signal	Safe Signal	CSM	Danger	Safe	Alert	CSM	Danger	Safe	Alert	CSM	Danger	Safe	Alert	CSM	Danger	Safe	Alert
1	5	0.00	0.94	1.41	-1.41	0.94													
2	17	0.00	1.00	2.91	-2.91	1.94		1.50	0.97	0.00									
3	88	0.00	0.77	4.07	-4.07	2.71	Safe	2.66	-0.19	0.77		1.16	-1.16	0.77					
4	89	0.00	0.73	1.10	-1.10	0.73		3.75	-1.28	1.50	Safe	2.25	-2.25	0.77					
5	89	0.00	0.73	2.19	-2.19	1.46		1.10	0.73	0.00		3.35	-3.35	0.77	Safe				
6	89	0.00	0.73	3.29	-3.29	2.19	Safe	2.19	-0.37	0.73		1.10	-1.10	0.73					
7	0	0.80	0.00	0.40	0.40	0.00		2.59	0.04	0.73		1.50	-0.70	1.53					
8	90	0.00	0.86	1.69	-0.89	0.86		3.88	-1.26	1.59	Safe	2.79	-1.99	1.53		1.29	-1.29	0.00	
9	90	0.00	0.86	2.98	-2.18	1.72		1.29	0.86	0.00		4.08	-3.28	1.53	Safe	2.58	-2.58	0.86	
10	82	0.65	0.00	3.31	-1.86	1.72	Safe	1.62	1.19	0.00						2.91	-2.26	0.86	
11	82	0.65	0.00	0.33	0.33	0.00		1.94	1.51	0.00						3.23	-1.93	0.86	Safe

Figure 46, Parallel Danger/Safe Signals

Figure 46 shows how the danger and safe alerts could be produced in parallel with a new artificial dendritic cell starting at each line. Additional columns were added to the previous table for the calculations of these additional dendritic cells. The yellow cells indicate the upper left corner of each new dendritic cell. The threshold was set to 3 for this figure in order to produce a smaller graphic. In this sample spreadsheet of the calculations, the next dendritic cell, for Time 9, would start off the page to the right. Eight dendritic cells are shown in this figure, each of them reporting safe conditions. Producing alerts in parallel such as this would provide faster and more detailed information to the security technician in real time. The concept level has changed to artificial dendritic cells indicating danger and safety in network traffic.

This system facilitates Type 1 Intrusion Detection by creating real-time alerts of network patterns which are known to be malicious. This system also facilitates Type 2 Intrusion Detection by creating real-time alerts of network patterns which are similar to (*symptoms of*) network patterns which appear to be

malicious. By indicating network patterns which are similar to other network patterns which are similar to malicious behavior, this system is also a benefit to Type 3 NID Research by providing potential ways of detecting intrusions that are currently not being detected. New Snort rules could be created as a result of using this system to monitor network traffic during times when danger signals are being produced.

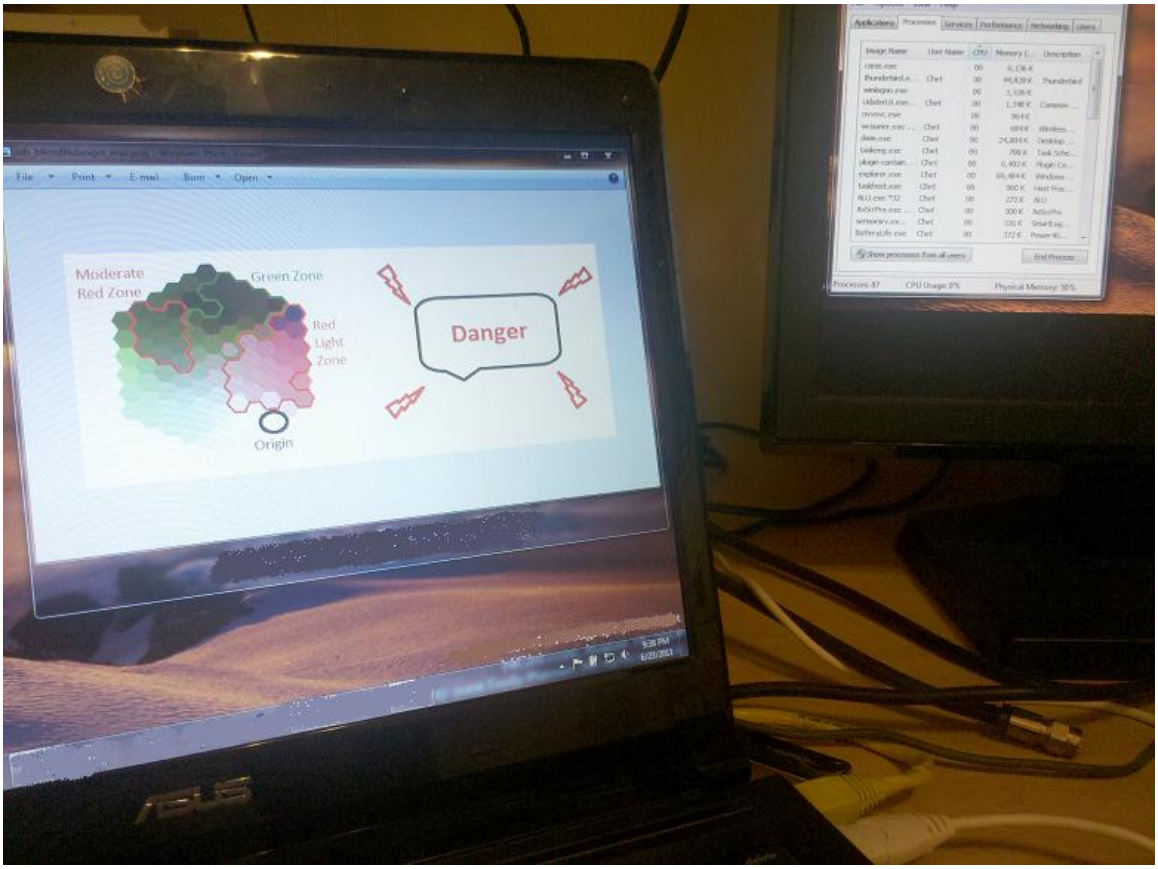


Figure 47, Real-Time Monitoring

Figure 47 is a dramatization of how the Invisible Mobile Network Bridge (IMNB) could be used for live monitoring with the trained SOM hybridized with AIS Danger Theory. Software could be developed to run on the IMNB (left) which would indicate the BMN in real time and flash danger alerts as they occur.

A security technician could take advantage of this system to use Task Manager or other analysis software to determine what processes are active on the subject computer (right) while the suspicious traffic is known to be occurring. Results from live analyses such as this could be used to create new Snort rules.

AN EVOLVING SOM

An advantage of using $|V_{\mathrm{N_{MAX}NBH}_{s,t}}|$ as the basis for the training factor is that the already trained SOM can evolve with new input data without having to start over with the training. The only existing SOM information which needs to be retained is the counts of the BMI for each node plus labeling information for each type of network traffic used as input. To continue training, use one iteration per neighborhood distance per new input vector with the inverse of $|V_{\mathrm{N_{MAX}NBH}_{s,t}}|$ as the training factor as explained earlier. The initial neighborhood distance could also start at a smaller size. Cascading calculations based on the SOM vector values would also need to be recalculated as appropriate. As the SOM gets exceedingly large with BMI, the BMI counts can be periodically fractionally reduced in order to limit long term memory of the SOM. As the SOM substantially changes or grows, it can periodically be retrained, and can potentially become a growing hierarchical SOM [63].

