Autonomic Intelligent Network Sensor Model for Protection of Critical Infrastructure Systems

A Dissertation

Presented in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Denis Todd Vollmer

April 2014

Major Professor: Milos Manic, Ph.D.

Authorization to Submit Dissertation

This dissertation of Denis Todd Vollmer, submitted for the degree of Doctor of Philosophy with a Major in Computer Science and titled "Autonomic Intelligent Network Sensor Model for Protection of Critical Infrastructure Systems," has been reviewed in final form. Permission, as indicated by the signatures and dates below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor:	______________________________ Milos Manic, Ph.D.	Date: ______________
Committee Members:	______________________________ James Alves-Foss, Ph.D.	Date: ______________
	______________________________ Terence Soule, Ph.D.	Date: ______________
	______________________________ Craig Reiger, Ph.D.	Date: ______________
Computer Science Administrator	______________________________ Gregory Donohoe, Ph.D.	Date: ______________
Engineering College Dean	______________________________ Larry Stauffer, Ph.D.	Date: ______________

Final Approval and Acceptance

Dean of the College Of Graduate Studies:	______________________________ Jie Chen, Ph.D.	Date: ______________

Abstract

This dissertation examines the concepts and implementation of a network based autonomic cyber sensor framework. The research provides an answer to the need to protect Ethernet connected control systems, such as those found in critical infrastructures, from cyber assaults. A layered architecture, which utilizes computational intelligence techniques for learning and a multi-level communication scheme, is described. Genetic Algorithms, Neural Networks, Fuzzy logic, Clustering, passive network scanning and dynamic virtual honeypots are all integral methods of the presented work. The application of computational intelligence techniques provides heuristics for specific problems such as anomaly detection and rule creation. The framework integrates several of these techniques into a broader overall solution while shielding the complexity from the user.

Contributions of this dissertation include introduction of a multi-level architecture with a two-layer information communication scheme. This scheme segregates modifications of components from changing standards and centralizes the complexity of external messaging to a single component reducing implementation costs and the security exposure of the sensor. A process of automatic creation and dynamic updates to emulated network hosts is described. This process provides an independent view of attached devices without interfering with an operational network. Additionally, a network anomaly recognition system based on data clustering and advanced fuzzy logic is presented. While traditional approaches improve false positives at the expense of false negatives, or vice versa, this approach enables improvement of both accuracy measurements simultaneously.

Two related algorithms for communication of network state awareness are detailed. They bridge the semantic gap between identifying a binary anomaly value to communicating what it means to a human. The use of intrusion detection rules as a knowledge base for learning systems such as neural networks is introduced. This leverages the large set of existing knowledge represented by the static rules sets and makes the information available for anomaly behavior systems. Finally, the automatic creation of intrusion detection rules based upon network traffic identified by anomaly behavior systems is shown resulting in a reduction of human effort needed to create rules.

Acknowledgements

I am thankful to several people and entities for supporting the creation of this dissertation. First, my major professor Dr. Milos Manic and committee members Dr. Jim Alves-Foss, Dr. Terry Soule and Dr. Craig Rieger guided me through the process. The Idaho National Laboratory provided me the opportunity to work on areas of mutual benefit. Finally, I am grateful to Nancy, Sarah, Hannah and the rest of my family for the support provided over the many years to reach this milestone.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

The proliferation of digital devices in a networked industrial ecosystem, with an exponential growth in complexity and scope, has resulted in elevated security concerns and management complexity issues. The expanding reach and wide deployment of digital systems into societies fundamental physical infrastructures pose both a complexity and security concern (Gungor & Hancke, 2009), (Cheminod, Durante, & Valenzano, 2013). The complexity of modern industrial networks primarily stems from three areas: presence of heterogeneous hardware and software, dynamic network composition and usage patterns, and decentralization of control (Gungor, Lu, & Hancke, 2010). Dealing with these complexities requires a solution that is resilient, flexible and manageable.

There is an increased awareness by the US Government and many other entities of the threat posed by cyber attacks (Shea, 2004). For instance, on February 28, 2013 the director of the FBI stated, “That network intrusions pose urgent threats to our national security and to our economy.” President Obama has stated that the, “cyber threat is one of the most serious economic and national security challenges we face as a nation” and that “America's economic prosperity in the 21st century will depend on cyber security.” Computer based systems, used within many critical infrastructures to monitor control systems, are increasingly being connected directly or indirectly to the Internet. Additionally, these control systems are composed of interconnected computational devices on local Ethernet networks. This elevates the threat status of these systems, as they can be even more vulnerable than traditional Information Technology (IT) systems (Duggan, Berg, Dillinger, & Stamp, 2005). Attacks against these systems can endanger public safety and lead to large expenditures of capital.

In the particular case of smart grid networks a large-scale deployment of 46 million devices has recently occurred (The Edison Foundation, 2013). These systems add Wireless Access Point (WAP) devices to existing utility networks. For instance, in a typical Advanced Metering Infrastructure (AMI) system 1,500 wireless sensors report to one or multiple WAP nodes (Iwao, et al., 2010). An example deployment is the Pacific Northwest Smart Grid Demonstration Project. As of August 2013, almost 69 million of these meters were planned for deployment in the United States (The Edison Foundation, 2013). Assuming a uniform deployment of sensors this calls for 46,000 WAP's without any regard for redundancy. This large influx of devices, either wired or wireless, into a network vastly expands the potential network attack surface.

Generally, critical control systems utilize large amounts of distributed data gathered from physical devices. Despite the large actual and planned deployment of wireless devices, a large core of wired connectivity is still prevalent. For reliability, security and financial reasons wireless networks eventually attach to a physical wired connection. The wired network connection points are a logical area of security concern that is addressed by this dissertation. Computational intelligence techniques and automatic control algorithms are key to the successful implementation of an Autonomic solution designed to help protect control system networks.

Fundamentally Autonomic computing is a systemic view of computing, modeled after self-regulating biological systems. The goal of autonomic computing, introduced in 2001, is to develop computational systems that are capable of self-management. The four foundational requirements of self-managed computerized systems are: self-configuration, self-protection, self-healing, and self-optimization (IBM, 2006).

Autonomic computing research is a response to the realization that traditional software systems are facing a decreasing benefit from technological advances because the complexities of management and development are overwhelming. The concept of implementing technology designed specifically to continuously manage and optimize the functionality of other technologies is an extension of the notions present in control theory. Control theory provides descriptions of closed systems whose components and desired properties are well known and described by linear or nonlinear models. However, when the system is constantly changing and exhibits varying or uncertain information performance may be poor (Dobson, et al., 2006). Autonomic design enables adaptive and flexible functionality. The implementation of this design reacts to evolving states and proactively looks forward to future demands.

1.1 Objective

This dissertation answers the need to protect Ethernet connected control systems, such as those found in critical infrastructures, from cyber assaults. This work shows that: Control system network security systems can employ standardized communication mechanisms and computational techniques in an autonomic computing structure to detect and mitigate operational disruption caused by malicious cyber assaults or other abnormal behaviors. Moreover, with the addition of computational intelligence algorithms, the resulting autonomic system simplifies human operator interaction with the cyber security mechanisms. This simplification stems from automatic configuration of honey pots, autonomous creation of detection rules and classification of anomalous network behavior into well-known categories.

1.2 Dissertation Contributions and Organization

The work presented in this dissertation examines the concepts and implementation of an autonomic cyber sensor framework and supporting algorithms. The goal of the sensor design is to provide state awareness to human operators and automated response systems utilizing integrated communications and computational intelligence techniques. The primary focus for validation is on a heterogeneous control system network but the presented concepts are flexible enough to be applied in other environments. The structure of this dissertation is as follows:

- **Chapter 2** presents a background and literature review. First, concepts of Autonomic computing are presented. Second, Service Oriented Architectures and both internal and external messaging are discussed. Third, anomaly detection, virtual hosts and network host monitoring methods are presented. Finally a literature review of related projects is given.
- **Chapter 3** describes a novel architecture utilizing concepts of Autonomic computing and a SOAP based IF-MAP external communication layer to create a network security sensor. Three complementary modules that utilize a standard internal data transport layer to dynamically reconfigure in response to a changing environment are presented. The Autonomic Intelligent Cyber Sensor (AICS) approach simplifies integration of legacy software and supports a secure, scalable, self-managed framework. The four primary contributions of this chapter are:
  1) A flexible two level communication layer based on Autonomic computing and Service Oriented Architecture is detailed.
  2) An Intelligent Anomaly Assessment (IAA) module that utilizes clustering and fuzzy logic to monitor traffic for abnormal behavior is introduced.

3) The Dynamic HoneyPot (DHP) section describes the collaborative use of dynamic virtual honeypots in a control system.
4) The Network Entity Identification (NEI) module description identities aspects of effective tools for extracting network host characteristics.

The presented AICS framework focuses on automatically managing the complexity of self-configurable dynamic virtual hosts by adapting to an operational network environment. A self-updating model, based on passive monitoring of the network devices, is created and maintained. This model is used to configure deceptive network entities designed to draw the focus of malicious intent. Finally, a test scenario is examined to show the communication and autonomic aspects of the system.

Supporting Publications:

T. Vollmer and M. Manic. "Autonomic intelligent cyber sensor to support control network situational awareness," *IEEE Transactions on Industrial Informatics),* 26 June 2013.

T. Vollmer and M. Manic. "Cyber-Physical system security with deceptive virtual hosts for industrial control networks," *IEEE Transactions on Industrial Informatics,* 27 February 2014.

O. Linda, T. Vollmer, and M. Manic. "Improving cyber-security of smart grid systems via anomaly detection and linguistic domain knowledge," *Fifth International Symposium on Resilient Control Systems*, August 2012, Salt Lake City, Utah, USA.

O. Linda, T. Vollmer, J. Wright, and M. Manic. "Fuzzy logic-based anomaly detection for embedded network security cyber sensor," *IEEE 2011 Symposium on Computational Intelligence in Cyber Security (CICS 2011)*, 11–15 April, Paris, France.

O. Linda, T. Vollmer and M. Manic. "Neural network based intrusion detection system for critical infrastructures," *IEEE 2009 International Joint Conference on Neural Networks*, 14–19 June 2009, Atlanta, Georgia, USA.

G. Rueff, B. Wheeler, T. Vollmer, T. McJunkin, R. Erbes. "INL Control System Situational Awareness Technology Annual Report 2012", *Idaho National Laboratory Technical Report,* 01 October 2013.

- **Chapter 4** introduces two communication algorithms, HISA and CeNISA, for network security awareness. The algorithms in this chapter enhance the communication of anomaly behavior detection systems such as the IAA component described in Chapter 3. The two contributions of this chapter are:

1) First the Human Interface for Security Awareness (HISA) algorithm which interprets cyber incident information for human operators from anomaly based intrusion detections systems is presented. HISA utilizes a similarity algorithm mapping anomaly behavior results to signature based intrusion system rules.

2) Next the Computationally Efficient Neural Network Intrusion Security Awareness (CeNISA) algorithm is discussed. This enhanced version of HISA improves upon the computational time required to produce answers. CeNISA leverages rule knowledge sets to produce classifications for anomaly based systems. A unique aspect is the training of an error back-propagation neural network with intrusion detection rule features to provide a recognition basis. Ethernet network packet details are subsequently provided to the trained network to produce a classification.

Supporting Publications:

T. Vollmer, and M. Manic. "Computationally efficient neural network intrusion security awareness," *Second International Symposium on Resilient Control Systems*, 11–13 August 2009, Idaho Falls, Idaho, USA.

T. Vollmer, and M. Manic. "Human interface for cyber security anomaly detection systems," *Second Conference on Human System Interactions*, 21-23 May 2009, Catania, Italy.

- **Chapter 5** explores a solution to autonomously create static IDS rule sets utilizing evolutionary computation techniques. This is accomplished by implementing a Genetic Algorithm (GA) to create static rule candidates from previously identified anomalous network packets. The contributions of this chapter are:

1) Distance measures as a means of determining similarity or closeness are a common algorithmic feature for many GA implementations. Given the importance of similarity measures to the solution of multi-modal GA, a substantial part of this chapter is devoted to exploring Mahalanobis, Euclidean and Distance Metric Learning metrics.

2) Translating a newly discovered intrusion recognition criteria into a distributable rule can be a human intensive effort. In order to minimize this effort, part of this chapter explores a multi-modal genetic algorithm solution for autonomous rule creation. The algorithm focuses on the process of automatically creating rules once an intrusion has been identified. The system described could be considered as a one-way communication mechanism enabling rule based intrusion systems to benefit from information derived from behavior modeling.

Supporting Publications:

T. Vollmer, J. Alves-Foss, and M. Manic. “Autonomous rule creation for intrusion detection,” *IEEE 2011 Symposium on Computational Intelligence in Cyber Security (CICS 2011),* 11–15 April 2011, Paris, France.

T. Vollmer, T. Soule, and M. Manic. “A distance measure comparison to improve crowding in multimodal problems,” *Third International Symposium on Resilient Control Systems,* 10–12 August 2010, Idaho Falls, Idaho, USA.

- **Chapter 6** Conclusion and Future Work

Chapter 2. Background and Literature Review

This chapter provides background context and relevant past research efforts. The technologies described are: Autonomic Computing, Communication and Computational Intelligence. These technologies are the central approaches used in this dissertation.

This section explains fundamental concepts for Autonomic research, Service Oriented Architecture, IF-MAP and D-Bus. In a large digital ecosystem, constituent members have to be able to continuously adapt to unforeseen circumstances and evolve in an autonomic way without human intervention. Devices are expected to cooperate to accomplish a mission. They become part of a larger, more complex infrastructure where complementary single elements are exploited to achieve an emergent complex behavior.

2.1 Autonomic Research

It is the Autonomous Nervous System (ANS), present in human anatomy, which serves as inspiration for computer based autonomic system architectures. The ANS is that portion of the nervous system concerned with regulation of activity of visceral functions generally not controlled by conscious thought. However some actions, such as breathing, work in tandem with active control. ANS is typically divided into sympathetic and parasympathetic divisions. These divisions function in opposition to each other in a complementary manner. The sympathetic division typically functions with actions requiring quick responses. The parasympathetic division functions with actions that do not require immediate reaction. Together the systems can work in concert to achieve homeostasis of the relevant activity (Brodal, 1998).

Mimicking the ANS, digital autonomic system architectures consist of a dynamic collection of autonomic elements each performing a constrained function (IBM, 2006). This architecture is generally composed of an autonomic manager (AM) that controls one or more managed resource elements (RE). The resources themselves may be legacy components or exhibit autonomic features.

A manageability interface composed of sensor and effector facilitates the communication between AM and RE and is implemented internally with D-Bus for this work. Sensors obtain data from the resources and effectors are used to perform operations on the resource. In addition, this sensor/effector layer is conceptually replicated to

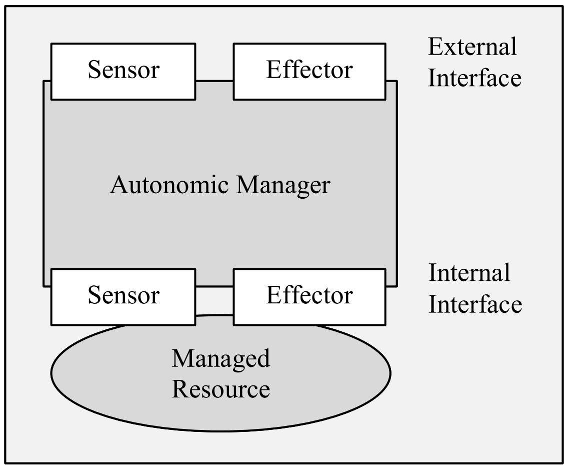


Figure 2-1. Autonomic Element Architecture

provide access to outside entities (Cheng, Leon-Garcia, & Foster, 2008). IF-MAP, a standards based messaging system, is proposed here as a solution for external communication. Fig. 2-1 presents the described autonomic element architecture with the two interface layers labeled as inner interface and outer interface. These interface concepts are a critical concept expanded upon later in this dissertation.

An Autonomic Digital Ecosystem (ADE) is a model for production systems that builds on the notion of autonomic self-management by embedding exploitable control features within modules (Ulieru & Grobbelaar, 2007). Cyber physical ecosystems (CPE) include interconnected subsystems that sense and act upon the physical world to form a complex system of systems. These CPE’s are a foundational layer of a wider more encompassing digital ecosystem. CPEs typically bridge the cyber world of computing and communications with the physical world by embedding wireless sensor nodes into the physical world (Yang, Xu, Li, & Chen, 2011).

Finally, the description of an autonomic system is incomplete without mention of the self-* properties. A key feature of autonomic systems is the automated management of resources exhibiting self-configuration, self-optimization, self-healing and self-protection characteristics (IBM, 2006). The focus of this dissertation is primarily on the self-configuration and self-protection features. In order to harness these properties into a cohesive whole, a control loop monitors, analyzes, plans and executes as appropriate.

2.2 System Messaging

A central concept from the autonomic nervous system is a standard communication channel that can carry information to processing entities. One important property of biological communications is the capability to communicate a multitude of information to a dissimilar number of recipients. This is accomplished using standardized messaging pathways that externally appear to be the same but internally contain specialized content. For example, information can occur over a distributed network of nodes such as synapses. A similar functionality can be found in electronic data networks. A flexible, dynamic description language is required to fulfill this function. It has been shown that XML is an appropriate choice of implementation (Shuaib, Anthony, & Pelc, 2010).

Application messaging in a distributed environment requires careful consideration for implementation. Messaging and component interaction can be handled by utilizing concepts from service-oriented computing (SOC). In SOC components are modeled as loosely coupled services that communicate using standard data formats and interfaces. A service-oriented architecture (SOA) can satisfy key elements of a SOC. Web services typically serve as a basis for a SOA. These services consist of a service provider, service consumers and a service registry. SOA is an enabling technology for development of large systems in industry. It is a flexible solution that can decrease deployment and maintenance costs. (Candido, Colombo, Barata, & Jammes, 2011).

The Simple Object Access Protocol (SOAP) is typically used as a messaging framework layer of a web based SOA. Version 1.2 of SOAP is a specification utilizing XML for exchanging structured and typed information in a distributed environment. Typically Hypertext Transfer Protocol (HTTP) or SMTP is used for message handling. It provides an extensible framework for transferring application specific information without specifically detailing the semantics of the data carried. The framework provides required actions to perform on a SOAP message in order to send and receive it.