An advantage of the LLNIDS SOM is that the data can be collected under tightly controlled conditions. The LLNIDS SOM could be expanded into a universal SOM for broader application by obtaining input from pcap repositories which are available on the Internet. Here are a couple of example repositories:

- http://sourceforge.net/apps/mediawiki/networkminer/index.php?title=Publicly_available_PCAP_files
- https://www.openpacket.org/capture/list

A common infection is FakeAV and a sample of FakeAV network traffic is available at openpacket.org. This sample can be converted to text with tcpdump which can then be normalized and aggregated into input vectors for the SOM. Here are the first three lines from the text of the FakeAV file showing that all of the necessary information is available to create SOM input from this data:

```
     21:02:03.865392 IP (tos 0x0, ttl 127, id 0,
offset 0, flags [DF], proto UDP (17), length 67)
12.183.1.55.54226 > 8.8.8.8.53: 11851+ A?
puskovayaustanovka.ru. (39)
     21:02:04.051015 IP (tos 0x0, ttl 53, id
47400, offset 0, flags [none], proto UDP (17),
length 83) 8.8.8.8.53 > 12.183.1.55.54226: 11851
1/0/0 puskovayaustanovka.ru. A 46.161.20.66 (55)
     21:02:04.052874 IP (tos 0x0, ttl 64, id
41435, offset 0, flags [DF], proto TCP (6),
length 44) 12.183.1.55.44385 > 46.161.20.66.80:
S, cksum 0x573b (correct),
3572979361:3572979361(0) win 5840 <mss 1460>
```

Other samples of both normal and malicious traffic are also available. The concept level has changed to a theoretical evolving universal hybridized SOM/FIS/AISDT intrusion detection system.

CHAPTER 5 – DISCUSSION

In this research, six years of experience was gained doing intrusion detection on the information security team of central Information Technology for the University. The literature was reviewed for information security noting the complexity of intrusion detection. The literature for Soft Computing methods of intrusion detection was reviewed for ways of overcoming this complexity. A lack of analytical descriptions of data and intrusion detection types in the literature was noted during these reviews of the literature.

Self-Organizing Maps (SOM) were studied as a possible solution and 1D ANNaBell, a SOM, was created and placed into production for the University for network intrusion detection. 1D ANNaBell successfully found Storm Worm infections and other network security problems being the first known time that a self-trained computational intelligence discovered previously unknown feral malicious software.

3D ANNaBell, a SOM hybridized with a Fuzzy Inference System (FIS), was created as an evolution from 1D ANNaBell. 3D ANNaBell produced a full-color 3D map with landscape features that was useful in forensics, profiling of networks, and in understanding the SOM.

A high-level overall SOM+ diagnostic system was created to join several components of intrusion detection together, much like a medical center where numerous diagnostic methods are utilized in decision-making. Based on the concept of a Local Landline Network Intrusion Detection System (LLNIDS), an Invisible Mobile Network Bridge (IMNB) was created to collect data; the LLNIDS

Types of Intrusion Detection were defined, the LLNIDS Computational Model was created, and the LLNIDS hybridized SOM/FIS/AISDT network analysis system was invented.

The LLNIDS SOM includes dynamic methods of changing the neighborhood sizes and of determining the training factors. A neighborhood size changes when the best matching nodes stop changing and the training factors are initially based on the number of best matching inputs for each neighborhood. The training factors then continue changing based on the amount of movement of the nodes. These dynamic methods allow the creation of a *live* SOM that, once initially trained, can continue to train as new inputs are entered. The LLNIDS SOM can also in theory be expanded to become a universal SOM with inputs from files of network traffic from other sources, such as from Internet repositories.

The LLNIDS FIS helps to overcome the *black box* environment of the SOM by allowing security technicians to correlate fuzzy descriptions of types of network traffic with a full color 3D SOM. This FIS, along with Svensson Josang (SJ) Fusion, can also be used to correlate findings with the AIS Danger Theory alerts.

The LLNIDS AIS Danger Theory works in coordination with the LLNIDS SOM to produce parallel periodic alerts signifying dangerous or safe network traffic.

The SOM+ system as a whole allows a security technician to take samples of network traffic for analysis or to monitor network traffic in real time.

Samples of infected computers can be taken while connected to the Internet or to a simulated network. The network traffic can be profiled or monitored live to see when suspicious network traffic is occurring. This can be correlated with observations of software which is running on a subject computer at the time of the suspicious traffic.

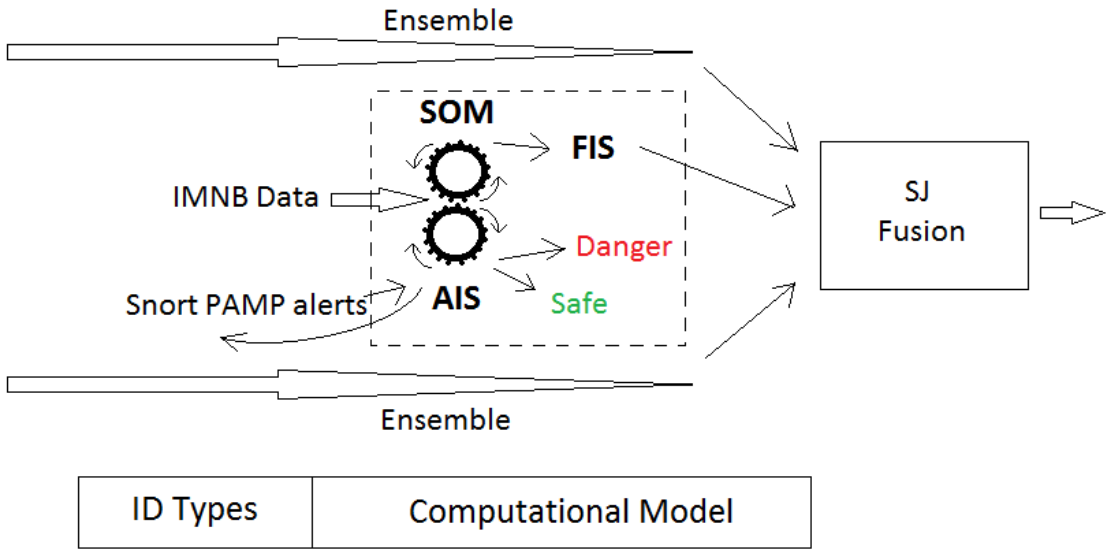


Figure 48, SOM+ Diagnostic System

Figure 48 summarizes the SOM+ Diagnostic System. IMNB data is fed into the hybridized SOM/FIS/AIS while Snort data is also fed into the AISDT. The SOM produces danger/safe signals read by the AISDT, re-trains itself in real time, and sends additional output to the FIS. The AISDT creates artificial dendritic cells to aggregate signals and periodically send danger/safe alerts. The FIS produces human intuitive output and also sends output to SJ Fusion. The SOM/AISDT aspects can be used in forensics to create new Snort rules. These components are all potentially part of a larger ensemble system consisting of other intrusion detection components which also send output to the SJ Fusion,

which produces a single output. The danger/safe signals can be separate output or can be sent to SJ Fusion to become part of a single output. The entire system is based on a foundation of ID Types and the Computational Model. The system detects Type 1 and Type 2 intrusions with type 1 and 2 methods and alerts. This is type 1, 2, and 3 research: it improves types 1 and 2 intrusion detection and potentially detects Type 3 intrusions by comparing similarities of network traffic with similarities of known types of dangerous network traffic.

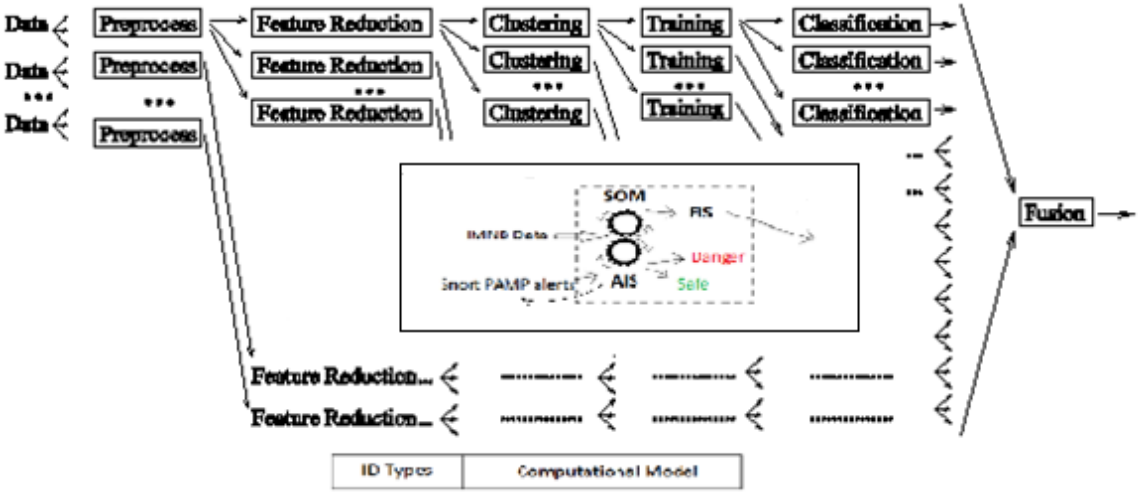


Figure 49, The Larger Structure

Figure 49 illustrates how the SOM/FIS/AISDT hybrid from the previous figure can drop into the larger generic structure of the diagnostic system. The SOM+ Diagnostic System is modular because the various components can be developed and/or programmed independently, yet they work cohesively. The SOM+ Diagnostic System is multitaskable because it can be used to collect data, on collected data, or for live analysis. It can be multitasked for detection capability, profiling, or forensics. AIS Danger Theory, unlike traditional AIS, is efficient and can be adjusted for scalability. The LLNIDS SOM scales well for

live retraining because of the dynamic methods of changing neighborhood sizes and training factors. The 3D full color SOM map coupled with FIS facilitates intuitive understanding by security technicians. The red and green zones on a live map along with danger and safe alerts also facilities intuitiveness. The dynamic methods of the LLNIDS SOM allow for adaptability to quickly changing scenarios. Periodic full retraining of the SOM needs either time or a high performance computing environment or can potentially be mitigated with a growing hierarchical SOM [63]. The Invisible Mobile Network Bridge can be a simple laptop to monitor a single system up to a high performance system to monitor a network, depending upon the needs of the system.

Advantages of the SOM+ Diagnostic System are:

- The full color 3D SOM is more intuitive than previous maps.
- AIS Danger Theory signals can provide real-time alerts and forensics information.
- The FIS presents a more intuitive SOM interface for security technicians.
- Newly defined intrusion types aid scientific analysis.
- The computational model unifies how the data is presented in numerous concept levels and methods.
- The IMNB allows more pure captures of network traffic without potential influence by malware and aids in forensics.
- The IMNB can be set up for data capture without disturbing a running subject computer.

- The IMNB can monitor a subject computer on a simulated network so that an infected computer does not need to endanger a real network for data collection.
- Data collected from the IMNB can also be analyzed by other kinds of software such as Snort and Wireshark.
- The system can potentially become a universal SOM by accepting network traffic data from repositories on the Internet.
- The system is modular, multitaskable, scales well, is adaptable to quickly changing scenarios, and uses relatively few resources.
- Emergence.

Disadvantages of the SOM+ Diagnostic System are:

- Re-training the SOM as it grows large will take considerably more resources.
- The expertise of a knowledge engineer is required.

Novelties in this research are:

- The complex hybridization of SOM, FIS, and AIS Danger Theory.
- Automated changing of neighborhood sizes and the training factor in the SOM.
- The IMNB for data collection and network monitoring.
- The computational model.
- The new intrusion detection types.
- A diagnostic system based on a medical center metaphor.
- Combining clustering and classification in the SOM.

- The Imprecision Principle.
- Listing the numerous concept levels involved in processing data from binary network traffic to sophisticated Soft Computing methods.

CHAPTER 6 – CONCLUSION

Information security is a complex and inexact process which lends itself to Soft Computing techniques. Current methods are stymied by continuously evolving malware techniques and the situation that determination of a virus in all cases is undecidable. Network security is an evolutionary system where both sides continually evolve to outwit the other side. Security technicians are not capable of knowing what every single packet on the Internet is doing. This research showed that Soft Computing methods can successfully be used in addition to other methods for intrusion detection. The SOM 1D ANNaBell discovered previously unknown network security problems. The SOM 3D ANNaBell with FIS demonstrated that the *black box* internal workings of the SOM can be displayed more intuitively for security technicians and that the system can be used for security profiling and forensics. The SOM/FIS/AISDT hybrid showed that artificially immunity danger theory can be hybridized with the SOM to produce danger and safe signals for security technicians in real time for live analysis, as well as for profiling, forensics, and intrusion detection, in a manner which can potentially be evolving and universal.

Like the situation with Lewis and Clark when they headed West in 1804 to see just how large the Louisiana Purchase really was: there is a huge amount of unexplored territory in Soft Computing to be investigated. The SOM+ Diagnostic System in this research is an archetype from which to move forth.

REFERENCES

1. U.S. Air Force: Cyberspace Operations. Air Force Doctrine Document 3-12 (2010)

2. Amoroso, E.: Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response. Intrusion.Net Books (1999)

3. Denning, D.: An Intrusion-Detection Model. IEEE Transactions on Software Engineering 13(2), 118-131 (1986)

4. Young, C.: Taxonomy of Computer Virus Defense Mechanisms. In : The 10th National Computer Security Conference Proceedings (1987)

5. Lunt, T.: Automated Audit Trail Analysis and Intrusion Detection: A Survey. In : Proceedings of the 11th National Computer Security Conference, Baltimore, pp.65-73 (1988)

6. Lunt, T.: A Survey of Intrusion Detection Techniques. Computers and Security 12, 405-418 (1993)

7. Vaccaro, H., Liepins, G.: Detection of Anomalous Computer Session Activity. In : Proceedings of the 1989 IEEE Symposium on Security and Privacy (1989)

8. Helman, P., Liepins, G., Richards, W.: Foundations of Intrusion Detection. In : Proceedings of the IEEE Computer Security Foundations Workshop V (1992)

9. Denault, M., Gritzalis, D., Karagiannis, D., Spirakis, P.: Intrusion Detection: Approach and Performance Issues of the SECURENET System. Computers and Security 13(6), 495-507 (1994)

10. Forrest, S., Allen, L., Perelson, A., Cherukuri, R.: Self-Nonself Discrimination in a Computer. In : Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, Los Alamos, cA (1994)

11. Crosbie, M., Spafford, G.: Defending A Computer System Using Autonomous Agents., COAST Laboratory, Department of Computer Science, Purdue University, West Lafayette, Indiana, USA (1994)

12. Kumar, S., Spafford, E.: An Application of Pattern Matching in Intrusion Detection., Purdue University (1994)

13. Ilgun, K., Kemmerer, R., Porras, P.: State Transition Analysis: A Rule-Based Intrusion Detection Approach. IEEE Transactions on Software Engineering 21(3), 181-199 (March 1995)

14. Esmaili, M., Safavi-Naini, R., Pieprzyk, J.: Evidential Reasoning in Network Intrusion Detection Systems. In : Proceedings of the First Australasian Conference on Information Security and Privacy, pp.253-265 (1996)