One SOA based implementation messaging standard is the Interface to Metadata Access Points (IF-MAP) version 2.0. IF-MAP is an open protocol standard published by the Trusted Computing Group for delivering network related information in a portable and secure framework. It utilizes a publish, subscribe, search messaging paradigm implemented with SOAP. Originally made available in April 2008, version 2.1 rev. 15 was published May 7, 2012. The design goal of the IF-MAP working group was to enable the sharing of security centric data between network devices and systems (Trusted Network Computing Group, 2011).

IF-MAP has multiple defined parts: a base protocol and metadata for a specific usage. Currently the only metadata defined is for network security (Trusted Network Computing Group, 2010). However, a draft version for Industrial Control systems has been released. One of the benefits provided by the released metadata definition is a common reference standard of network security data. In addition to the currently defined metadata, it is possible to enhance the metadata specification if the base protocol is followed. This provides flexibility to adapt to a given implementation.

2.2.1 Base Protocol and Metadata

The base protocol specifies actions for clients. IF-MAP clients have access to three actions for metadata: publish, search and subscribe. Clients store or publish metadata into a MAP (Metadata Access Point) for others to consume. A search allows clients to obtain published metadata from the MAP based on a flexible criterion. Subscribe requests asynchronous results from a predefined search whenever a client publishes new metadata. In this way clients can monitor for specific relevant events and be alerted when one occurs. All IF-MAP operations and data types are expressed utilizing XML. This helps ensure a consistent format of the data when it is exchanged between multiple diverse parties.

The protocol also defines two species of data: metadata and identifiers. Metadata in this context is any shared data about network devices, policies, status, behavior or observed relationships. The five defined identifiers are identity, IP Address (v4 and v6), MAC address, Access Request and Device. Identifiers are used to refer to specific metadata items. The metadata of interest to this dissertation are as follows.

ip-mac: A binding between an IP address and a mac address valid for a time frame.

device-characteristic: An inherent characteristic of an entity such as its manufacturer or OS.

device-ip: The IP address of an associated device.

discovered-by: A link identifying which sensor device has discovered a new IP or MAC address.

event: Activity of interest on a network such as a virus attack or abnormal behavior.

2.3 Host Messaging

D-Bus is a system created for inter-process communication (IPC) primarily used in Linux desktop windowing systems such as KDE and GNOME. Additionally, it has been utilized in several other projects including reconfigurable mobile network devices (Merentitis, Patouni, Alonistioti, & Doubrava, 2008). It was designed to avoid round trips and allow for asynchronous operation. Conceptually it works in the context of messages, and handles many of the difficult IPC synchronization issues. Several high-level language bindings are available which provides a flexible implementation solution. The following describes the D-Bus system and the terminology commonly used with it.

A *bus* is a virtual path that messages are passed over. Multiple instances of a bus may exist and are managed by a software daemon. Several Linux based operating systems offer two busses, system and session, for use without requiring user configuration. Fine-grained user access to buses and actions on the appropriate bus service is available via configuration files. Applications that utilize a bus take on the role of either a server or client. Server processes listen for incoming method requests from clients.

A *method* is made available by publishing it on a message bus. Methods are remotely invoked operations of an object. As is found in object-oriented programming languages, objects contain methods. The input and output parameters need to be defined in advance of publication. A method is registered with the D-Bus daemon under an *interface* and available through an *object path*. Interfaces are a collective group of methods that help define the type of an object.

A service is a named collection of methods with an associated .service file. The service file defines an executable program to handle receipt of a message related to a given method. If the configured executable is not currently running the message is queued and the process is started. When the process is ready the queued message is delivered. Fig. 2-2 provides a representation of the D-Bus process given two applications talking on a single host.

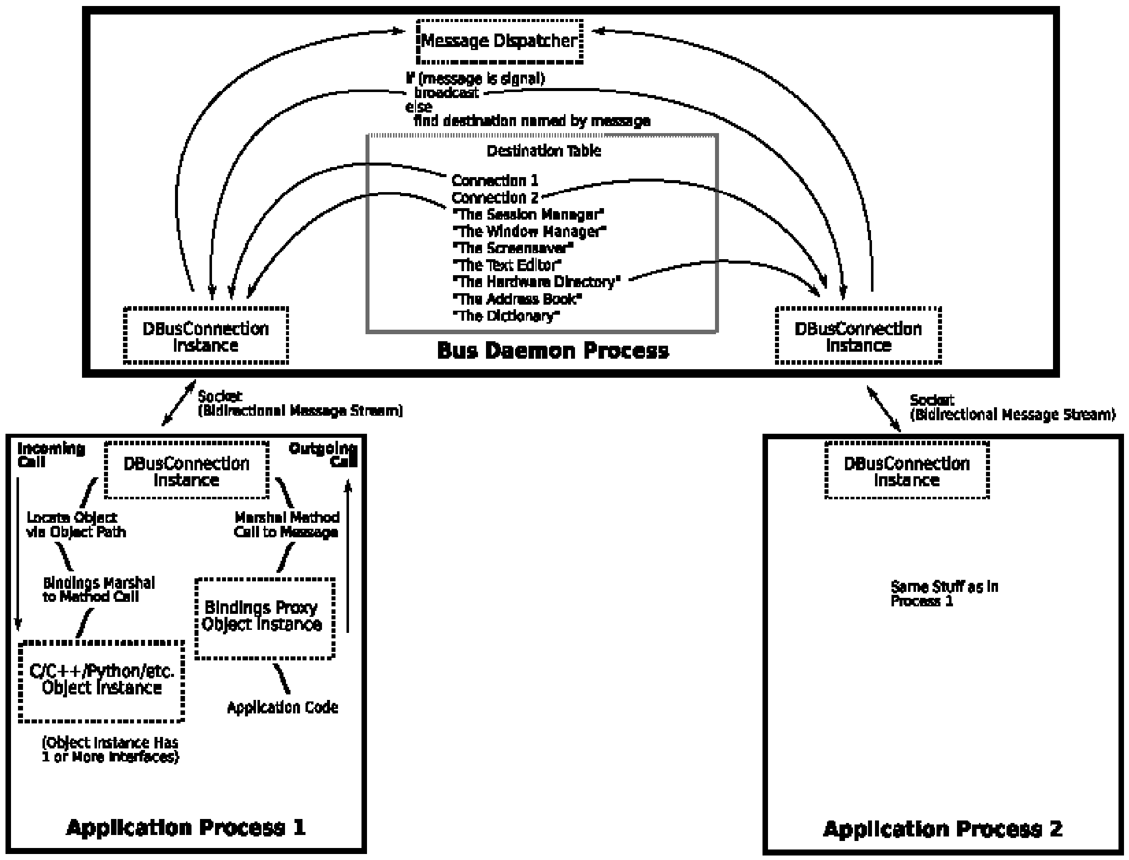


Figure 2-2. D-Bus Process

2.4 Machine Learning or Computational Intelligence

There does not appear to be a fundamental difference in the definitions of Machine Learning and Computational Intelligence (CI). It may be that one defining difference is Machine Learning's focus on large-scale data analysis (Barber, 2012). For the purposes of consistency in this dissertation, CI will be used with the understanding that an overlap in methodologies and approaches is present. Fundamentally, CI is concerned with adaptive mechanisms that enable intelligent behavior in complex environments. Frequently algorithms in this domain are derived from biological or human inspired models. These algorithms provide automated methods to detect patterns in data. Furthermore, these patterns may be used to predict future data or perform other analytical tasks. These capabilities provide a logical basis for inclusion in an Autonomic framework.

In general there are two types of learning in CI: *supervised* and *unsupervised*. In supervised learning both inputs and labeled outputs are provided. The goal is to derive a mapping from the inputs, otherwise known as features, to the correct output value. This is called pattern recognition when the output is categorical value. It is typically used to predict a probable future value based on previously unseen inputs. In unsupervised learning input features are presented but labeled outputs to gauge correctness are not. The aim is to derive a compact description of patterns found in the input. There is a third type of learning consisting of aspects of the two called semi-supervised learning.

CI techniques have been extensively applied to the problem of network intrusion detection (Sommer & Paxson, 2010). An additional beneficial feature of CI is the ability to learn from multi-dimensional, non-linear data (Witten & Frank, 2005)] Techniques such as artificial neural networks (Linda, Vollmer, & Manic, 2009), support vector machines (Hu, Liao, & Vemuri, 2003), genetic algorithms (Stein, Chen, Wu, & Hua, 2005), fuzzy logic (Gomez, Dasgupta, Gomez, & Kaniganti, 2003) and unsupervised clustering (Zhong, Khoshgoftaar, & Seliya, 2007) are powerful learning tools for modeling the network behavior.

2.4.1 Fuzzy Logic Systems

Zadeh originally proposed Fuzzy Logic (FL) as a tool for dealing with linguistic uncertainty and vagueness often found in the imprecise meaning of words (Zadeh L. A., 1965). Fuzzy Logic is based on the theory of fuzzy sets that utilizes a set membership function in the range of [0.0, 1.0]. A 0.0 indicates an element is not a member and 1.0 strongly indicates the opposite. FL describes relative concepts and eliminates the use of crisp arbitrary thresholds for continuous variables. Rule based systems can take advantage of these fuzzy thresholds to deal with uncertainty or variability and possibly provide more accurate classifications.

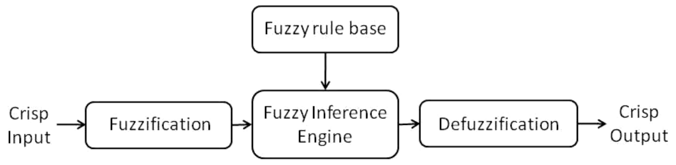


Figure 2-3. Fuzzy Logic System

A Fuzzy Logic System (FLS) is composed of four primary parts – input fuzzification, fuzzy inference engine, fuzzy rule base and output defuzzification, as depicted in Fig. 2-3. The Mamdani FLS implemented in this work maintains a fuzzy rule base populated with fuzzy linguistic rules in an implicative form (Klir & Yuan, 1995). Consider the rule R_k that is described as follows:

$$\text{Rule } R_k\text{: } \textbf{IF } x_1 \text{ is } A_1^k \textbf{ AND } \ldots \textbf{ AND } x_n \text{ is } A_n^k \textbf{ THEN } y_k \text{ is } B^k \qquad \textbf{(2-1)}$$

Here, symbol A_i^k and B^k denote the i^{th} input fuzzy set and the output fuzzy set of the k^{th} rule, respectively, n is the dimensionality of the input vector $\vec{x}$ and y_k is the associated output variable. Each element of the input vector $\vec{x}$ is first fuzzified using the respective fuzzy membership function (e.g. Gaussian, triangular, trapezoidal, etc.). The fuzzification of input value x_i into fuzzy set A_i yields a fuzzy membership grade $\mu_{A_i^k}(x_i)$. Using the minimum t-norm the degree of firing of rule R_k can be calculated as:

$$\mu_{R_k}(\vec{x}) = \min\{\mu_{A_i^k}(x_i)\}, i = 1 \ldots n \qquad \textbf{(2-3)}$$

After applying the rule firing strength via the t-norm operator to each rule consequent, the output fuzzy sets are aggregated using the t-conorm operator (e.g. the maximum operator) resulting in an output fuzzy set B. For detailed description of the fuzzy inference process refer to (Mendel, 2001).

In order to obtain the crisp output value, one of the available defuzzification techniques is applied. Upon discretizing the output domain into N samples, for example the centroid defuzzifier can be applied:

$$y = \frac{\sum_{i=1}^{N} y_i \mu_B(y_i)}{\sum_{i=1}^{N} \mu_B(y_i)} \quad (2\text{-}4)$$

2.4.2 Nearest Neighbor Clustering

Grouping data into a set of categories based on properties of a common criterion is called clustering (Xu & Wunsch, Clustering, 2009). The Nearest Neighbor Clustering (NNC) algorithm is an unsupervised clustering technique (Witten & Frank, 2005). As opposed to other commonly used clustering algorithms (e.g. K-Means, Fuzzy C-Means, Self Organizing Maps, etc.), the NNC algorithm does not require specification of the expected number of clusters a priori. Instead, the clustering process is controlled by an established maximum cluster radius parameter. The smaller the radius the more cluster will be generated and vice versa.

Assume an input dataset X composed of N input patterns denoted as:

$$X = \{\vec{x}_1,\ldots,\vec{x}_N\}, \vec{x}_i \in \Re^n \quad (2\text{-}5)$$

Here, n denotes the dimensionality of the input domain. Vector $\vec{x}_i$ can be expressed as $\vec{x}_i = \{x_i^1,\ldots,x_i^n\}$.

Each cluster constitutes a prototype of resembling instances, subject to a specific similarity measure. The Euclidean distance similarity measure is considered in this work. Each cluster P_i is described by its Center Of Gravity (COG) $\vec{c}_i$ and its associated weight w_i. The weight w_i stores the number of patterns previously assigned to cluster P_i. Following this notation, cluster P_i can be expressed as:

$$P_i = \{\vec{c}_i, w_i\}, \vec{c}_i \in \Re^n, w_i \in \aleph^+ \quad (2\text{-}6)$$

The learning process of the NNC algorithm begins by creating an initial cluster P_1 at the location of the first input pattern $\vec{x}_1$. Next, input patterns from dataset X are selected in a sequential manner. The nearest prototype from the set of available clusters is determined for each instance. For an input pattern $\vec{x}_i$, the nearest cluster P_a is determined using the Euclidean distance norm:

$$dist(\vec{c}_a, \vec{x}_i) = \min_j \sqrt{(c_j^1 - x_i^1)^2 + \ldots + (c_j^n - x_i^n)^2}, j = 1 \ldots C \quad (2\text{-}7)$$

Here, C denotes the number of currently acquired clusters. Using the maximum cluster radius parameter - *rad*, the input pattern $\vec{x}_i$ is assigned to cluster P_a if the following condition holds: $dist(\vec{c}_a, \vec{x}_i) \leq rad$. In this case, the parameters of cluster P_a are updated as:

$$\vec{c}_a = \frac{w_a \vec{c}_a + \vec{x}_i}{w_a + 1}, w_a = w_a + 1 \quad (2\text{-}8)$$

If $dist(\vec{c}_a, \vec{x}_i) > rad$, a new cluster is created at the location of input pattern $\vec{x}_i$, and its weight is set to 1.

2.4.3 Error Back Propagation Neural Networks

Error Back Propagation (EBP) networks are a well-researched supervised learning method introduced by Werbos in 1974. The power of a multilayer neural network lies in its ability to model multidimensional nonlinear problems. The learning vectors are presented as input and the calculations from each layer are fed forward to the next layer in the network. Results from the final layer are calculated and compared to the desired output producing an error measurement. This error information is propagated back from the output layer to the inner layers.

The input vectors or features often need to be adjusted. Preprocessing of input data is one of the most important steps in development of a neural network solution (Bishop, 1995). The data set may be missing values or valuable information can be obscured by an excessive number of attributes. In addition, the numerical values of the data are normalized to help equate the strength of the variables. Nominal data, such as colors, can be mapped to numerical equivalents and normalized as well. The data points that most affect the solution are optimal candidates for inputs while others are discarded. If too much information is removed, the resulting prediction ability will be affected.

The middle layers of nodes are added with a non-linear activation function. An activation function determines the output of a given node. The sigmoid function is typically used in the form of a hyperbolic tangent. This function is differentiable which is required to calculate back propagation.

The final layer's output is compared to the desired and an error is calculated. The gradient of this error function with respect to the weights is used to adjust the weights applied to each input of a node. Starting with the layer closest to the output nodes and working backwards. This process is repeated until overall performance of the network satisfies some user-defined limit.

The general process for creating an EBP network is described as follows:

1. Define a feature vector, gather the training data and present it to the network as input.
2. Determine the number of hidden and output layers.
3. Using an input feature vector compare the network's output to the desired output.
4. Calculate the error from each output node.
5. Incrementally adjust the weights of each node using the error calculations as a basis of calculation.
6. Using the error calculations for each node, feed the values back through the layers adjusting weights accordingly.

   Repeat steps 3 – 6 until some acceptable error level is reached.

2.4.4 Genetic Algorithms

A Genetic Algorithm (GA) is a search technique that incorporates ideas of natural evolution originally developed by John Holland in the 1970's. In general, a GA needs a population representation of possible solutions, variation operators, selection, and replacement mechanisms. The actual details, such as population representation and distance measure, of a GA can vary greatly and represent one of the challenges of an implementation.

In a GA an individual is a candidate solution out of a set of solutions called a population. This individual may be represented by a genotype where in turn each genotype maps to a phenotype (Eiben & Smith, 2003). For example, in a population of integers an integer value of 10 (phenotype) could be represented by a binary code of 1010 (genotype). Following this definition, it is critical to a robust solution that the genotype is capable of fully representing the optimal individual solution characteristics.

An evaluation function is used to determine the 'fitness' of individuals in a population. This fitness is a measure indicating how well the individual is solving a given problem and used for decision making in the evolutionary process. Given a fitness evaluation applying genetic operators such as crossover and mutation creates new individuals called offspring. In crossover portions of the candidate parent individuals are swapped and combined to form new offspring. Optionally a mutation, or random alteration, of the new offspring can be applied.

2.5 Anomaly Detection

Network intrusion detection systems (NIDS) originated in 1980's and in the seminal work of Denning, (1987). In general there are two types of NIDS; anomaly detection and signature based detection systems. Signature based detection systems attempt to match the observed behavior against a database of predefined signatures. In contrast, an anomaly based detection system seeks deviations from a learned model of normal behavior (Chandola, Banerjee, & Kumar, 2007). The system builds a representative normal behavior model based on observed data. It is then capable of detecting novel and dynamically changing intrusion instances, assuming that the instances differ from the model of normal behavior.

Anomaly IDS systems learn a model of normal behavior (Gosh, Schwartzbard, & Schatz, 1999). Certain key features are derived from observed traffic and a representative model of current activity is maintained. Subsequent monitored features are then compared to the historical model. Deviations from the model are recognized and treated as suspicious activity. This type of system is capable of detecting novel attacks or other abnormal systemic behavior that might be missed by rule based systems. Unfortunately, these new behaviors may be acceptable and simply were not present during the initial learning phase.

Rule based systems are analogous to virus protection software resident on personnel computers. Predefined rule sets capture characteristics of attack vectors. These sets perform well on known signatures but generally do not recognize novel attacks. In addition, minor variations in the signature of a known attack may not be noticed by the system. However, rules are developed and distributed by an active community and widely distributed. It is even possible to convert rules between various formats (Eckmann, 2001). This provides a rich base of historical information, as is shown in chapter 4, which is readily exploitable for use in CI based training.