15. Debar, H., Dacier, M., Wespi, A.: Towards a Taxonomy of Intrusion-Detection Systems. Computer Networks 31, 805-822 (1999)

16. Bace, R.: Intrusion Detection. MacMillan Technical Publishing (2000)

17. Marin-Blazquez, J., Perez, G.: Intrusion Detection Using a Linguistic Hedged Fuzzy-XCS Classifier System. Soft Computing -- A Fusion of Foundations, Methodologies, and Applications 13(3), 273-290 (2008)

18. Lunt, T.: IDES: An Intelligent System for Detecting Intruders. In : Proceedings of Computer Security, Threat and Countermeasures (1990)

19. Cannady, J.: Applying Neural Networks for Misuse Detection. In : Proceedings of the 21st National Information Systems Security Conference, pp.368-381 (1998)

20. Kayacik, H., Zincir-Heywood, , Heywood, M.: Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets. In : Proceedings of the Third Annual Conference on Privacy, Security, and Trust, St. Andrews, New Brunswick, Canada (2005)

21. Lee, S., Heinbuch, D.: Training a Neural-Network Based Intrusion Detector to Recognize Novel Attacks. IEEE Transactions on Systems, Man, and Cybernetics, Part A, 294-299 (2001)

22. Mukkamala, S., Janoski, G., Sung, A.: Monitoring System Security Using Neural Networks and Support Vector Machines. In : PRoceedings of the International Workshop on Hybrid Intelligent Systems, pp.121-138 (2001)

23. Zhang, Z., Shen, H.: Application of Online Training SVMs for Real-Time Intrusion Detection with Different Considerations. Computer Communications 28(12), 1428-1442 (2005)

24. LaRoche, P., Zincir-Heywood, : 802.11 De-authentication Attack Detection using Genetic Programming. In : Proceedings of the 9th European Conference on Genetic Programming, vol. 3905, pp.1-12 (2006)

25. Livadas, C., Walsh, B., Lapsley, D., Strayer, T.: Using Machine Language Learning Techniques to Identify Botnet Traffic. In : Proceedings of the Second IEEE LCN Workshop on Network Security (WNS), Tampa, Florida, USA (2006)

26. Mukkamala, S., Sung, A., Abraham, A.: Designing Intrusion Detection Systems: Architectures and Perspectives. The International Engineering Consortium (IEC) Annual Review of Communications 57, 1229-1241 (2004)

27. Mukkamala, S., Sung, A.: Identifying Key Features fo Intrusion Detection Using Neural Networks. In : Proceedings of the ICCC International Conference on Computer Communications, pp.1132-1138 (2002)

28. Sung, A., Mukkamala, S.: Identifying Important Features for Intrusion Detection Using Suport Vector Machines and Neural Networks. In : Proceedings of the International Symposium on Applications and the Internet (SAINT 2003), pp.209-217 (2003)

29. Abraham, A., Jain, R.: Soft Computing Models for Network Intrusion Detection Systems. In: Soft Computing Models for Network Intrusion Detection Systems. (Accessed 2004) Available at: http://arxiv.org/ftp/cs/papers/0405/0405046.pdf

30. Chen, Y., Abraham, A., Yang, J.: Feature Deduction and Intrusion Detection using Flexible Neural Trees. In : Proceedings of the Second IEEE International Symposium on Neural Networks (ISNN 2005), pp.439-446 (2005)

31. Chimphlee, W., Abdullah, A., Sap, M., Chimphlee, S., Srinoy, S.: Integrating Genetic Algorithms and Fuzzy c-Means for Anomaly Detection. In : Proceedings of IEEE Indicon, Chennai, India (2005)

32. Newsome, J., Karp, B., Song, D.: Paragraph: Thwarting Signature Learning by Training Maliciously. In Zamboni, D., Kruegel, C., eds. : Proceedings of Recent Advances in Intrusion Detection, 9th International Symposium, RAID 2006, Hamburg, Germany, vol. 4219,

pp.81-105 (2006)

33. Cohen, F.: Computer Viruses: Theory and Experiments. Computers & Security 6(1), 22-35 (1987)

34. Me, L.: A Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis. In : Proceedings of Recent Advances in Intrusion Detection (RAID '98), France (1998)

35. Pietraszek, T.: Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection. In : Proceedings of Recent Advances in Intrusion Detection, 7th International Symposium, RAID 2004, Sophia Antipolis, France, vol. 3224, pp.102-124 (2004)

36. Zhou, J., Heckman, M., Reynolds, B., Carlson, A., Bishop, M.: Modeling Network Intrusion Detection Alerts for Correlation. ACM Transactions on Information and System Security (TISSEC) 10(1), 1-31 (2007)

37. Bolzoni, D., Crispo, B., Etalle, S.: ATLANTIDES: An Architecture for Alert Verification in Network Intrusion Detection Systems. In : Proceedings of the 21st Large Installation System Administration Conference (LISA '07) (2007)

38. Zadeh, L.: History; BISC During 90's. In: BISC Program. (Accessed March 3, 1994) Available at: http://www-bisc.cs.berkeley.edu/BISCProgram/History.htm

39. Bayes, T.: An Essay Towards Solving a Problem in the Doctrine of Chances. Philosophical Transactions of the Royal Society of London 53, 370-418 (1763)

40. McCulloch, W., Pitts, W.: A Logical Calculus of the Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics 5, 115-133 (1943)

41. Zadeh, L.: Fuzzy Sets. Information and Control 9, 338-353 (1965)

42. Bedau, M., Humphreys, P.: Emergence: Contemporary Readings in Philosophy and Science. The MIT Press, Cambridge (2008)

43. Zadeh, L.: Fuzzy Logic, Neural Networks, and Soft Computing. Communications of the ACM 37(3), 77-84 (1994)

44. Zadeh, L.: Roles of Soft Computing and Fuzzy Logic in the Conception, Design and Deployment of Information/Intelligent Systems. In

Kaynak, O., Zadeh, L., Turksen, B., Rudas, I., eds. : Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications 162. Springer (1998)

45. Langin, C., Rahimi, S.: Soft Computing in Intrusion Detection: The State of the Art. Journal of Ambient Intelligence and Humanized Computing 1(2), 133-145 (2010)

46. Depren, O., Topallar, M., Anarim, E., Ciliz, M.: An Intelligent Intrusion Detection System (IDS) for Anomaly and Misuse Detection in Computer Networks. Expert Systems with Applications 29(4), 713-722 (2005)

47. Katar, C.: Combining Multiple Techniques for Intrusion Detection. International Journal of Computer Science and Network Security 6(2B), 208-218 (2006)

48. Mukkamala, S., Sung, A., Abraham, A.: Intrusion Detection Using Ensemble of Soft Computing Paradigms. In : Proceedings of the Third International Conference on Intelligent Systems Design and Applications, Advances in Soft Computing, pp.239-248 (2003)

49. Svensson, H., Josang, A.: Correlation of Intrusion Alarms wth Subjective Logic. In : Proceedings of the Sixth Nordic Workshop on Secure IT Systems (NordSec 2001), Copenhagen, Denmark (2001)

50. Greensmith, J., Aickelin, U., Cayzer, S.: Detecting Danger: The Dendritic Cell Algorithm. In : Robust Intelligent Systems. IGI Publishing (2008)

51. Mukkamala, S., Sung, A., Abraham, A.: Intrusion Detection using an Ensemble of Intelligent Paradigms. Journal of Network and Computer Applications 28, 167-182 (2005)