2.6 Deceptive Virtual Hosts

Network security monitoring systems, including NIDS, are a significant part of a solution to protecting control systems. In most contexts, they are rarely capable of providing perfect intrusion detection (Sommer & Paxson, 2010). To improve the cyber-security of a network system several approaches can be applied. Deceptive systems, called honeypots, that emulate critical network entities have been deployed in tandem with monitoring solutions to improve detection accuracy and precision rates (Garcia-Teodoro, Diaz-Verdejo, Macia-Fernandez, & Sanchez-Casad, 2007), (McQueen & Boyer, 2009).

Lance Spitzner introduced the idea of a dynamic honeypot (DHP) in 2004. The concept is based on automatically configuring a honeypot by gathering information gleaned from network traffic. This type of system has the benefit of requiring little human input and can readily adapt to changes in network behavior. A DHP requires functionality in two key areas: 1. Network host information gathering, and 2. Generating honeypot host configurations for deployment.

Implementations of honeypots fall into two categories: high or low interaction. Low interaction virtual honeypots are used to gather information. They are simpler to deploy, less likely to be compromised and can work collaboratively with other agents. Additionally, they might distract an attacker from hitting real systems and thereby waste an adversary's valuable time and effort on a low value target.

High interaction honeypot systems are typically hardware replicas of existing operational components that include the appropriate software. For purposes of this discussion, virtual machines are included in the high category. These systems do not mimic services but are deployed with working instances. This type of system provides a high fidelity solution that is less prone to discovery of its deceptive purpose by network intruders. However, they are at a higher risk for compromise by an attacker and require a more complicated deployment investment. Deploying a virtual machine is simpler than a hardware base system but still requires complex management scenarios for deploying a wide array of service software. This includes having copies of multiple OS distributions and server software.

Honeyd, created by Niels Provos, is an open source project implemented as a small software daemon that creates virtual hosts on a network (Provos & Holz, 2007). It is a low interaction honeypot that emulates an OS network stack and minimally provides superficial service functionality. Because of this, an attacker is never able to gain access to the host by compromising a service but might quickly realize that something is amiss. The primary goal is not to entrap the attacker into spending all his effort on the deceptive system. It is to attract his attention, for at least a short time, and gather information that helps identify the attacker and a possibly compromised internal attack platform. Another Honeyd advantage is the ability to deploy thousands of virtual hosts on a single host thereby reducing hardware costs.

Finally honeypots, high or low interaction, can only detect attacks directed at them. A competent attacker who discovers that a system is a honeypot will avoid any further contact with that system. The fidelity of the deception is in the presentation of the honeypot to the network. How the data is gathered to create this deception is important.

2.7 Related Work

The application of existing security tools to implement network protection within an autonomic concept has not been widely explored (Ruan, Lacoste, & Leneutre, 2010). However the following section describes three recent efforts closely related to the work presented in this dissertation.

The work presented by Candido, Colombo, Barata and Jammes (2011) pursued a common architectural solution to support phases of a device lifecycle. It utilizes concepts of Evolvable Production Systems (EPS) and Service Oriented Architectures to implement a proof of concept in an industrial automation scenario. In contrast to the approach of using IF-MAP, a Devices Profile for Web Services (DPWS) standard is used to define devices. This work concentrated mainly on deployment of production devices with no mention made of security related equipment.

Kyusakov, Eliasson, Delsing Deventer and Gustafsson (2011) discuss the integration of SOAP based web service implementations in resource constrained wireless sensor nodes. They propose removing middleware and deploying the services directly on devices. This approach is shown to be feasible but limited by performance overhead of the communication scheme. A dedicated security device, as proposed in this dissertation, with multiple deployments might be more feasible by providing dedicated relatively powerful hardware. This assumes that addition of a new device is preferable to adding additional functionality to hardware constrained devices.

A security related architecture implementing virus detection and removal services for digital ecosystems has been proposed. In (Agrawal, Grosky, & Fotouhi, 2009) the authors present a distributed approach with hosts working in tandem on a network. A service-oriented architecture was suggested as a communication framework. Any host entities finding virus information in the network would share this information with other hosts thus enabling further actions. This communication mechanism and functional area are relevant to this dissertation. However no empirical tests were executed to validate the approach.

Chapter 3. Autonomic Intelligent Cyber Sensor

This chapter describes a novel implementation of the Autonomic Intelligent Cyber Sensor (AICS) architecture designed to provide cyber security and industrial network state awareness for Ethernet based control network implementations. The first contribution focus of this chapter is the concept of an integrated framework of communication fundamentals derived from Autonomic research and Service Oriented Architecture (SOA). The other three contributions of AICS utilize collaborative intelligent mechanisms to: 1) identify anomalous network traffic, 2) provide network entity information and, 3) deploy deceptive virtual hosts. Additionally, AICS dynamically reacts to the industrial human-digital ecosystem in which it resides through examination of network traffic and communication of data from external entities (Vollmer, Linda, & Manic, 2013). A proof of concept implementation is tested and results are presented.

3.1 AICS Architecture

This section describes the architecture of AICS. Reflecting some of the core attributes of autonomic computing, the design structure for this project is broken into functional areas as depicted in Fig. 3-1. The managed resources are Intelligent Anomaly

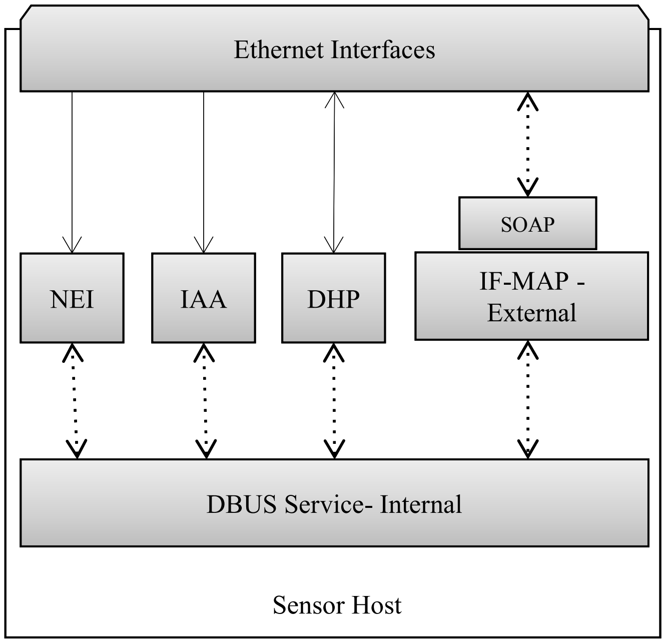


Figure 3-1. AICS Architecture and Data Flow

Assessment (IAA), Network Entity Identification (NEI) and Dynamic Honey Pot (DHP). Because of the flexible internal communication implementation, these resources modules can be enhanced or replaced with other potential solutions with little effect on the sensor operation.

The communications infrastructure provides a common interface for additional modules to be deployed on the sensor and is the autonomic manager. As can be seen in Fig. 3-1, the dotted lines represent messaging paths while the solid lines are direct network connections. Some modules are composed of legacy software that inherently did not communicate via D-Bus. For these components, software wrappers were required. Each individual component is explored in more detail in the following sections.

3.1.1 Communication Infrastructure

The communication component is responsible for the external and internal communication mechanism of the sensor.

3.1.1.1 External Communication

External communication is the transfer of information to network entities outside the confines of the sensor system. Examples of this communication include a list of IP addresses to monitor, information on network entities and alerts on abnormal network traffic. Internal communication includes the sharing of information between sensor components. An example includes the passing of passively discovered network entity information from NEI to DHP. All external communication is restricted to the IF-MAP based component with the exception of network packet monitoring and emulated honeypot hosts.

External communications conform to the IF-MAP specification. The published Web Services Description Language (WSDL) document for IF-MAP describes the interfaces available for ad hoc requests to the service. WSDL is an XML-based language for describing Web services and how to access them. The interface is exposed on the network over the SSL secured HTTP protocol. The Intelligent Reaction on Network Events project produced an open source IF-MAP server called Irond (Trust@FHH, 2012). This Java based program was used as the MAP communication server. The AICS message communication is solely with the MAP server.

Anomaly alerts and network entity metadata from the sensor system are pushed to the MAP server via an HTTPS based soap call. These asynchronous information pushes are executed upon the identification of behaviors in the network traffic or internal monitoring processes. External entities that have interest in this type of information need to register with the map server. Registered clients receive copies of the messages.

In the IF-MAP definition, the metadata type may be either singleValue or multiValue. This is explicitly communicated by setting the ifmap-cardinality attribute in a message. The D-Bus to IF-MAP service for host publication, described later, sets this to singleValue. When a publish request is processed this attribute determines how the MAP server updates the records for a given identifier. Single replaces the information while multi adds to the existing record even if it contains duplicate data. Because singleValue was chosen, a republish of the entire record is necessary even if only one sub-item changes. This simplifies publishing of records at the expense of added communication overhead.

All incoming messages are delivered from the MAP server. This includes configuration or capability querying. The sensor binds to an IF-MAP notify interface for updates and responds accordingly. Consequently asynchronous information is obtained through notify events. These events can be requested by interested outside parties as well. However they are ephemeral in that only subscribed clients will get the information if they attach prior to message publication.

The IAA IP address monitoring list and DHP IP address emulation list are sent via messaging from an external authority. Internally these lists are stored as a configuration file in each component. The stored IP's are then used to extract information from the internal messages created by NEI and used to configure the honeypot hosts. The IAA module tracks the monitor IP addresses for anomalies. The IAA, NEI and DHP modules are explained in more detail later in this section.

The presented two-layer path of communication isolates the sensor from direct network contact with other entities. The externally visible attack surface is limited to one component instead of multiple components. A primary benefit is that any change in external protocols affects only one component. Additionally, it provides the capability to wrap the legacy communication of internal tools that may not be IF-MAP compliant without having to modify the tools source. Finally, it provides a convenient debugging facility.

3.1.1.2 Internal Communication

There is no direct communication between the internal sensor components and the outside MAP server or between modules. All messages, regardless of internal or external destination, use the D-Bus IPC mechanism described in the background chapter. A D-Bus service for centralized processing of external messaging was created. Any module on the sensor host that wishes to send messages externally need only call the sendMessage method of the service. This provides flexibility and security with the additional expense of complexity from adding the D-Bus service. Only the message service needs be concerned with the communication particulars providing for a simplified configuration.

A service file, called org.sa.MessageService.service, created in the /usr/share/dbus-1/services directory specifies what file to execute if the D-Bus service is not already running. This ensures that client messages will be delivered even if the receiving process is not running. Messages are queued up in the D-Bus system until the appropriate process is started and then delivery ensues. The service is started with the same user id as that which belongs to the caller. This means it is possible to have multiple instances running, one for each user id that makes a call. The configuration values enable a rich set of operations for limiting access to the appropriate resources. A simple example service file is shown next.

```
[D-BUS Service]
Name=org.sa.MessageService
exec /opt/Sensor/MessageService/bin/msg_service.pl'
User=sensor
```

Figure 3-2. Example D-Bus Service

By default *system* and *session* bus types are provided by the Linux D-Bus daemon. A new *session* bus instance is created for each user login session. This can result in multiple busses or none depending upon interactive user logins. In contrast to the *session* bus, a singleton *system* bus is instantiated upon host startup.

The communication service was attached to the *system* bus, as it should not be dependent upon an interactive login. The *system* bus has restrictions where access is explicitly prohibited. Configuration changes were required to allow access for the specific user id associated with the communication service. Additionally, fine grain access control lists are maintained for access to the services provided. This includes who can call the methods and who can retrieve information from the bus.

The following sections of this chapter describe the managed resource components of the AICS. These components are all clients to the D-Bus internal message service just described.

3.1.2 Intelligent Anomaly Assessment

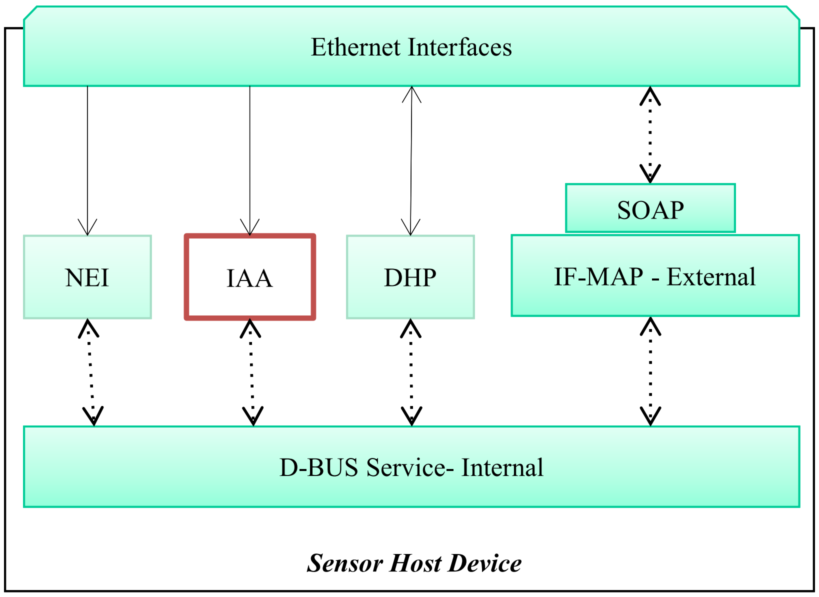


Figure 3-3. IAA Component

The Intelligent Anomaly Assessment (IAA) component reacts to information from external messaging and passive monitoring of network traffic. An internal bus delivers an externally originated message dictating, by IP address, which hosts are to be actively monitored. The IAA selectively monitors hosts for anomalous behavior while simultaneously utilizing global network trends to adjust its sensitivity. Any anomalous behavior triggers a message passed internally to the D-Bus service and ultimately provided to the external IF-MAP message service.

To ensure the cyber-security of network system various approaches can be applied (Sommer & Paxson, 2010). One of the most common approaches is anomaly detection. An anomaly detection system is trained on a set of normal network behavior. The extracted behavior model is then used to detect anomalous behavior in the newly observed testing data.

Two difficulties that might occur with this approach are identified as follows. First, building a single comprehensive normal behavior model for a specific system might be difficult due to the complexity of the network and the presence of multiple diverse communication streams. Second, the performance of anomaly detection algorithms can be tuned by adjusting a sensitivity threshold. As is shown later, the selection of a specific threshold value inevitably results in a tradeoff between false negative and false positive rate. Thus implementing a suitable sensitivity threshold value is an important feature of IAA.

The IAA component is an extension of a previously developed low-cost online rule extraction technique (Linda O. , Vollmer, Wright, & Manic, 2011). The model is composed of a set of fuzzy rules that are constructed based on a window-based feature vector using an online version of the adapted Nearest Neighbor Clustering (NNC) algorithm. The anomaly detection algorithm was specifically designed to allow for both fast learning and fast classification on the constrained computational resources of an embedded device.

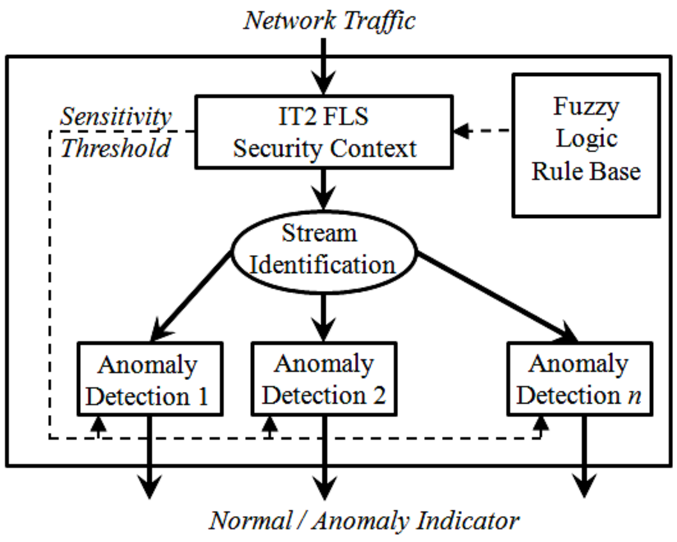


Figure 3-4. IAA Fuzzy Design

The overall architecture of the proposed anomaly detection system is depicted in Fig. 3-4. The network traffic features are processed by an Interval Type 2 Fuzzy Logic

System (IT2 FLS) that uses a fuzzy logic rule base with encoded linguistic domain knowledge to calculate the cyber-security context. This cyber-security context expresses the belief that an intruder is currently present in the system.

Fuzzy logic provides a framework for system modeling in linguistic form capable of coping with imprecise and vague meanings of words. The Type-2 Fuzzy Logic System is an extension of the Type-1 Fuzzy Logic System (T1 FLS) that has been successful in modeling and minimizing the effects of various kinds of dynamic uncertainties. Here the Interval Type-2 Fuzzy Logic System (IT2 FLS), as a specific case of T1 FLS, is used to encode the linguistic domain knowledge about the specific network system and dynamically adjust the sensitivity of the anomaly detection algorithms.

In the next stage, the network traffic is separated into individual communication streams. In the current implementation, a specific IP address is used to identify each communication stream. Other features, such as port numbers and protocol types could be used. Packets assigned to individual communication streams are then passed into dedicated anomaly detection algorithms. Each anomaly detection algorithm maintains its own buffer of incoming packets, which is used to extract the window-based features as described later in this section. The fuzzy logic based anomaly detection algorithm is used to assign a real value to each input vector, which expresses the belief that the current packet window contains intrusive packets. The closer this value is to 1 the more confident the algorithm is that an intrusion is present.

The windowing technique extracts statistical features from a limited set of consecutive packets and maintains them in a vector. The window is shifted over the stream of incoming network packets. As the next arriving packet is inserted into the window, the last packet is removed from the end. The new feature vector is then computed from all of the packets currently present in the window.

As described in (Linda, Vollmer, & Manic, 2012) the fuzzy logic based anomaly detection algorithm is used to assign a real value to each input vector. This value expresses the belief that the current packet window contains intrusive packets. The closer a value is to 1 the more confident the algorithm is that there is an intrusion present.

The final classification is performed by comparing the real-valued output to the sensitivity threshold. When the real-valued output is above the sensitivity threshold, a network anomaly is reported for the specific communication stream. When the output value is below the sensitivity threshold the network traffic is marked as normal. The actual value of the sensitivity threshold is dynamically computed based on the cyber-security context computed by the IT2 FLS. Hence, the IT2 FLS encoding human domain knowledge is not used directly for detecting anomalies; instead it is used to only tune the performance of the anomaly detection algorithm via adjusting the sensitivity threshold.