52. Langin, C., Zhou, H., Rahimi, S.: A Model to Use Denied Internet Traffic to Indirectly Discover Internal Network Security Problems. In : Proceedings of the First IEEE International Workshop on Information and Data Assurance, Austin, Texas, USA (2008)

53. Langin, C., Zhou, H., Gupta, B., Rahimi, S., Sayeh, M.: A Self-Organizing Map and its Modeling for Discovering Malignant Network Traffic. In : Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Cyber Security, Nashville, TN, USA (2009)

54. Kohonen, T.: Self-Organizing Maps 30. Springer-Verlag, Berline Heidelberg New York (2001)

55. Langin, C., Che, D., Wainer, M., Rahimi, S.: Visualization of Network Security Traffic using Hexagonal Self-Organizing Maps. In : Proceedings of the 22nd International Conference on Computers and Their Applications in Industry and Engineering (CAINE-2009), San Franciso, CA, USA, pp.1-6 (2009)

56. Langin, C., Che, D., Wainer, M., Rahimi, S.: SOM with Vulture Fest Model Discovers Feral Malware and Visually Profiles the Security of Subnets. Internation Journal of Computers and Their Applications (IJCA) 17(4), 241-249 (2010)

57. Langin, C., Wainer, M., Rahimi, S.: ANNaBell Island: A 3D Color Hexagonal SOM for Visual Intrusion Detection. Internation Journal of Computer Science and Information Security 9(1), 1-7 (2011)

58. Hoglund, A., Hatonen, K.: Computer Network User Behavior Visualization using Self-Organizing Maps. In : Proceedings of the International Conference on Artificial Neural Networks (ICANN), pp.899-904 (1998)

59. Ultsch, A., Siemon, H.: Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis. In : Proceedings of the International Neural Network Conference (INNC '90), pp.205-308 (1990)

60. Lichodzijewski, P., Zincir-Heywood, A., Heywood, M.: Host-Based Intrusion Detection Using Self-Organizing Maps. In : Proceedings of the IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks (IJCNN) (2002)

61. Kayacik, G., Zincir-Heywood, A., Heywood, M.: On the Capability of an SOM Based Intrusion Detection System. In : Proceedings of the IEEE International Joint Conference on Neural Networks, pp.1808-1813 (2003)

62. Ramadas, M., Ostermann, S., Tjaden, B.: Detecting Anomalous Network Traffic with Self-Organizing Maps. In Vigna, G., Jonsson, E., Kruegel, C., eds. : Proceedings of Recent Advances in Intrusion Detection, 6th International Symposiu, RAID, 2003, Pittsburgh, PA, USA, vol. 2820, pp.36-54 (2003)

63. Vicente, D., Vellido, A.: Review of Hierarchical Models for Data Clustering and Visualization. Tendencias de la Mineria de Datos in Espana, Red Espanola de Mineria de Datos (2004)

64. Tauriainen, A.: A Self-learning System for P2P Traffic Classification.

Helsinki University of Technology, Helsinki (2005)

65. Wetmore, L., Zincir-Heywood, A., Heywood, M.: Training the SOFM Efficiently: An Example from Intrusion Detection. In : Proceedings of the IEEE International Joint Conference on Neural Networks, IJCNN 2005, pp.1575-1580 (2005)

66. Kayacik, H., Zincir-Heywood, A.: Using Self-Organizing Maps to Build an Attack Map for Forensic Analysis. In : Proceedings of the ACM International Conference on Privacy, Security, and Trust (PST 2006), pp.285-293 (2006)

67. Bolzoni, D., Etalle, S., Hartel, P.: POSEIDON: A 2-Tier Anomaly-Based Network Intrusion Detection System. In : Proceedings of the Fourth IEEE International Workshop on Information Assurance (IWIA 2006) (2006)

68. Bolzoni, D., Etalle, S.: Approaches in Anomaly-based Network Intrusion Detection. In : Intrusion Detection Systems. Springer (2008) 1-16

69. Luo, J.: Mining Fuzzy Association Rules and Fuzzy Frequency Episodes for Intrusion Detection. International Journal of Intelligent Systems 15(8), 687-704 (2000)

70. Noel, S., Wijesekera, D., Youman, C.: Modern Intrusion Detection, Data Mining, and Degrees of Attack Guild. In : Applications of Data Mining in Computer Security. Kluwer (2002)

71. Tillapart, P., Thumthawatworn, T., Santiprabhob, P.: Fuzzy Intrusion Detection System. Assumption University Journal of Technology (AU J.T.) 6(2), 109-114 (2002)

72. Dasgupta, D., Gonzalez, F., Yallapu, K., Gomez, J., Yarramsettii, R.: CIDS: An Agent-Based Intrusion Detection System. Computers & Security 24(5), 387-398 (2005)

73. Wang, W., Daniels, T.: A Graph Based Approach Towards Network Forensics Analysis. ACM Transactions on Information and System Security 12(1) (2008)

74. Su, M.-Y., Yu, G.-J., Lin, C.-Y.: A Real-Time Network Intrusion Detection System for Large-Scale Attacks Based on an Incremental Mining Approach. Computers & Security, 301-309 (2009)

75. Tajbakhsh, A., Rahmati, M., Mirzaei, A.: Intrusion Detection using Fuzzy

Associate Rules. Applied Soft Computing 9, 462-469 (2009)

76. Kim, J., Bentley, P.: An Evaluation of Negative Selection in an Artificial Immunce System for Network Intrusion Detection. In : Proceedings of the 2001 Genetic and Evolutionary Computation Conference, pp.1330-1337 (2001)

77. Matzinger, P.: Tolerance, Danger, and the Extended Family. Annual Revue of Immunology 12, 991-1045 (1994)

78. Aickelin, U., Cayzer, S.: The Danger Theory and Its Application to AIS. In : 1st International Conference on AIS, pp.141-148 (2002) http://www.hpl.hp.com/techreports/2002/HPL-2002-244.pdf (accessed 3/12/11).

79. Powers, S., He, J.: A Hybrid Artificial Immune System and Self Organising Map for Network Intrusion Detection. Information Sciences 178(15), 3024-3042 (2008)

80. Fu, H., Li, X.: A Bio-Inspired Multi-Tissues Growing Algorithm for IDS Based on Danger Theory and Data Fields. In : Proceedings of The 7th World Congress on Intelligent Control and Automation, Chongqing, China (2008)

81. Li, X., Fu, H., Huang, S.: Design of a Dendritic Cells Inspired Model Based on Danger Theory for Intrusion Detection. In : Proceedings of the IEEE International Conference on Networking, Sensing and Control (ICNSC 2008), pp.1137-1141 (2008)

82. Dasgupta, D., Nino, L.: Immunological Computation. CRC Press (2009)

83. Wu, S., Banzhaf, W.: The Use of Computational Intelligence in Intrusion Detection Systems: A Review. Applied Soft Computing 10(1), 1-35 (July 2010)

84. Kulis, A., Rahimi, S., Lee, Y.-C.: Finding Danger using Fuzzy Dendritic Cells. In : Proceedings of the 2011 IEEE Workshop on Hybrid Intelligent Models and Applications, Paris, France (2011)

85. Tcpdump/Libpcap: Tcpdump/Libpcap Public Repository. In: Tcpdump.org. Available at: http://www.tcpdump.org/

86. Chebrolu, S., Abraham, A., Thomas, J.: Feature Deduction and Ensemble Design of Intrusion Detection Systems. Computers and Security 24(4), 295-307 (2005)

87. Mukkamala, S., Sung, A.: Identifying Significant Features for Network Forensics Analysis Using Artificial Intelligent Techniques. International Journal on Digital Evidence (IJDE) 1(4) (2003)