3.1.2.1 Window based Feature Extraction

The anomaly detection algorithm is trained on a set of network traffic features extracted by a window-based technique. This technique is applied directly to the stream of packets. Internal to IAA, a vector stores the statistical properties of the network traffic in the window.

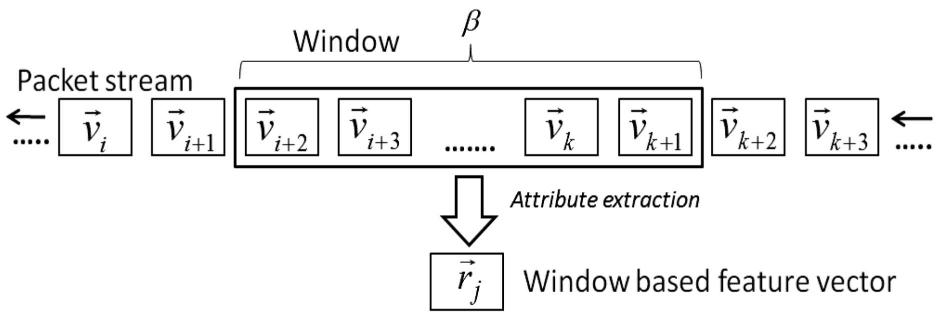


Figure 3-5. Feature Vector Window

As is described in previous work, a window of specified length is shifted over the stream of network packets (Linda, Vollmer, & Manic, 2009). At each position of the window, a descriptive feature vector is computed. As the next arriving packet is pushed into the window, the last packet is removed from the end. Fig. 3-5 schematically depicts

Table 1. Derived Window Features

Num. of IP addresses	Num. of Flag Codes
Min. Num. of Packets / IP	Min. Num. of Packets / Flag Code
Max. Num. of Packets / IP	Max. Num. of Packets / Flag Code
Avg. Time between Packets	Num. of Packets with 0 Win. Size
Time Length of the Window	Num. of Packets with 0 Data Len.
Data Speed	Avg. Win. Size
Num. of Protocols	Avg. Data Length
Min. Num. of Packets / Protocol	Num. of Ports
Max. Num. of Packets / Protocol	

this feature extraction process. Table 1 summarizes the list of extracted statistical features from the packet window. This set of features was empirically selected based on the motivation to capture the time series nature of the packet stream and provide for a wide range of behaviors. For further details and an evaluation of the feature extraction refer to (Linda O. , Vollmer, Wright, & Manic, 2011).

Up to this point the two classes of feature information include: 1) statistics and information about packets that reside within a packet buffer, i.e., time differences, number of protocols seen, and 2) individual packet header information such as protocol type, payload size, etc. The information contained within the network packets is only partially exploited by use of these features. Additional features, derived from the packet payloads, can improve accuracy and precision (Sommer & Paxson, 2003). One source of packet payload information can be found in the SPADUC (SCADA Protocol Anomaly Detection Using Compression Techniques) project (Rueff, Roybal, & Vollmer, 2013).

The purpose of the SPADUC project is to investigate whether a compression algorithm can be utilized to differentiate between proper control system traffic versus hostile network traffic. Supervisory Control and Data Acquisition (SCADA) systems operating over TCP/IP protocols in network hardware typically contain several static layers of headers. The ratio of header size to control or response data is often very high. Moreover, the types of headers are often limited because of the repetitive nature of SCADA networks. Control of hardware and status of systems and sensors often occur along regularly timed sequences. Therefore, network traffic on dedicated SCADA systems and at the boundaries of SCADA systems where data transfer takes place tend to be of a cyclic, predictive nature. Because a large portion of the TCP/IP message traffic is repetitive, the concept of using compression techniques to identify abnormal traffic is proposed as a way to identify and quarantine malicious traffic at the packet level on a SCADA network.

Initially, to get a baseline for the use of compression in IAA, the commonly used Zlib compression library was used. The Zlib routines report enough information to determine the resulting byte size of a compressed data buffer. This value was added as the only feature instead of the existing 17 feature set. Two data compression candidates are individual network packet data portions and a buffer filled with all 20 network packet data portions concatenated together. The AICS algorithm uses a sliding 20 packet buffer routine to calculate inter packet statistics. A test using both data candidates was performed on a single IP. The results of the two runs are shown in Table 2. With the exception of false negatives the buffer approach provided better results. This implementation was chosen for further exploration.

Table 2. Initial Compression Results

Data Source	Correct	Incorrect	False Negative	False Positive
Avg. Single Packet	78.49	21.51	45.3	14.6
20 Packet Buffer	83.05	16.95	75.2	0.02

Utilizing the compression library resulting from early SPADUC project results, an implementation using the 20 packet buffer was explored. The SPADUC library routines return the number of encodes performed on the data buffer. All of the standard ICS features were removed and only the SPADUC encode value was considered. The results of this testing on nine different IP network streams are shown in Table 3. A general trend observed is a high false positive rate and a low false negative rate. The variation in percentage values might be attributable to the heterogeneous nature of the devices tested.

Table 3. SPADUC Compression Feature Results

IP Address	Correct	Incorrect	False Positive	False Negative
*.99.5	91.68	8.3	0	53.71
*.99.101	99.67	0.33	0	3.46
*.99.107	95.32	4.68	0	26.91
*.99.135	90.55	9.45	0	100
*.99.140	96.98	3.02	0	62
*.99.160	35.49	64.5	26.7	72
*.99.170	68.99	31	0	99.9
*.99.206	97.60	2.40	0	46.8
*.99.220	21.05	78.95	0	94.49

The original feature vectors with the addition of the compression feature were tested using identical input and configuration.

Table 4. All Features including Compression

IP Address	Correct	Incorrect	False Positive	False Negative
*.99.5	99.29	0.71	3.17	0
*.99.101	89.80	10.20	26.01	0
*.99.107	99.33	0.67	2.64	0
*.99.135	97.64	2.36	2.36	0
*.99.140	93.81	6.19	6.19	0
*.99.160	98.16	1.84	67.91	0

*.99.170	97.64	2.36	14.74	0
*.99.206	80.94	19.06	28.11	0
*.99.220	24.67	75.33	77.40	0

Finally the following table shows the results of running the system with the original 17 feature set.

Table 5. Original Features

IP Address	Correct	Incorrect	False Positive	False Negative
*.99.5	99.87	0.13	.58	0
*.99.101	99.8	0.20	0.5	0
*.99.107	99.8	0.20	0.78	0
*.99.135	98.33	1.67	1.67	0
*.99.140	99.51	0.49	0.49	0
*.99.160	98.16	1.84	67.91	0
*.99.170	99.18	0.82	5.16	0
*.99.206	99.85	0.15	0.20	0.04
*.99.220	24.67	75.33	77.40	0

A comparison of Table 4 with Table 5 shows that there was no improvement in any of the categories from addition of the packet payload compression information. The result of utilizing just the single compression feature indicates that there is some value in the compression information. However, it may be that the compression information is redundant. Additionally, given the relative large computational expense of the compression algorithm, the compression feature is not included in the rest of the IAA discussion.

3.1.2.2 Fuzzy Logic Rule Extraction via Online Clustering

A low-cost online rule extraction technique is utilized to model the network traffic (Linda, Vollmer, & Manic, 2012). The model is composed of a set of fuzzy rules that are constructed based on the window-based feature vectors using an online version of the adapted Nearest Neighbor Clustering (NNC) algorithm. The NNC is an unsupervised clustering technique. As opposed to other commonly used clustering algorithms such as K-Mean or Self Organizing Maps, the NNC algorithm does not require a priori specification of the expected number of clusters. Instead, the clustering process is controlled by an established maximum cluster radius parameter. The smaller the radius the more clusters will be generated and vice versa. This adapted algorithm maintains additional information about the spread of data points associated with each cluster

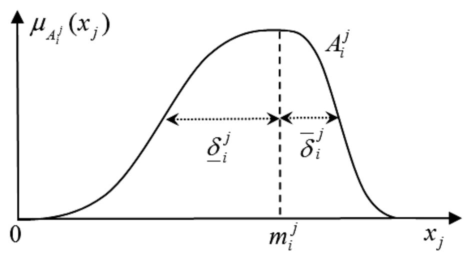


Figure 3-6. Non-symmetric input Gaussian fuzzy set A.

throughout the clustering process. Each cluster P_i of encountered normal network behavior is described by its center of gravity $\vec{c}_i$, weight w_i and a matrix of boundary parameters M_i. Hence:

$$P_i = \{\vec{c}_i, w_i, M_i\}, \vec{c}_i = \{c_i^1, \ldots, c_i^n\}, M_i = \begin{vmatrix} c_{i,1}^U & \cdots & c_{i,n}^U \\ c_{i,1}^L & \cdots & c_{i,n}^L \end{vmatrix} \tag{3-1}$$

Here, i is the index of the particular cluster, c_i^j is the attribute value in the j^{th} dimension, $c_{i,j}^U$ and $c_{i,j}^L$ are the upper and lower bounds of the encountered values of the j^{th} attribute for data points assigned to cluster P_i and n denotes the dimensionality of the input. The algorithm is initialized with a single cluster P_1 positioned at the first supplied training input vector $\vec{x}_1$. This initial input vector is received once the shifting window is first filled with the incoming network packets.

Upon acquiring a new data vector $\vec{x}_i$ from the shifting window buffer, the set of clusters is updated according to the NNC algorithm. First, the Euclidean distance to all available clusters with respect to the new input feature vector $\vec{x}_i$ is calculated. The nearest cluster P_a is identified. If the computed nearest distance is greater than the established maximum cluster radius parameter, a new cluster is created. Otherwise the nearest cluster P_a is updated according to:

$$\vec{c}_a = \frac{w_a \vec{c}_a + \vec{x}_i}{w_a + 1}, w_a = w_a + 1 \tag{3-2}$$

$$c_{i,j}^U = \max(x_i^j, c_{i,j}^U), c_{i,j}^L = \min(x_i^j, c_{i,j}^L)\ j = 1 \ldots n \tag{3-3}$$

The rule extraction phase of the learning process produces a set of clusters, which describe the normal network communication behavior. In the next stage, each cluster is converted into a fuzzy logic rule. Each fuzzy rule describes the belonging of a particular sub-region of the multi-dimensional input space to the class of normal behavior.

Each cluster is transformed into a fuzzy rule. Each fuzzy rule is composed of n antecedent fuzzy sets A_i^j that are modeled using a non-symmetric Gaussian fuzzy membership function with distinct left and right standard deviations. There are three parameters of the membership function, the mean m_i^j and the left and the right standard deviations $\overline{\delta}_i^j$, $\underline{\delta}_i^j$, as shown in Fig. 3-6. The parameter values are extracted based on the computed cluster P_i in the following manner:

$$m_i^j = c_i^j \tag{3-4}$$

$$\overline{\delta}_i^j = \alpha(\overline{c}_i^j - c_i^j) \tag{3-5}$$

$$\underline{\delta}_i^j = \alpha(c_i^j - \underline{c}_i^j) \tag{3-6}$$

Here symbol α denotes the fuzziness parameter, which is used to adjust the spread of the membership functions. This set of fuzzy rules is then used to calculate a similarity score between the input vector and a stored model of normal behavior.

NNC Algorithm Variations

For an input pattern, the nearest cluster is determined using the Euclidean distance norm:

$$dist(\vec{c}_a, \vec{x}_i) = \min_j \sqrt{\left(c_j^1 - x_i^1\right)^2 + \ldots + \left(c_j^n - x_i^n\right)^2}, j = 1 \ldots C \tag{3-7}$$

The Euclidean distance is a simple geometric distance based on the Pythagorean formula. This measure is computationally inexpensive and simple to code. However it does have two drawbacks. First, in the geometric problem domain, variables are typically measured utilizing the same units of length. Data values from real world problems may have different scales. For example a regression problem making use of class information such as age, test scores and time are all on a different scale and therefore not directly comparable. The Euclidean distance is sensitive to the scales of the variables involved and may not perform optimally. A standardized or weighted Euclidean distance that incorporates variance but not covariance can overcome this problem. The Mahalanobis distance incorporates both variances and covariances.

Second, the Euclidean distance does not compensate for correlated variables. Given a test data set containing multiple variables where one variable set is an exact duplicate of another set, these sets are highly correlated. The Euclidean distance calculation will weigh the duplicate variables more heavily than the others. It has no method of accounting for the fact that the duplicate provides no new information.

P.C. Mahalanobis introduced the Mahalanobis distance in 1936. It is based on both the mean and variance of the variables in addition to the covariance matrix. The iso-surface formed around the mean is an ellipse in two-dimensional space or an ellipsoid or hyper-ellipsoid when more variables are used. It is a multivariate quantitative method that can solve for multiple dimensions simultaneously. The covariance among the variables is taken into account when calculating the distance. Because of this, the problems of scale and correlation inherent in the Euclidean distance are not an issue. Given an individual as a vector $\vec{x}_i = (\mathrm{x}_0, \ldots, \mathrm{x}_\mathrm{n})$ of floating point values *x*, a vector representing the mean of a data set $\vec{\mu} = (\mu_0, \ldots, \mu_n)$ and a covariance matrix C of size *n x n* representing the covariance values between all dimensions *n*, the Mahalanobis distance is calculated with the given formula:

$$md(\vec{x}_i) = (\vec{x}_i - \vec{\mu})C^{-1}(\vec{x}_i - \vec{\mu})^T \qquad \textbf{(3-8)}$$

Formula 3-8 shows three variables that need definition. The first is a feature vector $\vec{x}_i$ which is gleaned from the input network packets. The second is a sample mean $\vec{\mu}_i$. The original NNC code was changed to calculate and store the mean of the 17 network features ($\vec{x}_i$) found during the initial learning phase. The covariance matrix C^{-1} is a matrix of all the covariance's between each feature vector. Given a data point and a sample mean the covariance can be calculated. In the case of AICS this produces a 17 by 17 covariance matrix. Formula 3-8 specifies an inverted, or nonsingular, covariance matrix. Not all matrices are invertible. The Eigen decomposition of the sample covariance matrix gleaned from test data was computed. This produced Eigen values that were zero and thus indicating that the matrix was not invertible. This prohibited the use of Mahalanobis as a distance measure with the AICS features.

It has been noted that the use of a Euclidean distance measure on high dimensional data can be problematic (Xu & Wunsch, Clustering, 2009). When used as a nearest neighbor measure in high dimensions, data points that appear to be relatively close in low dimensions can be falsely determined in higher dimensions. One possible solution to this is to use a cosine similarity measure. This measure is the cosine of the angle between two vectors. Each data point is treated as a vector. The Euclidean dot product, or inner product, is calculated as in Eq. 3-9 and equates to the cosine of the angles between the vectors. The angle thus determines whether two vectors are pointing in roughly the same direction.

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i\,)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} \qquad (3\text{-}9)$$

The unmodified NNC routine utilizes a 17 dimensional feature set. Each data point is normalized to a floating point value between 0.0 and 1. The NNC algorithm utilizes the distance measure as a key criterion for a binary decision of cluster membership. The first action is to determine if a candidate point falls within a predefined configurable distance to already defined clusters. If this occurs the point is added to the cluster with the minimum distance. If not then a new cluster is created and the candidate point is the exemplary member. For our normalized Euclidean distance, this value was 0.25.

The cosine similarity algorithm was applied to a test set of network data. The definition of a membership value for a cosine angle was explored. We were unable to find a single value, that when applied to all the data, could adequately separate the anomalies from normal vectors. There were individual cases of a define value correctly differentiating the input vectors. However, the overall accuracy never surpassed that of the original algorithm. It is surmised that the number of data points in the feature points made the differences in the angle too small to detect. This is partially supported by the effectiveness of the original Euclidean measure based system. In a test run, the cosine distance measures were captured, this resulted in 23,021,335 entries. The run produced 376,166 unique values with a minimum value of 0.587196 and a maximum of 1. The average of the values was 0.81099 with a standard deviation of 0.11011.

3.1.2.3 Representing Domain Knowledge Using Linguistic Fuzzy Rules

This section provides a brief introduction to Interval Type-2 Fuzzy Logic and describes a methodology for representing cyber-security domain knowledge.

Interval Type-2 Fuzzy Logic Systems

Type-1 Fuzzy Sets (T1 FSs) and T1 Fuzzy Logic Systems (FLSs) have been successfully applied in many engineering areas (Valente de Oliveira & Pedrycz, 2007). However, when modeling linguistic terms, which can mean different things to different people, T1 FSs have been shown to provide only limited design capabilities (Mendel, Uncerain Rule-Based Fuzzy Logic Systems: Introduciton and New Directions, 2001). To address these issues, Type-2 (T2) FSs and T2 FLSs were originally proposed by Zadeh (1975). T2 FSs offer more modeling flexibility because they employ membership degrees that are themselves fuzzy sets (Beglarbegian, Melek, & Mendel, 2011).

In this section, the Interval T2 (IT2) FSs are considered. IT2 FSs restrict all membership grades into intervals, which result in significant simplification of the computational complexity associated with computing with IT2 FSs. An IT2 FS $\tilde{A}$ can be described by its membership function $\mu_{\tilde{A}}(x,u)$, where $x \in X$ and $u \in J_x$ (Mendel, Uncerain Rule-Based Fuzzy Logic Systems: Introduciton and New Directions, 2001):

$$\widetilde{A} = \int_{x\in X}\int_{u\in J_x} 1/(x,u) \; J_x \subseteq [0,1] \tag{3-10}$$

Here, x and u are the primary and the secondary variables and J_x denotes the interval support of the secondary membership function. The domain of the primary memberships J_x defines the Footprint-Of-Uncertainty (FOU) of FS $\tilde{A}$:

$$FOU(\widetilde{A}) = \bigcup_{x\in X} J_x \tag{3-11}$$

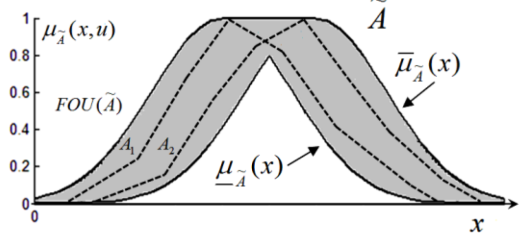


Figure 3-7. Interval type-2 fuzzy set $\tilde{A}$.