88. OpenBSD: PF: The OpenBSD Packet Filter. In: openbsd.org. Available at: http://www.openbsd.org/faq/pf

89. Roesch, M.: Snort -- Lightweight Intrusion Detectin for Networks. In : 13th USENIX Conference on System ADministration (LISA '99), pp.229-238 (1999)

90. Wireshark Foundation: Wireshark. In: Wireshark. Available at: http://www.wireshark.org/

91. YouTube, LLC: YouTube. In: YouTube. Available at: http://www.youtube.com/

92. Eastman, P.: Art of Illusion Home. In: Art of Illusion. Available at: http://www.artofillusion.org

93. Negnevitsky, M.: Artificial Intelligence: A Guide to Intelligent Systems. Addison Wesley (2002)

94. O'Gorman, M., Donnenberg, A.: Handbook of Human Immunology. CRC Press, Boca Raton, USA (2008)

95. Parham, P.: The Immune System 3rd edn. Garland Science, London and New York (2009)

96. Nelson, D., Cox, M.: Lehninger Principles of Biochemistry 3rd edn. Worth Publishers, New York, USA (2000)

97. Clark, D., Russell, L.: The Molecular Defense Initiative: Your Immune System at Work. In : Molecular Biology Made Simple and Fun. Cache River Press, Vienna, USA (1997) 357-381

98. Hungenberg, T., Echert, M.: INetSim Project Homepage. In: INetSim: Internet Services Simulation Suite. (Accessed 2010) Available at: http://www.inetsim.org/index.html

99. Wall, L.: The Perl Programming Language. In: www.perl.org. Available at: http://www.perl.org

100. Python Software Foundation: Python Programming Language -- Official

Website. In: Python Programming Language -- Official Website. (Accessed 2011) Available at: http://www.python.org

101. Cruse, J., Lewis, R.: Illustrated Dictionary of Immunology. CRC Press, Boca Raton, USA (2009)

APPENDICES

APPENDIX A, REFERENCE OF VARIABLE NAMES

Below is a listing of the variable names which are used consistently throughout this paper.

- α Alpha. The training factor.
- $\alpha_{s,t}$ Alpha, size, and time. The training factor for a certain neighborhood time for a certain iteration of training.
- Δ Delta. The change in d_S
- $\Delta \mathrm{V}_{\mathrm{N}_i}$ Change, vector, node, index (SOM). The change in vector space for node i.
- BMI Best Matching Inputs. A label.
- BMN Best Matching Node. A label.
- d Distance (SOM). Can be the difference in space between two location vectors or can be the neighborhood distance between two nodes, depending upon the context.
- d_{V_I,V_N} Distance, vector, input, node (SOM). The distance in space between an input vector and a node vector.
- D Data. A set of V (vectors) or a matrix, depending upon the context.
- D_I Data, input. Any of D_{TRAIN}, D_{TEST}, or D_{REAL}.
- D_{N} Data, nodes (SOM). A set of the node vectors V of a SOM. A matrix D that represents a SOM.
- D_{REAL} Data, real (SOM). The set of data D that is real input for a SOM used in production.

- D_{TEST} Data, testing (SOM). The set of data D that is used for testing a SOM.
- D_{TRAIN} Data, training (SOM). The set of data D that is used for training a SOM.
- e Element. A generic element in a set.
- E Event. A set of e (elements) derived from a T (transmission set).
- i Index.
- I Intrusion. A set of R (records) for an intrusion.
- I_1 Intrusion, Type 1. A kind of I (intrusion).
- I_2 Intrusion, Type 2. A kind of I (intrusion).
- I_3 Intrusion, Type 3. A kind of I (intrusion).
- $I_1^D( )$ Intrusion Detection, Type 1. A function.
- $I_2^D( )$ Intrusion Detection, Type 2. A function.
- $I_3^D( )$ Intrusion Detection, Type 3. A function.
- L Log. A set of R (records).
- m Meta-data. Items of data about e (elements) in an E (event set).
- M Meta-data. A set of m (meta-data items)
- n_D Number, Data. The number of elements in a Data set.
- n_E Number, Event. The number of elements in an Event set.
- n_M Number, Meta-data. The number of elements in a Meta-data set.

- n_{N} Number, node. The number of elements in D_{N} (Data, Nodes). The number of nodes in a SOM.
- n_T Number, Transmission. The number of objects in a Transmission set.
- n_V Number, Vector. The number of elements in a Vector set, vector, or matrix.
- NBH Neighborhood. A label.
- o Object. An element of a T (transmission) set.
- R Record. A set including elements from both M (meta-data) and E (events).
- t Time (SOM). A count of the iterations in the training
- T Transmission. A set of o (objects) representing a frame or packet which is transmitted over a network.
- V Vector. An n-tuple of real numbers. May be perceived as being a set, a vector, a matrix, or in other ways depending upon the context.
- V_a Vector, aggregate. A V (vector) containing aggregate information.
- V_{BMN} Vector, Best Matching Node (SOM). The V (vector) for the Best Matching Node.
- V_d Vector, detail. A V (vector) containing detail information.
- V_{I} Vector, input. A V (vector) used as input.
- V_{N} Vector, node (SOM). A V (vector) for a node.

- $|V_{\mathrm{N}}|$ Vector, node, count (SOM). The number of BMI for a node.
- $|V_{\mathrm{N}_i}|$ Vector, node, index, count (SOM). The number of BMI for node i.
- $V_{\mathrm{N}_i\mathrm{N}_{s,t}}$ Vector, node, neighborhood, size, time (iteration) (SOM). The neighborhood for Node i with a neighborhood size of s for training iteration t.
- $|V_{\mathrm{N}_i\mathrm{NBH}_{s,t}}|$ Vector, node, neighborhood, size, time (iteration), count (SOM). The number of BMI for the neighborhood for Node i with a neighborhood size of s for training iteration t.
- $|V_{\mathrm{N}_{\mathrm{MAX}}\mathrm{NBH}_{s,t}}|$ Vector, node, maximum, neighborhood, size, time (iteration), count. The maximum count of MBIs for a neighborhood of size s in training iteration t.

APPENDIX B, CONCEPT LEVELS

The concept levels in this research are presented in a general ascending order beginning with binary code from network transmissions and progressing to abstract Soft Computing ideas. Most concept levels as presented increase the abstraction of the data. Here is a summary of the concept levels discussed in this research.

1. A network transmission of objects.
2. A set of elements.
3. A set of meta-data about the set of elements.

4. A set that includes both the elements and meta-data about those elements
5. A collection of sets of meta-data and elements.
6. A matrix.
7. A set of matrices.
8. Generalized n-tuples.
9. A multidimensional solution space.
10. A topological graph.
11. A feature map of vector dimensions.
12. A feature map, such as landscape features, representing groups of vector dimensions.
13. Fuzzy inferences from the feature map.
14. Crisp output from the fuzzy inferences.
15. Individual danger signals.
16. Opinions of danger signals.
17. Discounted opinions of danger signals
18. A fusion of danger signals.
19. Antigen as an indicator of an intrusion.
20. Antigen as a label which just identifies something.
21. A dendritic cell algorithm being a fusion method for intrusion detection.
22. Snort and SOM outputs as *biological* inputs to artificial dendritic cells.

23. Artificial dendritic cells indicating danger and safety in network traffic.
24. A theoretical evolving universal hybridized SOM/FIS/AISDT intrusion detection system.

APPENDIX C, BIOLOGICAL TERMS

These definitions are generalized for non-biologist readers in order to provide enough information to show context for the Artificial Immune System Danger Theory. These definitions are not intended to satisfy strict biological requirements. They are paraphrased from [50], [77], [84], [94], [95], [97], and [101]. See immunology texts such as the ones cited for more strict biological definitions.