The FOU of an IT2 FS can be completely described by the upper and lower membership functions:

$$FOU(\tilde{A}) = \bigcup_{\forall x \in X} (\underline{\mu}_{\tilde{A}}(x), \bar{\mu}_{\tilde{A}}(x)) \tag{3-12}$$

It is this FOU that allows for modeling of linguistic uncertainty. As an example depicted in Fig. 3-7, consider two possibilities for modeling an arbitrary linguistic concept using T1 FSs A_1 and A_2 (e.g. two experts designed two different membership functions for the same concept) and the possible model of this concepts using IT2 FSs $\tilde{A}$. It can be seen that the IT2 FS encapsulates the T1 FS models and it can model the linguistic uncertainty. This flexibility in modeling vague linguistic concepts was the reason for employing IT2 FSs and IT2 FLS for modeling the linguistic human cyber-security domain knowledge in the proposed system.

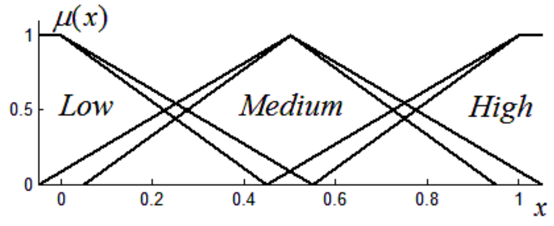
Linguistic knowledge can be formulated using implicative IT2 fuzzy rules as follow (Mendel, Uncerain Rule-Based Fuzzy Logic Systems: Introduciton and New Directions, 2001):

$$\text{Rule } R_k\text{: } \textbf{IF } x_1 \text{ is } \tilde{A}_1^k \textbf{ AND } \ldots \textbf{ AND } x_n \text{ is } \tilde{A}_n^k \textbf{ THEN } y_k \text{ is } \tilde{B}^k \tag{3-13}$$

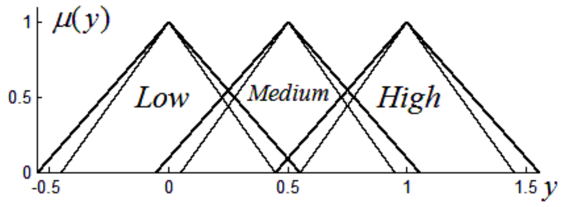
Here, symbols $\tilde{A}_i^k$ and $\tilde{B}^k$ denote the i^{th} input IT2 FS and the output IT2 FS of the k^{th} rule, respectively, where n is the dimensionality of the input vector $\vec{x}$ and y_k is the associated output variable.

The set of linguistic rules together with the representation of the input and output IT2 FSs can be used to create an IT2 FLS. The specific technical details of fuzzy inference using IT2 FLSs have can be found in literature (Mendel, John, & Liu, 2006).

Cyber-Security Domain Knowledge Modeling



(a)



(b)

Figure 3-8. Input IT2 FSs (a) Output IT2 FSs(b)

The IT2 fuzzy rules can be used to linguistically describe the relationship between various features of the network communication and the possibility of a cyber attack. The window-based feature extraction technique is used to describe the global features of the monitored network traffic.

Each window-based feature is first normalized into a unit interval. There are different approaches to fuzzifying the input domain of each attribute. Because of its simplicity, the fuzzification scheme depicted in Fig. 3-8(a) was used in the presented work. Here, two trapezoidal and one triangular IT2 fuzzy sets were used to fuzzify each input domain into fuzzy sets "*Low*", "*Medium*" and "*High*".

The output IT2 FSs express the likelihood of an intrusion in the system and can be used to adjust the sensitivity threshold of each anomaly detection algorithm. As was chosen for the input domain, the output domain is modeled using the three triangular IT2 FSs: "*Low*", "*Medium*" and "*High*". These sets are depicted in Fig. 3-8(b).

The provided set of linguistic fuzzy rules and the described input and output IT2 FSs are used to implement an IT2 FLSs, which calculates the specific sensitivity threshold of the anomaly detection. For instance, the domain knowledge can be encoded using IT2 FL rules such as: "*If number of protocols is high then sensitivity threshold is low*".

The anomaly detection algorithm utilizes an assumption that a representative normal behavior training data set has been collected. If a representative training data set of normal behavior is not collected, all subsequent observed traffic is marked as anomalous if the system is in an observation state. This IAA behavior is a fundamental concept underlying the use of anomaly detection techniques and is utilized to monitor dynamic virtual honeypots as is described later in this chapter.

3.1.3 Network Entity Identification

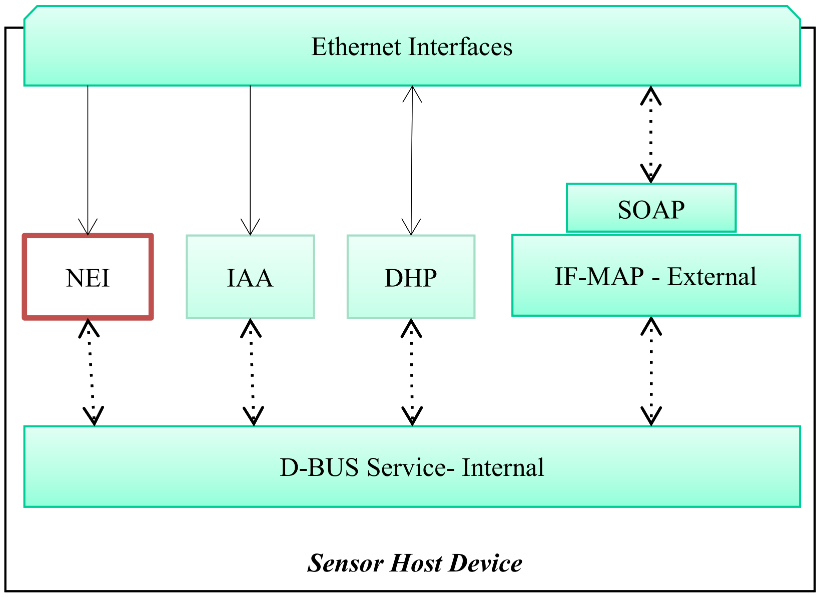


Figure 3-9. NEI Component

The Network Entity Identification (NEI) process passively monitors network traffic from which it extracts the IP and MAC address, ports and probable OS identification of hosts. While the primary purpose of NEI is to support dynamic virtual host configuration, the host information is valuable in of itself. This network entity information is updated on a continuous basis and made available, as messages, to both internal sensor components and external communications.

3.1.3.1 Passive vs. Active Scanning for DHP support

The two primary means for gathering the necessary network host information to create a honeypot includes passive and active network scanning. Unfortunately, most research to this point provides minimal analysis on suitable tools for passive information gathering. This is a key enabling capability if the intent is to deceive an attacker into believing an emulated system is real. This deficiency is corrected by examining characteristics of six existing tools and consequently recommending a tool, previously not used in this context, called Ettercap (2012).

An evaluation was conducted on six passive network information gathering open source tools to determine their strengths and weaknesses relevant to support of automated configuration. The tools evaluated for providing network host identification are: p0f (2012), Tshark (2012), Ettercap, Snort (Sourcefire Inc., 2009), Tcpdump (2012) and Ntop (Deri, 2012). Of the six tools, Ettercap and Ntop provide well-formatted structured output as an option.

Another tool, called SinFP (Auffret, 2010), was removed from consideration because it did not execute correctly on the test sensor system. SinFP is an operating system identification tool that incorporates both active and passive scanning. We were not able to get this tool to behave properly on our test system. It would execute and never return when running on stored network pcap files. Little time was spent trouble shooting the problem and it could be examined in future efforts.

In most of the literature reviewed, passive scanning has been implemented with p0f and occasionally Snort (Hieb & Graham, 2004), (Hecker & Hay, 2010). P0f is a command line tool that utilizes an array of mechanisms to identify hosts in a network stream. It is a passive OS fingerprinting tool frequently cited in creation of dynamic virtual honeypots. P0f by default only inspects the specifics of a SYN packet found in the three way handshake of a TCP connection. Not all systems in a network make this type of connection, or the connection initiation may be outside the window of observation, and without them p0f will not be able to make an OS determination. This p0f feature means that any hosts not sending SYN packets will be ignored and therefore not tracked as an entity on the network. In addition, port activity is only provided as a port number for each individual packet as opposed to a succinct summary output record. Port usage provides important network behavior information as it leads to deriving a host's service availability.

Although inherently an intrusion detection system, Snort may be run in packet capture mode. This mode prints individual packet information to standard out and provides a protocol summary. It does not track network host connections and the free form text output must be parsed for information. This output should not be confused with the intrusion alert outputs, which are more flexible in form and distribution capability. While individual packet information is potentially useful for gathering low-level details such as IP addresses and MACs, the output is not readily parsable for identifying network entities. For completeness, a commercial version of the Snort program that was not evaluated does provide a passive OS detection system.

Tshark is a popular open source network protocol analyzer. It is a command line version of the Wireshark analyzer that is capable of capturing live network traffic or reading data from a capture file. In addition to printing individual packet information, statistical and summary information can be printed. However only the packet information is well formed. This makes programmatic processing of the information difficult. In addition to the data format, several repeated executions of the software with different command options are necessary to gather the requisite information to meet the requirements of this project. Finally, output for each command is unique and requires custom parsing. Despite Tshark's output challenges the summary information it provides is extensive and useful for HoneyPot configurations. Sample output statistics include: Conversation tables for eight different types, Protocol statistics, IP hosts, and port values.

Tcpdump is a command line network capture and display tool. Individual packet information in configurable levels of detail is written to standard out. Similar to Snort, the tool provides potentially useful information but it is at a finer granularity than is necessary and requires a custom parsing tool. In future development, this type of detail might prove relevant to creating a higher fidelity of service level emulation.

Ntop, created by Luca Deri, is an actively maintained network usage reporting tool. Ntop utilizes the Ettercap database and mechanisms for OS discovery in addition to its own implementation of network host tracking. It has two mechanisms for delivery of information: web visualization and a formatted text stream. The text stream does not contain the same information as displayed via the http protocol. The OS identification string and host port specifics are not present. These missing pieces of information are important for configuring a dynamic honeypot. This deficiency provided one decision point used to drop Ntop in favor of Ettercap.

The amount of information that may be gleaned from passive scanning is a limited subset of possible information (Gagnon & Esfandiari, 2011). A passive scanning based tool is restricted to only gathering data that is offered in the captured stream. If a service on a host is available, but not utilized, this data point will be missed. Active scanning may prove more successful at extracting this type of information.

Table 6. Tool Capability Matrix

Tool	OS Identify	Port identify	MAC Vendor
ettercap	yes	yes	yes
ntop	yes	yes	yes
p0f	yes	no	no
snort	no	yes	no
tcpdump	no	yes	no
tshark	no	yes	yes

Nmap is an active scanning tool that has proven useful for interrogating hosts on a network (2012). However, a downside to active scanning is the possible interruption of services on hosts. This problem is especially acute in control systems. For instance, ping sweeps on older systems have been known to disrupt normal operation and cause physical damage (Duggan, Berg, Dillinger, & Stamp, 2005). Active scanning also provides a beacon of network activity outside the norm and could be revealing for intruders listening in on the traffic. In either case of active or passive scanning, the resulting information may be used to configure a honeypot.

In addition to identifying network entities, NEI needs to provide the information necessary to create a representative virtual network presence. The essential capabilities examined were operating system identification, port or service identification per host and the capture of MAC addresses with a resolution to the appropriate vendor (Hecker &

Table 7. Tool Performance Results

Tool	# OS ID	# of Hosts	# of MACs	# of IPs
ettercap	16	45	35	44
ntop	0[a]	202	43	39
p0f	13	NA	NA	10
tshark	NA	NA	69	44
tcpdump	Not Tested			
snort	Not Tested			

[a]Ntop displays OS information only in the web output.

Hay, 2010). Considering the results in the critical capabilities matrix shown in Table 6, it is clear that Ntop and Ettercap fulfill all three criteria. Of the two candidates, Ettercap was chosen for its support of XML output, completeness of information provided from this output and available functionality for support of future work.

Table 7 presents the results when running the tools against the test network described in the DHP section. The system, as configured during the test, had 46 physical connections to the network. The second column contains the number of operating systems identified by each tool. Ntop's identification of 202 hosts in column 3 contains duplicate entries for entities that have both IPv4 and IPv6 addresses. Additionally, records created for broadcast addresses inflate the host number. Ettercap outperformed or equaled the other tools in three of the four categories.

Ettercap is an extensible network manipulation and reconnaissance tool. It is an established and popular tool in the hacking community. However, this research is the first to establish its use as a source of information for dynamic honeypot creation. It was run as a daemon process with unified sniffing. In this mode it maintains internal network host records and updates them as new information is found. A binary log file is continuously updated as well. An Ettercap companion executable Etterlog is then run on the log file

```
<host ip="192.168.99.220">
    <mac>00:12:3F:61:4A:69</mac>
    <manuf></manuf>
    <distance>1</distance>
    <type>LAN host</type>
    <fingerprint type="unknown">
      2000:05B4:80:08:1:1:1:0:S:34
    </fingerprint>
    <os type="nearest">D-Link DWL-900AP</os>
    <port proto="udp" addr="137" service="netbios-ns"></port>
    <port proto="udp" addr="138" service="netbios-dgm"></port>
</host>
```

Figure 3-10. Example Host Entry

with a -x option to produce an XML file. An example host record is found in Fig 3-10. This data file is the source for communication of the network entity information to the dynamic virtual host configuration process.

3.1.3.2 Messaging

Two implementations of an IF-MAP metadata host entity message structure were pursued for use in NEI. The first followed the predefined IF-MAP network security schema and the second utilized a custom addition to the schema. This modification is allowed in the specification. A third possibility, that was not pursed, is creating an entirely new predefined metadata with any necessary custom additions.

The second implementation was necessary because two fields are not available in the predefined schema. These are transport protocol, i.e. UDP, TCP, etc. and the service name. The protocol information is important for the DHP component. As DHP is the primary internal consumer for this information, it was considered important to retain the second option.

In conclusion of this section, the Ettercap tool was selected for identifying network entities. It provides information on host IP addresses, MAC values and port usage. When compared with the five tools listed in Table 6, it performed as well or better than all of them. An additional key driving capability is Ettercap's formatted XML output that can easily be integrated into other systems. Communication within an automated system requires a defined consistent messaging system. Lastly, Ettercap is capable of performing more advanced operations that could be useful for future functional enhancements.

3.1.4 Dynamic Honeypot

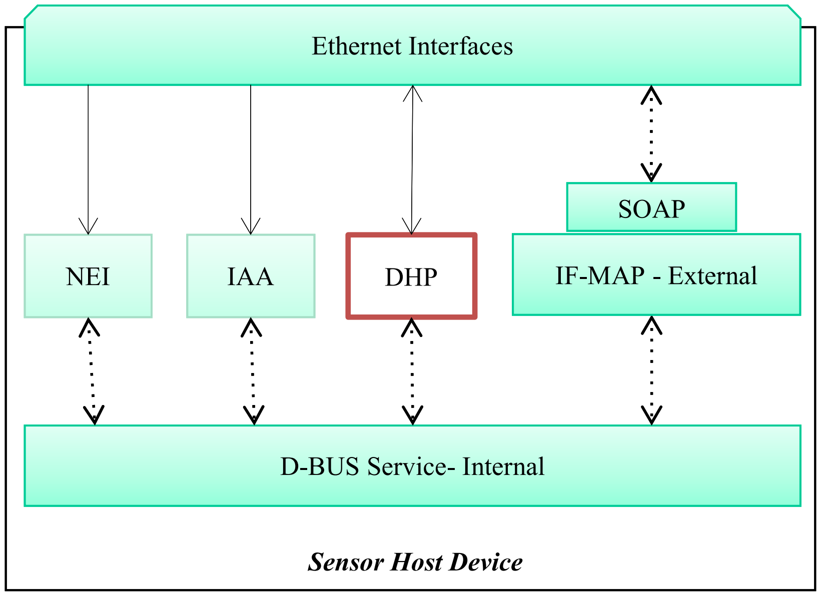


Figure 3-11. DHP Component

In this section an algorithm is proposed and demonstrated to automatically deploy deceptive virtual network entities, or Honeypots, in a control system network. The Dynamic HoneyPot (DHP) algorithm of AICS focuses on managing the complexity of self-configuration by adapting to a deployed network environment. A self-updating model of the network devices is created and maintained by passively monitoring traffic. This model is used to automatically configure deceptive virtual network entities, called honeypots, designed to draw the focus of malicious intent (Provos & Holz, 2007). Given that the NEI component is deployed on the sensor, it is a natural source for the host information required to instantiate a dynamic virtual honeypot. This information is available internally on the message bus and as an XML file. The DHP algorithm creates unique network stack signatures with information provided by the NEI component.

The conceptual architecture for the DHP module is shown in Fig. 3-12. The two primary activities consist of monitoring for host changes and dynamic updating of the emulated hosts. Passively gathered host information is retrieved from the internal message bus. For each IP address found, a comparison is made to any existing host information. First an operating system is identified. This identification may include creating an OS signature based on statistics of existing systems on the network or random generation. Next a MAC address for the virtual host is created with a correct vendor

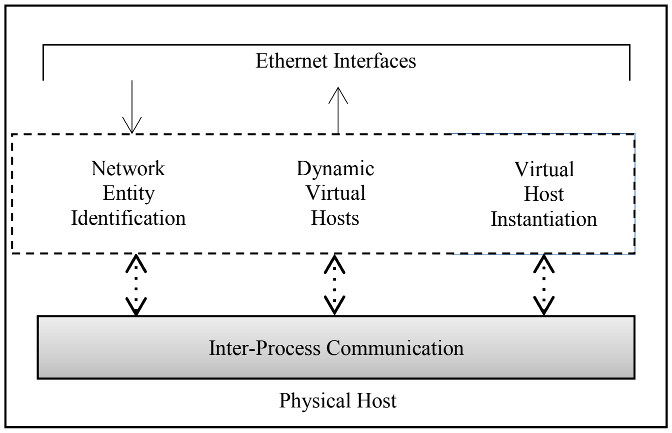

Ethernet Interfaces
Network Entity Identification
Dynamic Virtual Hosts
Virtual Host Instantiation
Inter-Process Communication
Physical Host

Figure 3-12. Conceptual Design

identification portion. Care is taken not to create duplicate MAC addresses. Device specific information files are then consulted for inclusion of typical services. Finally, the existing virtual host configurations are reviewed and updated as needed.

As part of the monitor and update process, messages are delivered internally via D-Bus providing information on the simulated hosts. This information may then be leveraged by external components to differentiate virtual hosts from real systems on the network.