- Activated: The description of an element of the immune system which has successfully found an antigen. A dendritic cell which has reached the threshold to make a decision.
- Adaptive Immune System: One of two types of human immunity. The adaptive immune system has two parts: humoral immunity and cell-mediated immunity. The other type of human immunity is innate.
- AIS: Artificial Immune System.
- AISDT: Artificial Immune System Danger Theory.
- Antibody: A substance secreted by B-cells in response to an antigen.

- Antigen: Something which reacts with antibodies and/or T-cell receptors. Potentially an invader. Potentially non-self in self/non-self immunity.
- Antigen Presenting Cell (APC): A cell which presents antigens on its cell boundary where they can interact with T-cells.
- APC: Antigen Presenting Cell.
- Apoptosis: Intentional death of an unwanted cell for the good of the person.
- Artificial Immune System: A system of protection based on the human immune system.
- B-Cell: An antibody producing cell in the humoral immune system. The *B* originally stood for the bursa of Fabricius where it matures in birds, but it can also stand for *bone marrow* where it matures in humans.
- B-Cell Receptor: An antibody which has been retained on the cell wall of a B-cell. Compare with T-cell receptor.
- B-Lymphocyte: B-cell.
- Cell-Mediated Immunity: One of two kinds of adaptive immunity. Cell-mediated immunity is mediated by T-cells. The other kind of adaptive immunity is humoral immunity.
- Clonal Selection: The duplication of antibodies after an antibody matches with an antigen.
- Cytokine: A chemical communicator between cells.

- Cytotoxic T-Cell: Killer T-Cell.
- Danger Context: One of two possible outputs of the Dendritic Cell Algorithm (DCA). A danger context indicates that the immune system should be activated. The other possible output is a safe context.
- Danger Signal: Molecules from stressed or dying cells or pathogens.
- Danger Theory: A theory of the immune system which considers indications of danger for triggering the immune system. Compare to self/non-self immunity.
- Dendritic Cell: A professional Antigen Presenting Cell (APC) with branched structures that can activate T-cells. They are mature (activated) or immature (non-activated) depending upon whether or not they can activate T-cells. [101]
- Detector Cell: An antibody or T-cell in an AIS. A detector cell has various states: a semi-mature state while it undergoes stochastic sampling; a mature state indicating that it was not eliminated for being self; an activated state means that it matches an antigen; and, a memory state means that the activated state was verified by a human.
- Effector Cell: An activated experienced cell, ready to kill targets, help B-cells or macrophages, or secrete antibody.
- Epitope: The part of an antigen which can be matched.

- Experienced T-Cell: A T-cell that has responded at least once to an antigen.
- Helper T-Cell: A type of T-cell that activates B-cells to make more antigen. Compare with killer T-cell.
- Humoral Immunity: One of two kinds of adaptive immunity. Humoral immunity is extra-cellular in bodily fluids and is mediated by antibodies produced by B-cells. The other kind of adaptive immunity is cell-mediated immunity.
- Immature Dendritic Cell: A dendritic cell which has not made a decision. Compare with semi-mature and mature dendritic cell.
- Immune System: A system of protection. The two general types of the human immune system are innate, and adaptive.
- Immunoglobulin: A mature B-cell stimulated by an antigen, such as an antibody.
- Inflammation: An example response of the innate immune system involving increased circulation of the blood and other activities.
- Innate Immune System: One of two types of human immunity. The innate immune system has a fast non-specific response with no memory and is usually short-lived. The other type of human immunity is adaptive.
- Interferon: A type of cytokine.

- Killer T-Cell: A type of T-Cell that kills an infected or cancerous cell. Compare with Helper T-Cell. Contrast with Natural Killer Cell.
- Langerhans Cell: A type of dendritic cell.
- Lymphocyte: A B-cell or T-cell.
- Macrophage: A kind of non-specific phagocyte that interacts with both B-cells and T-cells.
- Major Histocompatibility Complex (MHC): A part of a cell wall that presents an antigen.
- Mature B-Cell: A B-cell whose antibody has matched an antigen.
- Mature Dendritic Cell: A dendritic cell with a matching antigen and with a danger signal.
- Mature T-Cell: A T-cell activated by an APC.
- MHC: Major Histocompatibility Complex.
- NK: Natural Killer Cell.
- Natural Killer Cell (NK): A cell in the innate immune system that kills tumor cells and certain virus-infected cells, and does not require prior contact with antigen. Contrast with killer T-cell.
- Neutrophil: A phagocyte in the Innate Immune System that ingests and destroys foreign bodies.
- Non-Self: Chemical compounds which an organism considers to be not part of itself.
- PAMP: Pathogen-Associated Molecular Pattern.

- Pathogen: An agent that can produce disease.
- Pathogen-Associated Molecule Pattern (PAMP): Motifs of molecules that are not found on host tissues.
- Phagocyte: A cell that ingests particles.
- Plasma Cell: A mature B-Cell, one whose antibody has matched an antigen.
- Professional Antigen Presenting Cell (APC): A macrophage, B-cell, or dendritic cell that can activate a T-cell. [101] Any cell that can activate virgin T-cells.
- Safe Context: One of two possible outputs of the Dendritic Cell Algorithm (DCA). A safe context indicates that the immune system should not be activated. The other possible output is a danger context.
- Safe Signal: A signal of safe to a dendritic cell.
- Self: Chemical compounds which an organism considers to be part of itself.
- Self/non-self immunity: An explanation of immunity in which non-self entities trigger the immune system. Compare to Danger Theory.
- Semi-Mature Dendritic Cell: A dendritic cell with a safe context.
- T-Cell: The type of cell which mediates in the cell-mediated immune system. The *T* is for *thymus*, where the T-cell matures.

Two types of T-cells are helper T-cells and killer T-cells. Compare with B-cell in the humoral immune system.

- T-Cell Receptor: The part of the surface of a T-cell which matches a presented antigen. For T-cell receptors, the corresponding antigen is a partially consumed antigen which has been presented by a phagocyte. Compare with B-cell receptor.
- T-Lymphocyte: T-cell.
- Virgin T-Cell: A mature T-cell that has not yet met antigen.

APPENDIX D, KNOWLEDGE ENGINEER DECISIONS

Below is a summary of decisions that must be made by a knowledge engineer in the system described in this research:

- Deciding the research type goal.
- The design of the hybridized system.
- The initial data selection.
- How to normalize the input data.
- The SOM size.
- The original assignment of node vectors for the SOM.
- What constitutes height in the 3D scheme for the SOM.
- How danger and safe signals are produced by the SOM.
- The color scheme for the SOM.
- The atomicity value for SJ Fusion.
- How often to start a new artificial dendritic cell.

- What the base weights and threshold are for the Interim DCA Formula.

VITA

Graduate School
Southern Illinois University

Chester L. (“Chet”) Langin

P.O. Box 1262, Carbondale, Illinois 62901

clangin@siu.edu

Southern Illinois University Carbondale
Bachelor of Science, Journalism, June 1974

Southern Illinois University Carbondale
Master of Science in Computer Science, August 2003

Special Honors and Awards:

Professional Certifications:
GIAC Security Essentials Certification (GSEC) Gold
GIAC Certified Intrusion Analyst (GCIA) Gold
GIAC Reverse Engineering Malware (GREM)
GIAC Payment Card Industry (GPCI)

Was a staff adviser for the SIU winning team at regional Cyber defense competition at University of Illinois at Urbana, March, 2006, and at national playoffs in San Antonio, April, 2006.

On winning SIU team at ACM programming contest at University of Illinois in Urbana, October, 2002.

Awarded a Sears Congressional Internship in Washington, D.C., for Congresswoman Marjorie Holt, Spring, 1974.