3.1.4.1 DHP Background and Related Work

For the AICS solution, Honeyd was evaluated and logic created to automatically configure it. The resulting configuration is designed to emulate, as close as possible, any user identified host on the network. This is in contrast to previous work that focused primarily on dynamically creating several honeypots, called a honeynet, that are statistically similar to a network of hosts (Hieb & Graham, 2004).

Honeyd simulates the network stack and generally provides only superficial services. Because of this, an attacker is never able to gain access to the host by compromising a service but would quickly realize that something is amiss. The primary goal is not to entrap the attacker into spending all his effort on the deceptive system. It is to attract his attention, for at least a short time, and gather information that helps identify the attacker and a possibly compromised internal attack platform.

Honeypots have uses other than presenting an emulated host. For instance, gathering malware by presenting a vulnerable service, acting as a mail host to collect

SPAM email and acting as a ‘tarpit’ that consumes all attempts to break into a system. These uses are not explored in this research but are provided for completeness.

Dynamic honeypot solutions that gather network information, process that information into a configuration and deploy appropriately have been created as in (Hieb & Graham, 2004), (Hecker & Hay, 2010), (Hecker, Nance, & Hay, 2006), (Jiang & Xu, 2004). These papers propose monitoring methods that are active, passive, combined or are ambiguous. When passive monitoring is implemented, the chosen tool is typically P0f with no analysis of competing tools provided. Finally, the test implementations are all on non-control system networks.

There are two notable projects related to control system honeypots. The SCADA Honeynet project by Matthew Franz and Venkat Pothamsetty (2012) of the Cisco Critical Infrastructure Assurance Group (CIAG) was initially released in March of 2004. The project is not actively maintained, with a last release date of July 15, 2005, however it is still available from Sourceforge. The design utilizes Honeyd for simulating a set of services for a PLC. The major contributions of this project are service scripts, which include functionality for FTP, MODBUS, Telnet and a web server. However, the SCADA Honeynet does not consider automatic provisioning of the virtual hosts and is a manually configured project.

Digital Bond, Inc. is a control system security consulting and research group founded by Dale Peterson. Their SCADA Honeynet (2012) implementation is an evolution of the original project just described. It utilizes two virtual machines instead of Honeyd. One virtual machine includes network monitoring tools such as Snort with Digital Bond’s Quickdraw IDS signatures to detect activity. The other virtual machine simulates a PLC with several exposed services. There is no dynamic provisioning of hosts or services, although it is possible to replace the virtual machine PLC with an actual hardware component. This assures complete deception if the PLC is configured correctly with the added expense of an actual hardware device.

3.1.4.2 DHP Solution Design

This section describes the software tool evaluation and implementation logic of the DHP solution. Fig. 3-12 shows the relationship of three key functional areas: Network

Entity Identification (NEI), Dynamic Virtual Host configuration (DVH) and Virtual Host Instantiation (VHI). These act in a continuous cycle of processing and updating information represented by the dotted line box.

A pseudo code of the algorithm is shown in Fig. 3-13 with implementation details for procedures in italics found afterward in each section. The details of NEI are discussed in the section found prior to this one.

```
Create and update virtual hosts with following:
  Network Entity Identification.
  Write entities to XML.

  Read_data; from input files and Ettercap
  For each IP create a Dynamic Virtual Host
    Find_closest representative OS.
    Map_OS values to Honeyd names
    Create_MAC address for new hosts
    Create_Features for device specific behaviors
    Create_Config for virtual hosts
  End
```

Figure 3-13. DHP Pseudo Code

Dynamic Virtual Hosts

This section discusses the configuration creation of the Dynamic Virtual Hosts (DVH). These hosts emulate the network signature of actual systems on a physical network. Honeyd is a popular open source solution for virtual honeypots that provides a flexible and feature rich configuration capability. As autonomous configuration is a desired aspect for minimization of expensive manual configuration, Honeyd's configuration flexibility is an advantage. The overall goal is the automatic configuration and dynamic update of a variable length list of virtual hosts based on information gathered from actual hosts using Ettercap.

The following sections describe four functional areas in DVH: OS selection, OS name mapping, MAC creation and Service (port) emulation.

Operating System Selection

For any given host on a network, Ettercap may not be able to identify the operating system. If this occurs, for an emulation target, then an OS must be chosen. It is desirable to provide an exact match in network behavior. This does not necessarily require an exact match with the OS name in the database.

Read_data consists of extracting n host records h from the Ettercap entries and forming a record set O such that $O = \{h_1, h_2, \ldots, h_n\}$. O then becomes a source of information for creation of virtual hosts. The intention is to examine these records for similarities to an IP address i provided in a list IL of j target IP addresses where $IL = \{i_1, i_2, \ldots, i_j\}$. An assumption is being made that the hosts h on the network have an OS similar to a candidate i even if an exact match is not found.

Given that P_h is a set of port values for a host h and a network port set S_i for target *i, Find_closest* examines the intersection of $S_i \cap P_h$ for all h in O. The integer count of matching ports is stored for each intersection. Additionally, the number of ports for the target is calculated. Given these values, a match percentage is calculated, e.g. two candidate ports and an intersection count of two constitute a 100% match. Candidates with a higher percentage were considered to be more similar. Some OS's utilize ports specific to services offered by that OS and they could be used in identification (Gagnon & Esfandiari, 2011). Similarly several industrial control systems use specific ports not usually found in use by typical IT systems. Usage of these ports provides a clue towards the identification of the device and the possible OS.

If a candidate OS is not identified by examining ports, then the MAC address is examined. *Find_closest* compares the vendor identification section of the candidate MAC address of i to the MAC addresses for each host h in O. If a match is found that has an identified operating system, this value is placed on a candidate list. After exhaustively examining O, the largest matching value, if one exists, from the candidate list is chosen as the OS. The assumption is that if hosts on the network have the same NIC vendor they may be performing similar functions and thereby have a similar operating system. As is described later, several control system vendors have an organizationally unique MAC identifier for their network devices.

If no prior step has identified an OS, a random number r is generated in the range 0 to N where N is the cardinality (O). If the host record h_N OS field exists, this value is utilized. If not, a random value supported by Honeyd is chosen. In other words, a field is possibly selected for inclusion proportional to the relative frequency of its presence in O.

Given that not all host records contain an OS, and possibly none of them; the completely random OS value is required.

Once an OS is identified by the selection algorithm or trivially identified by Ettercap further action, as described in the next section, is still required.

Operating System Name Mapping

The Honeyd configuration value for an operating system makes use of the Nmap version 1 database defined named values. Similarly, Ettercap utilizes its own defined name values that do not directly match Nmap. To make a functional configuration, a simple algorithm implemented in *Map_OS* was developed to associate Ettercap names with Nmap names. The algorithm's initial pass compares the word tokens of the OS names looking for case insensitive string matches. The number of word matches were summed and stored. After iterating through each possible OS combination, the one with the largest count total is presented as a candidate. Finally, each OS name combination is written to a file for reference during creation of the configuration.

MAC Creation

Honeyd provides two options for specifying the MAC address, either by vendor name or the six-octet string. Because Honeyd has hard coded vendor strings, the six-octet representation was chosen for use in the algorithm. Ettercap captures this MAC octet address for all hosts in *O*. The MAC protocol specifies that the first three octets are organizationally unique and should not overlap with any other vendor. Thus, in order to create a new MAC address that appears to come from a specific vendor, these first three octets were used. The vendor typically assigns the remaining three octets. In this algorithm these last three octets are created as described next.

In the *Create_MAC* function, the last three octets are randomly generated and appended to the end of the captured candidate vendor portion. This new MAC is then compared with all other MAC's noted in the Ettercap host list *O*. Any collision of addresses instigates a recreation of another random set of values. Given the 2^{24} possible values, the probability of a collision is low. Depending upon the security configuration of a deployed switch, these generated MAC values may require more refinement. For instance, if port security is enabled on the network switch the available MAC's would have to be predefined.

Network Service Emulation

The host entries in *O* contain network ports, previously defined as P_h, that were active during the capture session. Along with the port number, a port service name is available. This service name is a human readable text value that is defined in an Ettercap configuration file called etter.services. Utilizing the service names contained in this file, a new configuration file called serv.conf was created. This file maps the service name to a service emulation script path.

The *Device_Features* function examines any service ports found in the Ettercap output and loads the serv.conf file. Any service name match to entries in the file results in the appropriate service script value placement in the Honeyd configuration. This enables the creation of service specific behaviors that furthers the goal of deception. Currently, the manual creation of scripts is necessary although some service scripts are already available from other projects.

In addition to services found during passive scanning, a variable number of ports associated with the common services are randomly activated. A common service mapping file for control system devices is utilized by the *Device_Features* function. It consists of a hierarchical MAC mapping structure. Generally, in the case of a control system device, the vendor portion of the MAC is directly tied to the device manufacturer enabling usage of the mapping file (MF) to find relevant services.

Constructed utilizing XML, the MF maps the vendor MAC to a list of common services that are possible to find activated on a device of this type. Each service in the file is described by the following attributes: port number, protocol, service description and action script. The action script specifies which script Honeyd should utilize, if any, when it sees traffic to this port. A value in this field will overwrite any previously defined default script found in serv.conf. This provides the capability to customize a response to this specific device type while still retaining generic service emulation functionality.

Each service description has an 'include' value. This is a floating-point value between 0.0 and 1.0. This value is compared to a randomly generated value in the appropriate range. If the random value is less than the include value, the port is added to the honeypot configuration. The intention is to vary port inclusion to represent the variability in device configurations.

An analysis of available vendor product specifications was used to create this file. For example, the test system contains a Rockwell Micrologix 1100 Programmable Logic Controller (PLC) and the possible services listed for this consist of Ethernet/IP, web services, SMTP email (outbound) and SNMP (Rockwell Automation, 2005).

Virtual Host Instantiation

The candidate emulation hosts are provided at startup as a list of IP addresses. It is assumed that if a host in the list disappears from passive sensing or is not initially found in the traffic, the user still desires to have an emulated version of it. The overhead to maintain the missing hosts records is minimal. Of course, the actual system has to have appeared in the passive analysis during the monitoring period to create an initial virtual host configuration that is representative. Otherwise random values are chosen until further information is found.

An initial configuration file is created by *Create_Host_Conf*. Changes to the configuration of the virtual hosts running under Honeyd are performed while the system is running. After a configurable time period, currently an arbitrarily chosen 60 seconds, etterlog is called on the ettercap daemon log file. The resulting XML output is saved and compared to an existing configuration file. Differences in network host information are noted and stored in a list for possible action. Actions include adding network services, updating OS configuration and changing MAC addresses. A companion Honeyd executable file, called Honeydctl, provides this functionality.

```
create vh1
set vh1 personality "Linux 2.4.xx"
set vh1 default tcp action reset
set vh1 default udp action reset
set vh1 default icmp action reset
add vh1 tcp port 23 "/script/router-telnet.pl"
set vh1 ethernet “00:00:BC:A1:00:23”
bind 192.168.1.125 vh1
```

Figure 3-14. Example Honeyd Configuration

A simple example Honeyd configuration file, created from the process just described, containing one virtual host configuration is shown in Fig. 3-14.

3.2 Empirical Test

This section describes an AICS test system, a test scenario and empirical results. First, the network layout and sensor hardware platform are shown. Second, a five-step testing scenario is presented. Third, each testing step and its execution is described in more detail. The description includes a performance evaluation of the relevant components and sampled IF-MAP messages.

In the following test scenario, scans and probes are directed at all devices on the network representing the reconnaissance phase of an intrusion. This assumes the attacker does not have a network map and is interested in discovering what devices are present. The goal of the security system is to generate informational alerts about the anomalous behavior from this activity. A secondary goal is the diversion of attention and effort of the attacker to a virtual honeypot system. Keys for success include: a faithful imitation of real devices on the network, a mechanism for monitoring activity directed at the hosts, and appropriate communication of host information and alerts.

The test deployment is an operational control system network with a heterogeneous mixture of devices. There are two possibilities for timing when the honeypots are instantiated to emulate a subset of the devices. The first approach, used in this test scenario, is to create the virtual hosts in advance of any anomalous occurrences. This would increase the probability of a network scan identifying the hosts. It removes the race condition between recognizing an anomaly and getting the hosts instantiated in time to be noticed. One weakness of this approach is the initial reduced fidelity of emulation by the virtual hosts. Unless the NEI component has been run in advance there will be little information about the actual hosts on the system. However as NEI gathers more complete information the virtual hosts are reconfigured to reflect the updated knowledge.

The second approach, with the race condition, would be to instantiate the hosts after an indication of intrusion has occurred. This indicator could come from a traditional intrusion detection system or from the IAA component. Given the DHP use of virtual hosts with its reduced hardware requirements, a dedicated integrated host and low network impact; there is little benefit to delaying instantiation until after detection.

3.2.1 Test Network

At the beginning of the scenario, all real devices are running and an AICS sensor host is connected to the control network as seen in Fig. 3-15. As the NEI component becomes aware of changes in the host characteristics, the honeypots are automatically reconfigured to include the new behavior. The emulated hosts become more authentic appearing, in the service ports offered, over time. As already mentioned, this early instantiation reduces the risk of a stealthy intruder bypassing the honeypots, as they will most likely be present prior to the malicious activity.

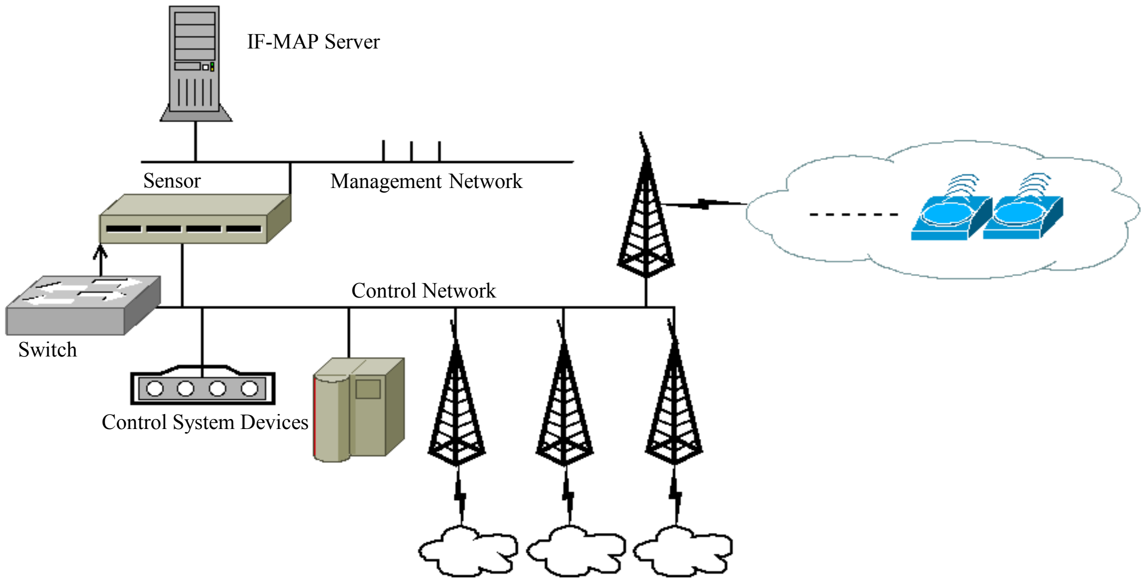


Figure 3-15. Test Network Diagram

A small campus grid (SCG) and sensor network that physically resides at the Center for Advance Energy Studies in Idaho Falls, Idaho was used as a test platform. The network consists of a heterogeneous mixture of 46 directly connected devices, including wireless sensors monitoring environmental conditions in the building. The SCG is connected to a small wind turbine, a solar power station, a wireless Advanced Metering Infrastructure (AMI) with two wireless access points (WAP) and a variety of control system devices. Twelve smart meters, which wirelessly connect to the WAP's, are physically attached to test harnesses to generate a signal. The network has several Windows based computers, web camera's, a Rockwell Automation Micrologix 1100 PLC and a National Instruments PLC. The Micrologix controls or monitors 6 lighted push buttons, 7 lights, 2 potentio-meters, 2 temperature sensors and a small electric fan which constitutes both digital and analog input/output points. All of the network hosts are part of a single network segment directly attached to a Cisco network switch with a SPAN port. The SPAN port receives a copy of all network traffic and is used for monitoring by AICS. This network segment is labeled in Fig. 3-15 as the Control Network.

The SCG includes access to three wireless sensor networks from the following commercial vendors: Emerson, Honeywell and Arch Rock. Each one of these three systems connects to wireless environmental sensors through one or more wireless access points. As with the previously mentioned AMI deployment, the WAP gateways have both a wired network connection and a wireless interface to the remote sensors. The multitude of environmental sensors is not directly visible to the wired test network. However the wired side of the WAP communication is visible to the AICS device. Several data historian hosts attach to the WAP's on the wired side and retrieve the environmental data using the appropriate protocol. Each vendor has a unique implementation of a wired network protocol built on top of Ethernet.

The implemented AICS software was deployed on a test hardware platform. This platform runs a 32 bit Ubuntu 12.04 LTS OS on a recently released Via AMOS-5002 fanless compact embedded system. It consists of a VIA dual core EdenX2 (U4200) CPU, 4 GB's of DDR3 1333MHz RAM, a 320GB SATA II hard drive and 3 Gb Ethernet ports. One of the Ethernet ports, indicated in Fig. 3-15 as an arrowless line between the sensor and control network, was assigned to the dynamic virtual honeypot for emulation of network hosts on the control network. A second interface was dedicated to passive monitoring on the switch SPAN port by the anomaly detection and entity identification modules. The third interface was utilized to provide external IF-MAP communications on the management network for the sensor. The management network is a separate network segment from the Control Network. Its use is to convey messages between the AICS device and other management components without interfering with the operational network. The AICS host hardware was chosen for its flexibility of deployment in industrial/commercial environments and a relatively strong computational capability.

3.2.2 Test Scenario

The steps in Fig. 3-16 provide a scenario devised to test the communication and managed resource functionality of AICS when deployed on the SCG network. Each step provides a high level action followed by the primary functionality to be evaluated. In general, verification is accomplished by inspection of IF-MAP messages or local component data stores. The results of the verification and other related information is provided in the evaluation section following the test layout.

Step 1: Attach sensor to SCG network and initiate system.
- Evaluate NEI host identification model, dynamic updates and related IF-MAP messages.

Step 2: Send IP lists to DHP Component
- Test dynamic virtual host creation based on IP list and utilization of NEI host information from step 1.

Step 3: Target Network probes at DHP hosts.
- Verify emulated network presence of devices.

Step 4: Send IP list to IAA anomaly behavior monitoring.
- IAA initiates learning mode, creates normalization and clustering values for fuzzy rule creation.

Step 5: Send monitor message to IAA and initiate network attacks.
- IAA recognizes attacks and sends IF-MAP alerts.

Figure 3-16. AICS Test Scenario

A C++ and PERL implementation of the AICS framework was run on the test sensor platform attached to the operational test network. Additionally, two test tools were created to facilitate execution of the scenario. Tool one provides the input IP addresses to the MAP server for subsequent delivery to both the DHP and IAA. Tool two monitors the MAP server for published messages from the sensor.

During execution of the scenario, traffic from the control network was captured on the switch's SPAN port and preserved in a pcap formatted file. The captured data provides a consistent baseline for repeat system evaluation. In the presented scenario and results, the sensor was attached to the live system unless otherwise indicated. The preserved traffic was subsequently used to perform evaluations on the AICS sensor. The test results from the system, using this stored data, produced no discernable differences with those found in the live scenario. In other words, the results are repeatable and deterministic.

3.2.3 Scenario Execution and Results

STEP 1- Attach sensor to network and power on.

As the sensor started to recognize hosts on the network the NEI component created its internal model and produced host messages. The internal host model is stored as an XML file and made available to the DHP component through messaging. Updated host messages are published to the MAP service every 60 minutes if a change occurs. During the complete test cycle, 45 of the 46 devices were identified by NEI. The single unidentified host used a custom protocol that does not utilize IP addresses and consequently was not recognized by NEI. This host is the Honeywell wireless access point. It communicates using raw Ethernet frames and is characteristic of a Honeywell proprietary protocol. Because it does not utilize an IP address for communication, Honeyd cannot emulate this device.

A sample IF-MAP NEI host message captured during step 1, slightly modified for space, follows:

```
<metadata>
<meta:host ifmap-cardinality="multiValue" ifmap-publisher-id="test--137625522-1"
ifmap-timestamp="2012-03-26T14:40:33-06:00">
        <start-time>2012-03-26T14:40:00</start-time>
        <mac>00:12:3F:61:4A:69</mac>
        <ip>192.168.99.220</ip>
        <os>D-Link DWL-900AP</os>
        <port_list>
                <port proto="udp">137</port>
                <port proto="udp">138</port>
        </port_list>
</meta:host>
</metadata>
```

Figure 3-17. NEI IF-MAP host message

STEP 2 – Send IP lists to DHP component.

A PERL implementation of the DHP algorithm was run on the test sensor platform. Twelve heterogeneous systems, shown in the second column of Table 8, were evaluated. The ID column is used as a reference identifier and corresponds to the last octet used in the emulated IP address. These hosts represent a variety of equipment found on the test network including network switches, PLC's and WAP's. Using tool one, an IF-MAP message containing the twelve host IP addresses was published to the IF-MAP server. AICS retrieved the list and delivered the information to DHP. An example message with three hosts is shown in Fig. 3-18. The format is a list of tuples with the R= line indicating the real host to emulate and the E= line designating the IP to use for the corresponding virtual host. The IP's in the message are used as a key to lookup host information published from the NEI internal model.

```
R = 192.168.1.1 192.168.1.3 192.168.1.10
E = 10.10.10.1 10.10.10.3 10.10.10.10
```

Figure 3-18. Sample IP monitor list

The DHP component, using the NEI information, was able to automatically create virtual hosts for all twelve of the network-attached devices. Each emulated host was assigned its own unique IP and MAC address and was instantiated on the test sensor hardware. Furthermore, as the NEI component became cognizant of more host details, each related virtual device managed by DHP was updated to reflect the new information. For example, when a host started communicating on a new port the virtual host was automatically reconfigured to emulate it.

Table 8. Host Identification Results

ID	Device	Mapped OS
1	Rockwell HMI	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
2	Micrologix 1100 PLC	Novell NetWare 3.12 - 5.00
3	Arch Rock Server	Random
5	Honeywell HMI	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
10	Arch Rock WAP	Random
25	D-Link WAP	Apple Airport Extreme Base Station (WAP)
99	D-Link Wireless camera	Apple Airport Extreme Base Station (WAP)
130	Arch Rock HMI	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
150	Nat. Inst. PLC	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
200	Emerson WiHart AP	Linux 2.4.16 - 2.4.18
215	HMI(Windows PC)	Windows for Workgroups 3.11 / TCP/IP-32 3.11b stack or Windows 98
253	Moxa 505A Switch	FreeBSD 4.4 for i386 (IA-32)

Twelve systems shown in the first column of Table 8 were evaluated; six control devices and seven more typical information technology devices. The ID column is used as a reference identifier and corresponds to the last octet used in the emulated IP address. Of the twelve devices chosen for emulation, ten operating systems were configured autonomously and two were assigned as ‘random’. If NEI is not able to identify an OS, then DHP randomly choses a value from the list of identified OS’s in the host model file. The assumption is that devices on a network are likely to have OS’s similar to each other. Another option is to randomly pick one of the thousand OS’s defined in Honeyd. The third column in Table 8 shows the mapped OS names selected by the algorithm. Host IF-MAP messages similar to those in Step 1, only with a different identifier to differentiate real from emulated hosts, were created and published to the MAP server for the twelve emulated hosts.

STEP 3 – Target network probes at 12 DHP emulated hosts.

This step verified, from a network perspective, that the emulated hosts exist and at a superficial level appear as separate, diverse devices. In addition to the OS emulation performance exercised in Step 2, seven network test probes with Nmap were completed. Nmap is a network scanning tool that is commonly used to find attached hosts, scan ports and identify operating systems (Lyon, 2008). The four probes included OS detection, IP protocol evaluation, ICMP echo 'pings' and UDP/TCP port scanning. Nmap version 5.21 was chosen to test the network presence of the emulated devices. This version utilizes the second generation Nmap OS database that is actively maintained. It uses a more robust guessing implementation for uncertain signatures. Additional improvements include probe changes, recognition algorithms and performance. A laptop, with Nmap and ping installed, was assigned the IP address 192.168.1.15 and attached to the SCG network. The laptop filled the role of network intruder and was the source of the network probes. The seven tests are described in detail next.

STEP 3 - Test 1: nmap -n -sP 10.10.10.0/24

This simple test performs a 'ping sweep' on all 256 addresses in the range that contains the 12 emulated devices. A combination of an ICMP echo request, TCP SYN to port 443, TCP ACK to port 80 and ICMP timestamp request are sent. Any system that responds to one of these requests is considered available on the network. All twelve of the emulated addresses were found in 2.2 seconds.

STEP 3 - Test 2: nmap -n -v -A -T4 -iL nmap.testhosts

This command line is the first example provided in the Nmap man page documentation. The –A option enables aggressive scan options including OS detection, version scanning, script scanning and traceroute. The –T4 option is a timing template that improves scan time on reasonably stable networks. Note, that by default, Nmap only scans 1000 of the most commonly used ports. It completed in 234 seconds.

OS detection in Nmap is based on a database of signatures. Each fingerprint record in the database contains four fields: vendor, OS family, OS generation and device type. Output from detection includes lists of possible operating systems and device classes with an accuracy score. The score falls in a range of 0.0 to 1.0 with the later indicating a perfect match.

The OS detection produces large amounts information. For the 12 emulated devices, 223 device types and 40 OS matches were returned. In both cases, accuracy ranged from 85% to 97%. As there were multiple results for most of the emulated devices, any of the entries that matched either the original device or its mapped OS were considered a success. Of the ten non-random devices, Nmap correctly identified seven for a 70% success rate. Of the three that failed, no information was produced for device 2. Twenty-one incorrect entries were created for device 215. Device 5 was identified by one incorrect entry.

STEP 3 - Test 3: nmap -sU -sS -O --osscan-guess -n -p1-65535

The –sU option executes a UDP scan to each port specified. For some common ports a protocol specific payload is included but for most of them the packet is empty. The –sS option tells Nmap to send only a single SYN packet to each port. This is the initial packet sent in a TCP connect sequence. The –O option enables standard OS guessing while –osscan-guess makes Nmap guess more aggressively. Finally, the –p argument specifies to scan all possible ports instead of the default top 1000. The tool took 258 seconds to complete the configured actions.

The results from this Nmap execution were similar to those in Test 2 with some exceptions. First the correct OS guess for device 253 increased in accuracy by 2 points. Second device 215 was correctly identified with an accuracy score of .86 were previously it had failed. This increased the overall identification rate to 80%. It should be noted that, because of the broad port scan range, port 44818 for device 2 was found. This port was missed in Test 2. This is a common port used by the Rockwell Ethernet/IP protocol that is specific to that control system implementation.

STEP 3 - Test 4: nmap –sO –n

This scan sends IP packets and iterates through the eight-bit IP protocol field. The emulated hosts responded to only three of the 256 protocols: ICMP, TCP and UDP.

STEP 3 - Tests 5-7: ping -c2 –R, ping -c2 -T tsonly, ping -c2 -T tsandaddr

Utilizing the ping command line tool, ICMP echo requests were sent to the 12 emulated and 46 actual devices on the test network. ICMP packets are wrapped in an IP datagram and can contain IP option fields. Three rounds of requests were sent, one with the Record Route (-R) option, one with timestamp only (tsonly) and finally the option with both IP and timestamp (tsandaddr). All but one of the physical devices responded with varying levels of correctness to the pings. None of the 12 emulated devices responded correctly. The results for tests 4-7 are discussed more fully in the evaluation section.

STEP 4 – Send IP List and monitoring request to IAA.

Three IP addresses were selected for monitoring by a C++ implementation of the IAA component. A message containing the IP's was sent to the MAP server with tool one. IAA retrieved this message and started passively monitoring for the appropriate network traffic. For evaluation purposes, the training mode was constrained to contain 100,000 packets recorded during normal network usage. This training occurred prior to the network scans in step 3. An isolated network was maintained to ensure the normality of the recorded data.

The configuration of the IAA fuzzy logic based anomaly detection routine is maintained in an XML file. This allows for exploring the effect of changing key parameters for clustering, fuzzy rules, network packet windows and more. Prior work exploring these variables can be found in (Linda O. , Vollmer, Wright, & Manic, 2011), (Linda, Vollmer, & Manic, 2012). The values used for the test scenario are provided in Table 9. Seventeen features were derived from a twenty packet network buffer. These

Table 9. IAA Configuration Values

Configuration Item	*Value*
Data Window Length	20
Data Dimension Input	18
Data Dimension Output	17
Data Normalization Input Length	1000
Normalize Data	Yes
Cluster Spread	2.0
Cluster Blur	0.2
Maximum Cluster Radius	0.25

features were normalized and clustered using an online nearest neighbor algorithm with a radius of 0.2. The resulting clusters, for the three individual IP communication streams, were composed of 19, 57 and 2 cluster members. These cluster sizes reflect the heterogeneous network behaviors of the different devices.

STEP 5 – Send monitor message and initiate network attacks.

Following the collection of training information in the previous step, 200,000 packets with abnormal behavior comingled with normal behavior were evaluated. This set of data included the Nmap tests that were run during the DHP testing in Step 3.

The Nmap and Nessus software applications were used to generate anomalous network traffic behavior. Nessus provides auditing capabilities, system vulnerability assessments, and profiling information (Nessus webpage, 2012). The simulated intrusion attempts included but were not limited to: ARP pings, SYN stealth scans, port scanning and control system specific device probes. Cyber attacks ranged from long attacks composed of many packets to short attempts of single packets.

Table 10. Anomaly Performance

Threshold	*Correct Rate*	*False Pos.*	*False Neg.*
STREAM 1			
0.3	99.8539%	0.1461%	0.0000%
0.6	99.8705%	0.1295%	0.0000%
0.9	99.8788%	0.1212%	0.0000%
IT2 FLS	99.8722%	0.1278%	0.0000%
STREAM 2			
0.3	99.9037%	0.1217%	0.0275%
0.6	99.5504%	0.1082%	1.3753%
0.9	99.3799%	0.1082%	2.0079%
IT2 FLS	99.9111%	0.1116%	0.0275%
STREAM 3			
0.3	99.8643%	0.2953%	0.0000%
0.6	99.8960%	0.2265%	0.0000%
0.9	99.8960%	0.2265%	0.0000%
IT2 FLS	99.8960%	0.2265%	0.0000%
STREAM 4			
0.3	99.9025%	0.0975%	0.0019%
0.6	99.8705%	0.0985%	0.3538%
0.9	99.8788%	0.0712%	0.9091%
IT2 FLS	99.8722%	0.1278%	0.0025%
STREAM 5			
0.3	93.5655%	5.002%	6.7257%
0.6	93.7096%	4.9295%	6.8538%
0.9	93.7088%	4.9294%	6.9091%
IT2 FLS	93.7092%	4.9295%	6.7330%

Table 10 depicts the results of the anomaly detection for five streams. The three rows labeled 0.3, 0.6 and 0.9 show the performance with a fixed sensitivity threshold value. The IT2 row provides the performance with domain knowledge encoded in the form of fuzzy rules performing dynamic adjustment of the sensitivity threshold. The correct classification, the false negative and the false positive rates were used as performance measures. The correct classification rate is the percentage of correctly categorized data packet instances. The false negative rate is the ratio of incorrectly labeled normal behavior inputs and the overall number of normal behavior instances. The false positive rate is the ratio of incorrectly labeled anomalous inputs and the overall number of anomalies.

An example asynchronous IF-MAP message describing abnormal behavior is shown in Fig. 3-19. This message provides the device name of the sensor and describes an anomaly found between two IP addresses. The .220 address is one of the three being monitored and the .10 was the origination of the attacker. The proto_list shows that the 20 packet window contained UDP and ICMP protocol packets.

```
<notify>
<device><name>CS:ID1:002</name></device>
<metadata>
        <event ifmap-cardinality="multiValue">
        <disc-time>2012-03-26T14:50:00</disc-time>
        <confidence>1.0</confidence>
        <type>behavior</type>
        <window-size>20</window-size>
        <ip_list>
                <ip>192.168.99.220</ip>
                <ip>192.168.99.10</ip>
        </ip_list>
        <proto_list>
                <proto>1</proto>
                <proto>3</proto>
        </proto_list>
        </event>
</metadata>
</notify>
```

Figure 3-19. IAA IF-MAP Anomaly Message

3.3 AICS Evaluation

This section describes the performance of the AICS implementation. First it provides hardware and software performance numbers that were monitored during test execution. Second the hardware and system OS issues are discussed. Finally some limitations and issues with the software tools are explored.

3.3.1 Hardware and Software Performance

The CPU and RAM utilization was monitored during the test scenario. As the hardware platform is a fanless system, temperature data was recorded as well. The 200,000 anomaly evaluation packets contained 19,627,063 bytes of payload information. Approximately 97 packets per second were processed at normal network transmission speed. It was observed that each anomaly alert sent from IAA on the IF-MAP mechanism incurred an overhead of .7ms of communication processing time.

The CPU thermal sensors were polled every 5 seconds. Room temperature at the start of the test was 23C. Core 0 temperature after a 5-minute warm-up period was 44C. Values during the test ranged from 42C to 47C. Core 1 temperature during the warm-up period was 45C. The test values ranged from 43C to 48C. It is speculated that the difference in the range low value and the initial start might be due to variations in room temperature during the test.

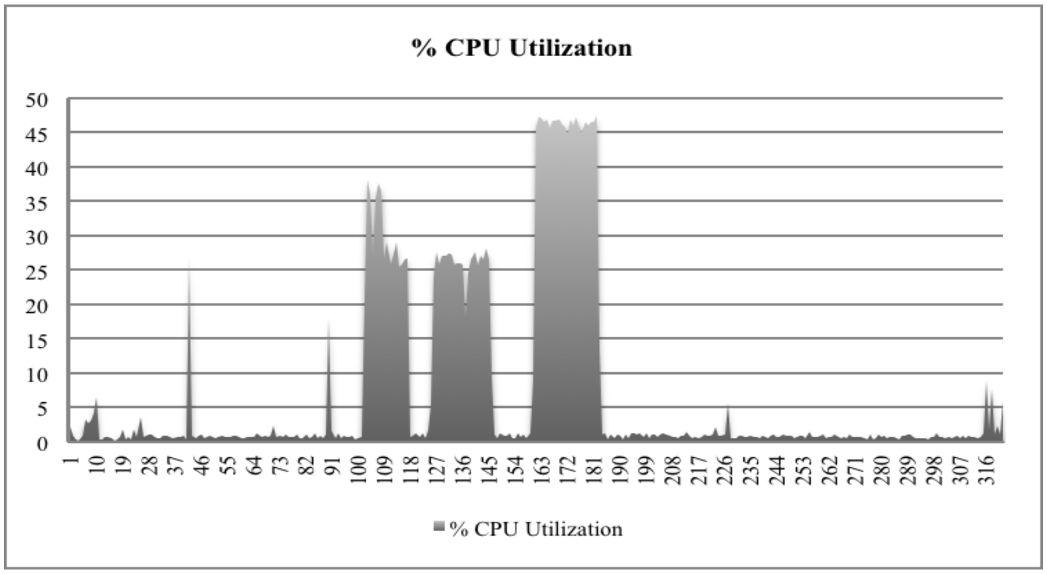


Figure 3-20. Test system CPU utilization

The at-rest CPU utilization, as measured by the Linux top command on one-second intervals, was 1.1%. This was due mainly to the Xorg process providing HMI services. In a production release services of this type would be disabled. The utilization maximum during the test was 47.4% with an average of .65% and standard deviation of 2.29%. The maximum occurred when the virtual honeypots were probed with Nmap. The graph in Fig. 3-20 shows the sampled CPU utilization before and after the maximum that occurred between time index 100 and 181. Another key performance indicator for Linux machines is the amount of time the CPU has been waiting for I/O to complete. The maximum value was 10% with an average of 0.59% and standard deviation of 0.62%. In other words, the machine was not overwhelmed with the test data when running in real time.

System memory on the sensor platform at the beginning of the test, prior to initiating network activity, was 801,516 kilobytes. Peak overall usage was 881,308 kilobytes. DHP related processes consumed a steady 20 megabytes of information. Internal messaging averaged 4 megabytes. IAA related processes used 13 megabytes. Finally, the NEI components utilized a consistent 36 megabytes of memory. Overall continued growth of sensor memory usage was attributed to non-sensor related processes such as Xorg. Turning off unneeded services, such as graphic management, would reduce the memory footprint.