Awarded a Newspaper Fund copyediting internship at Cincinnati Enquirer, Summer, 1973.

Grant Proposal Submitted:
“A Versatile SOM+ Platform for Intrusion Detection,” submitted to US National Science Foundation (NSF), December, 2010, co-PI (pending).

Dissertation Title:

A SOM+ Diagnostic System for Network Intrusion Detection

Major Professor: Shahram Rahimi

Publications:

Book:

"An Easy Course in Using DOS," Chester Langin, illustrated by Virginia Rohrbacher, Grapevine Press, 1990, 1992.

Peer Reviewed Journals Articles:

"ANNaBell Island: A 3D Color Hexagonal SOM for Visual Intrusion Detection," Chet Langin, Michael Wainer, and Shahram Rahimi, International Journal of Computer Science and Information Security, Vol. 9, No. 1, pages 1-7, January, 2011.

"SOM with Vulture Fest Model Discovers Feral Malware and Visually Profiles the Security of Subnets," Chet Langin, Dunren Che, Michael Wainer, and Shahram Rahimi, International Journal of Computers and Their Applications, Vol. 17, No. 4, pages 241-249, December, 2010.

"Soft Computing in Intrusion Detection: The State Of The Art," Chet Langin, and Shahram Rahimi, Soft Computing in Intrusion Detection: The State Of The Art. Journal of Ambient Intelligence & Humanized Computing, Vol. 1, No. 2, pages 133-145, June, 2010.

"Protein Fingerprinting: A Domain-Free Approach to Protein Analysis," Jeffry L. Shultz, Chet Langin, Dennis G. Watson and David Lightfoot, 2004, Journal of Genome Science and Technology. 3:41-47.

Peer Reviewed Conference Articles:

"Visualization of Network Security Traffic using Hexagonal Self-Organizing Maps," Chet Langin, Dunren Che, Michael Wainer, and Shahram Rahimi, The International Conference on Computer Applications in Industry and Engineering, San Francisco, International Society for Computers and their Applications (ISCA), November 4-6, 2009.

"A Self-Organizing Map and its Modeling for Discovering Malignant Network Traffic," Chet Langin, Hongbo Zhou, Bidyut Gupta, Shahram Rahimi, and Mohammed R. Sayeh, 2009 IEEE Symposium on Computational Intelligence in Cyber Security, Nashville, TN, USA, March 30-April 2.

"A Model to Use Denied Internet Traffic to Indirectly Discover Internal Network Security Problems," Chet Langin, Hongbo Zhou and Shahram Rahimi, The First IEEE International Workshop on Information and Data Assurance, Austin, Texas, USA, 2008.

White Papers:

"Capturing and Analyzing Packets with Perl," advisor for author John Brozycki, SANS, 2010.

"Smart IDS - Hybrid LaBrea Tarpit," advisor for author Cristian Ruvalcaba, SANS, 2009.

"A System of Persistent Baseline Automated Vulnerability Scanning and Response in a Distributed University Environment," Chet Langin, SANS, 2007.

"Documentation is to Incident Response as an Air Tank is to Scuba Diving," Chet Langin, SANS, 2007.

Book Section:

"Tear Down the Wall," (Game), featured in Phenomenal PC Games, Bob LeVitus and Ed Tittle, Prima Publishing, 1992.

Book Review:

Scalable Computing: Practice and Experience (8:4), 2007.

Reviewer:

Neural Computing and Applications, 2010.
SANS Adviser for Gold professional certification white papers, 2009-present.
The 4th International Symposium on Bio- and Medical Informatics and Cybernetics (BMIC 2010).
The 5th International Symposium on Bio- and Medical Informatics and Cybernetics (BMIC 2011).
The 7th International Symposium on Risk Management and Cyber-Informatics (RMCI 2010).
Educause Computer Security Awareness Poster & Video Contest 2009.

Presentations:

"Uses Of The Soybean Sequence Ready Physical Map," Jeffry Shultz, N Lavu, Chet Langin, Kay Shopinski, S Kazi, R Bashir, J Iqbal, J Afzal, C Town, K Meksem, H Zhang, C Wu, David Lightfoot, Presented at Plant & Animal Genome XIII Conference, San Diego, CA, 2005.

Posters:

"An EST And Bes Based Physical Map Of Genes Distribution In The Soybean Genome," (Poster Abstract), K Shopinski, MJ Iqbal, J Shultz, Chet Langin, N Lavu, DA Lightfoot, Presented at Plant & Animal Genome XIII Conference, San Diego, CA, 2005.

"Development Of Three Minimum Tile Paths For Soybean Functional Genomics," (Poster Abstract), J Shultz, J Potter, K Wakefield, Chet Langin, MJ Iqbal, DA Lightfoot, Presented at Plant & Animal Genome XIII Conference, San Diego, CA, 2005.

"Proteomic And Genomic Approaches To Molecular Breeding Of Resistance To Soybean Sudden Death Syndrome And Cyst Nematode In Elite Cultivars," (Poster Abstract), J Afzal, R Ahsan, S Kazi, C Langin, MJ Iqbal, DA Lightfoot, Presented at Plant & Animal Genome XIII Conference, San Diego, CA, 2005.

"Mapping Relationships Among Soybean Genomic Features," (Poster Abstract), C Langin, J Shultz, N Lavu, D Wainer, J Iqbal, DA Lightfoot, Presented at Plant & Animal Genome XIII Conference, San Diego, CA, 2005.

"The Soybean Gbrowse Database Maps Relationships Between Soybean Genomic Features," (Poster Abstract), C Langin, J Shultz, N Lavu, D Wainer, J Iqbal, DA Lightfoot, Presented at Plant & Animal Genome XIII Conference, San Diego, CA, 2005.

"Cataloging Homeologous Genomic Regions Of The Duplicated Soybean Genome By Physical Map Analysis," (Poster Abstract), J Shultz, S Yaegashi, K Zobrist, Chet Langin, MJ Iqbal, DA Lightfoot, Presented at Plant & Animal Genome XIII Conference, San Diego, CA, 2005.

"Genome Evolution-Framework Genome Comparisons Among Medicago truncatula, Arabidopsis thaliana and Glycine max Shows Separate Paleopolyploid Genome Structures has Provided Aneuplorosis in Disease Resistance and Other Trait Loci," (Poster Abstract), Jeffery Schultz, Chester Langin, Nagajothi Lavu, M Javed Iqbal, Kay Shopinski, Jawad Afzal, Samreen Kazi, Rabia Bashir, Chenchag Wu, Chris Town, Hongbin Zhang and David Lightfoot, Model legume Congress, PacificGrove, California, June5-9, 2005, pg:153.

"Development and mapping of 1053 new SSR markers from the Soybean Sequence Ready Physical Map," (Poster), Samreen Kazi, Rabia Bashir, Jeffry Shultz, Chester Langin, Javed Iqbal and David A. Lightfoot, Plant and Animal Genome conference. January 15-19, 2005 Town and Country convention center San Diego, CA, 2005.

Browsing the soybean genome: Educational challenges from physical map builds of a recently duplicated genome," (Poster), Shultz, Jeff , Meksem, Khalid, Langin, Chet, Kazi, Samreen, Zobrist, Kimberly, Yaegashi, Satsuki, Lavu, Nagajyothi, Iqbal, Javed, Potter, Jamie, Yesudas, Charles, Wainer, David, Watson, Dennis, Wu, Chencang,

Zhang, Hong Bin, Town, Christopher, Lightfoot, David, American Society of Plant Biologists (ASPB), Lake Buena Vista, FL, July 24 - July 28, 2004.

Numerous newspaper articles on non-computer-related topics.