3.3.2 Hardware and OS Evaluation

Scalability of the AICS solution relies primarily on the capability of the hardware host. Honeyd is technically capable of emulation 65,535 hosts. Testing by the Honeyd authors shows that, on a modest system, thousands of different honeypots are possible (Provos & Holz, 2007). To validate this claim, a test with 986 unique virtual hosts was run on the test platform. The Honeyd OS signature database contains 986 entries.

The Nmap command in Test 1 was executed targeting the 986 IP's. The top command was run on a 1 second interval to capture CPU and memory usage of the Honeyd daemon. At rest, prior to the Nmap tests, 8,748 KB's of memory was consumed. 8,860 KB's were used at the conclusion of the test. The average CPU utilization was 0.3% with a standard deviation of 1.23% and a maximum of 14.9%. This testing is not comprehensive but does validate that, at a superficial level; a large number of virtual hosts can be created on the test hardware. Honeyd is single threaded and with more intensive probing it is possible to maximize utilization of a single CPU. The test system has two CPU's and can continue to function even if this occurs.

The sensor host device setup requires physically connecting the device to the networks. There are three physical copper network ports on the device. Eth0 was connected to the network switch SPAN port in read only mode. The NEI and IAA components shared this monitoring port. Eth1 was connected to the management network for read and write. This port was utilized for IF-MAP communications. Eth2 was connected to the operation network and assigned for use by the virtual honeypots. The average poll cycle of the wireless monitoring systems occurred once every minute. Playback of captured test data at rates from 0.6 Mb/s to 20 Mb/s showed no discernable difference in the AICS average detection rate. The first observable accuracy issue occurs when the physical network port capacity is overrun.

The Honeyd instance running on Eth2 is the primary network threat to the AICS host. This is a natural outcome as honeypots are designed to attract attention. This threat is partially mitigated by the design of Honeyd. The software runs as a restricted user and, by default, does not provide any real services to compromise. For instance, on a high-interaction honeypot there are real shell services that might be compromised. Note that this does not rule out a denial of service or exploitation of a flaw found in Honeyd itself. In addition to the Honeyd features, a host monitoring system such as OSSEC (Hay, CId, & Bray, 2008) can be utilized to provide self-monitoring.

AICS was developed as a deployable modular framework on a commonly available hardware platform with components that provide network host information, identify anomalous network traffic behaviors and deploy dynamic virtual honeypots. In the test scenario, the sensor was deployed on a self-contained hardware platform. This effectively hides the communication between components from other network entities, thereby adding another layer of defense. When considering the addition of information from external hosts on the IF-MAP interface, a balance between security and information access would need to be evaluated.

The test hardware host uses a Long Term Support (LTS) version of Ubuntu 12.04. This OS has a five-year support cycle that includes security upgrades. As part of the hardware design, three physical Ethernet ports were specified. The ports are all assigned to a specific communication task to avoid a complete denial of network service on the sensor. For instance, if a large number of honeypots are active and consuming the entire bandwidth of a single port the system can still communicate on another port assigned to the management network.

Finally, it is not required that Honeyd and the anomaly behavior routines reside on the same machine. However, by condensing the software installs to one platform, it simplifies configuration. It creates a more secure mechanism for passing messages, as the information never leaves the machine. This also provides an opportunity to explore the recently expanding computational capabilities of low power multi-CPU devices.

3.3.3 Tool Limitations Evaluation

This section describes issues found with the tools used to implement the automatic configuration algorithm primarily Honeyd. They are provided as findings relevant to the specific tools and are not detractors directly related to performance of the AICS algorithm. The deficiencies found are special cases of characteristics that are not commonly examined. A review of literature has found similar types of weaknesses in other honeypot implementations so this is not necessarily unique to Honeyd (Mukkamal, Yendrapalli, Basnet, Shankarapani, & Sung, 2007).

Examination of the emulated test systems, using the Nmap protocol scan, revealed a Honeyd limitation. As was noted in Test 4, the emulated hosts only responded to three protocols. When run against real devices in the test network, a variety of responses are noted. This includes a varying number of protocols acknowledged. A review of the Honeyd source code reveals that basic support for other protocols could be added.

An issue with the handling of the IP options field was discovered with Honeyd in tests 5-7. The IP datagram format consists of a header, option and data sections (Internet Protocol, 1981). The option section is a variable length list of options, up to 40 bytes, that is not typically used. Of the five currently defined options, two are relevant to this project: Record Route and Timestamp. Record Route requests that the target, and each hop on the path to it, add their IP to a list in the option field. Timestamp has three request variations: timestamp only, timestamp with IP and a preloaded IP list. Honeyd does not support any of the options.

As this field in the IP header is optional, support by vendors vary from none to dropping packets that contain options. A study done in 2005 found a 45% success rate for Record Route and a 36% success rate for the Timestamp option when implemented in a SYN packet sent to web servers located on the Internet (Fonseca, Porter, Katz, Shenker, & Stoica, 2005). Another study, performed in 2010, on 267,736 Internet addresses found a 47.7% response rate to Timestamp requests delivered in an ICMP Echo Request (Sherry, 2010). These studies show that, despite being optional, a significant number of devices provide some level of support and therefore make it a concern for emulation. Honeyd is built with a library named libdnet. This library has the requisite functionality to correct the issues noted.

Honeyd uses an older version of the Nmap database scheme. There are two primary drawbacks to this condition. First, the fingerprints are not updated and are missing more modern signatures. Second, there is very little control of the emulation behavior outside of the signature definitions. Effectively this means emulation is dependent almost entirely on the definitions. A better solution might be a melding of a historical signature with observed characteristics found in the live network traffic and upgrading to the latest Nmap version 2 formatted database.

Finally, as was noted earlier, a host on the network used a custom protocol that did not utilize IP addresses and consequently was not recognized by Ettercap. The host communicates using raw Ethernet frames and is characteristic of a Honeywell proprietary protocol. However Ntop did notice the host communication and tracked the host in its node list.

The choices are to choose Ettercap, Ntop or merge data from both sources of information. A weakness with Ntop is in the data export routine. The interface does not contain all the information needed to create a configuration. Ettercap was chosen based on its XML output functionality and general recognition performance. It is feasible to correct the Ntop programmatic interface to provide all necessary information as it is an open source project and it internally tracks the data. Likewise, Ettercap could be modified to track host information on IP's or MAC addresses.

3.3.4 Aggregation Approach to Minimize False Alerts

To enhance the cyber security of a network system various approaches can be applied. One approach, utilized by the AICS IAA module is anomaly detection. An anomaly detection system is trained on a set of 'normal' network behavior. The extracted behavior model is then used to detect anomalous behavior in newly observed testing data. The system first identifies individual communication streams in the overall network traffic and then individually applies a developed network security cyber-sensor algorithm to selected streams. This approach allows for learning accurate normal behavior models specific to each network communication stream. In addition, an Interval Type-2 Fuzzy Logic System (IT2 FLS) is used to model human background knowledge about the network system and to dynamically adjust the sensitivity threshold of the anomaly detection algorithms. The IT2 FLS is used to model the linguistic uncertainty in describing the relationship between various network communication attributes and the possibility of a cyber attack.

The proposed anomaly detection system was implemented and tested on an experimental control system test-bed. It was demonstrated that the system does learn normal behavior models for each selected communication stream and performs accurate anomaly detection. In addition, it was also demonstrated that the availability of domain knowledge can improve the performance of the anomaly detection method. Table 10 in section 3.2.3 shows results from monitoring two different hosts on the network while testing with intrusive activity. For comparison purposes the first three values in the Threshold column represent a fixed sensitivity value. The IT2 FLS row depicts the results of a sensitivity value that changes according to the human created logic rules. At first glance, a correct identification rate of 99.8722% and 99.9111% looks good. However, on the test network with 46 directly connected devices 635,560 packets were captured over a 5 hour interval. If it is assumed that the performance monitoring of stream 1 and 2 are representative of monitoring all devices then 813 and 565 packets, respectively, would be misidentified. This equates to 813 false positives or alerts from the stream 1 performance. For stream 2 performance this means 462 alerts and 103 missed malicious packets. From a human operator perspective this number of alerts is relatively high in a 5 hour time frame.

An anomaly alerts occur for each packet that is identified as not belonging to a normal cluster. This makes for a possibly large amount of alerts depending upon the traffic seen. It was observed in the test scenarios that anomalous traffic tends to occur in bursts of continuous packets. A mechanism was implemented to aggregate related alerts together and result in a single alert. This has several desirable effects. The first is the reduction of alert traffic for the same incident. The second is the possible reduction of false positives. The false positive traffic has also been observed to occur in continuous segments. This type of mechanism would not repair the recognition mechanism but would effectively filter the false positive reporting and reduce the volume sent to a consumer of alerts.

Log::Log4perl is a Perl port of the popular log4j logging package. It allows for effectively controlling the amount of logging messages generated. Logging levels can be changed during the running of a process. Additionally messages can be redirected to multiple outputs such as a file, email or database. Extensive logging functionality can change without having to modify a programs source code. The AICS messaging component utilizes a standard centralized interface based on PERL. Therefore Log4perl is a natural candidate to implement both syslog capability and managing alert message volume.

Log4perl has a concept called appenders. Appenders define logic for which output devices the log data is to be written. There are default appenders provided as part of the distribution but custom logic can be implemented as well. This later functionality was utilized by the AICS project. A custom appender was developed to aggregate alerts from the anomaly detection routine. The appender consumes an alert as it is created and stores the information in an internal buffer. Any subsequent alerts are then examined for similarity to the buffered alert. If it is identical in IP addresses, port numbers and IP protocol then a counter value is incremented. If these criteria are not met then the buffered alert is released and the current alert is buffered. This has the effect of reducing the amount of alerts sent from the sensor device. In addition to a change in alert values, if the incremental counter reaches a configurable limit the message will be sent. A trivial enhancement to the appender would be to add a default time limit on how long alerts are held. This would ensure timely delivery of alerts if the monitored network has few anomalies. In addition to the aggregation logic a syslog output capability was added to the appender.

The test scenario in section 3.2.2, when run initially without aggregation, produced 178,191 alerts to the system syslog file. Of these alerts 1,549 messages were false positives. The aggregate appender was enabled with an increment count maximum of 20. The total number of alerts was reduced to 29,270 and the false positive message totaled 845. These messages included the criteria mentioned previously with the addition of a confidence value extracted from the alert. The 20 count maximum test was run without considering the confidence value in the message similarity logic. The total number of alerts from this change was 17,631. The number of false positives was reduced to 352. These simple measures reduced the false positive rate to 23% of the original. An example message is shown Fig. 3-21. The 12 in brackets is the value of the increment counter for the message.

```
2013/12/10 13:31:33 INFO [12]: CS:ID1:001:IAA:1:anomaly:20:192.168.99.135
192.168.99.10 192.168.99.206 192.168.99.140:6:43.52124 -112.05260
```

Figure 3-21. Alert Message

3.4 Comparison to Existing Work

An anomaly detection project from (Dussel, Gehl, Laskov, Buber, Stormann, & Kastner, 2010) relies on the computation of similarity between transport-layer packet payloads embedded in a geometric space. They performed experiments on traffic from a SCADA test bed containing various application-layer protocols. In contrast to AICS use of header statistics derived from a sliding window, the project used packet data content to create *n*-gram populated feature vectors. This is a common technique utilized in document comparison routines. A simple distance measure is then compared with stored 'normal' vectors to determine if an anomaly exists in the traffic. The AICS IAA algorithm maintains two models of system behavior: a global threat model that considers all traffic and a set of individual models for a given stream. Their solution appears to broadly apply itself to all traffic. It is difficult to compare results with different test data sets but their average detection rate of 88%-92% at a false positive level of 0.2% may be due to the differences just mentioned.

Another approach taken in (Macia-Perez, Mora-Gimeno, Marcos-Jorquera, GilMartinez-Abarca, Ramos-Morillo, & Lorenzo-Fonseca, 2011) is similar to the AICS architecture. The design incorporates a self-organizing map (SOM) anomaly detection routine and SOAP communication. The routine is embedded on a small sized low power MOXA device with two wired Ethernet ports. This implementation is comparable to work done at the genesis of the AICS solution (Linda O. , Vollmer, Wright, & Manic, 2011). A binary XML (MTOM) version of SOAP is used for eternal messaging. The detection rate is apparently affected by the limited hardware. As network traffic increases the anomaly detection rate decreases. For example, the detection rate is 80% with a 6 Mb/s network load. It was not clear if this was a result of the feature extraction algorithm or a loss of packets captured. In addition to the different computational intelligence routines utilized, the AICS algorithm is effectively an evolution of this design that takes advantage of improved hardware. Furthermore, AICS is able to provide complementary services such as virtual honeypots (DHP) and host identification (NEI).

The use of IF-MAP by AICS is an additional improvement over the previously mentioned designs. IF-MAP provides a predefined data dictionary for network security information that eases integration efforts with other components. The modular nature and common communications infrastructure provides a flexible platform for changing functionality. The AICS solution delivers observed information via a published interface for integration with a system wide solution. While the components are self-configuring, the data used to perform that configuration comes from passive observations. Because of the IF-Map communication scheme, host information could be provided by external entities. For instance, information about host entities could be published on the IF-MAP SOAP interface. This broadens the potential use of AICS in a more sophisticated hierarchical security system.

An additional beneficial outcome of the presented solution is removal of the configuration burden from the human operator. This capability stems from the self-configuring aspects of an Autonomic design. For example, the IAA anomaly detection routine doesn’t require human created rules. It learns normal behavior from observations of the system. Additionally the dynamic honeypots are configured from system observations. This functionality reduces dependence on network expertise and level of human configuration effort.

3.5 Chapter Summary

Inspired by self-configuring aspects of Autonomic computing, the work presented here supports the important goal of securing industrial ecosystems by providing network security awareness in a heterogeneous control system network. Contributions of this chapter include: 1) A flexible two level communication layer based on Autonomic computing and Service Oriented Architecture. 2) Description of a module that utilizes clustering and fuzzy logic to monitor traffic for abnormal behavior. 3) Exploration of tools and characteristics appropriate to passively monitor network traffic and identify network hosts. 4) An algorithm to deploy deceptive virtual network hosts utilizing the AICS communication system.

As can be seen in the architecture and test sections, the self-configuration capability of Autonomic systems is exemplified by the IAA and DHP components. The DHP host entities are created automatically based upon passive monitoring of network activity. Anomaly behavior in IAA is detected by clustering of information retrieved from a moving window of traffic. The only source of outside information explicitly provided is the IP addresses of devices deemed to be crucial in the functioning system. This information is asynchronously delivered by a decoupled two-part communication system.

A novel working sensor prototype, based on a unique implementation of the Trusted Network Group's IF-MAP web services based communication protocol, is described. Sensor communication between self-contained modules is accomplished with D-Bus. At multiple steps of a test scenario, the communication layer was utilized to provide information in a well-defined format between components and to external entities.

Multiple intelligent modules were deployed on a test system to monitor for anomalous behavior and create deceptive emulated hosts. The modules are exemplary implementations of self-learning components. In a test scenario, 45 of the 46 network attached devices were recognized and 10 of the 12 emulated devices were created with representative characteristics. Additionally, 99.9% of anomalous packets were recognized. The modules utilized the communication layer to provide notifications and retrieve information.

An algorithm was proposed and demonstrated to automatically deploy deceptive virtual network entities in a control system network. Six open source passive network-monitoring tools were evaluated and Ettercap was chosen for host identification. This differs from prior work in the field in which p0f is typically used. The algorithm created unique network stack signatures for twelve test devices. Eight of the twelve emulated devices were correctly identified by an aggressive Nmap scan. Several deficiencies with both the monitoring tools and virtual honeypot implementation Honeyd were discovered and discussed. These problems are: non-IP based traffic, OS identification database support, missing information and well formatted program output.

In order to show the necessary depth of the proposed automatic deployment and configuration, a usage scenario was executed. In this scenario an anomaly detection system monitored the network activity of the honeypots. The role of the automatically deployed honeypots was to attract and possibly delay an intruder on the network. The primary enabling technologies included continual host monitoring, reconfigurable deceptive virtual hosts and a network anomaly behavior monitor. The benefits of the presented system include: 1) reduced operator interaction, 2) low operational network impact, 3) increased awareness of the security state, and 4) an independent view of hosts and services that are active on the network. The behavior system alerted on 100% of the packets targeted at the virtual hosts.

Chapter 4. Communication Algorithms for Security Awareness

In the AICS architecture the IAA component is an anomaly based detection algorithm. Network monitoring systems in general and anomaly based systems in particular often provide information that is difficult for non-expert operators to comprehend. For instance, low-level network traffic information such as protocol types, port numbers or packet data is often times beyond the experience of common control system operators. This chapter contributes a novel solution to provide a human centric categorization of anomalies.

The **first** contribution of this chapter details a Human Interface for Security Awareness (HISA) algorithm for interpreting cyber incident information from anomaly based intrusion detections systems to present to a user (Vollmer & Manic, 2009). A similarity algorithm mapping anomaly results to signature based intrusion system rules is developed. Categorizations of attacks found in rules created for the Snort intrusion system were used as a basis of information to present to the user. A proof of concept system was developed using Perl native functions and custom modules.

The **second** contribution of this chapter an enhanced version of HSIA called the Computationally efficient Neural Network Intrusion Security Awareness algorithm (CeNISA) (Vollmer & Manic, 2009). A unique aspect is the training of an error back-propagation neural network with intrusion detection rule features to provide a recognition basis. Ethernet network packet details are subsequently provided to the trained network to produce a classification. This leverages rule knowledge sets to produce classifications for anomaly based systems. Several test cases executed on ICMP data packets revealed a 60% identification rate of true positives. This rate matched the previous work, but 70% less memory was used and the run time was reduced to less than 1 second from 37 seconds.

4.1 Problem Description

HISA/CeNISA focuses on the specific manner in which the results of anomaly-based systems are presented to an operator by making use of information from a signature based IDS. Rule based systems can identify specific intrusions and report human friendly messages. The rules themselves are created by people allowing for the opportunity to embed relevant information in the rule definition such as an attack category or well known vulnerability reference. Hence these rules are easy to comprehend as they present an abstract view of the information that is typically beyond the knowledge of the common control system operator.

Network traffic that is outside the norm can be identified by anomaly based systems. This feature provides the ability to detect new attacks. Systems such as these can report on the peculiarities that caused identification of traffic determined to be outside the norm. They do not have the information or generally the capability to express the attack alert in human terms. In contrast, rule based systems have a built in categorization mechanism. The human creation of rules provides the opportunity for general categorization of attack features. In addition to the rules primary purpose as an attack recognition repository, they are a valuable source of community knowledge. They provide a useful historical database of information and characteristics. This database of attack features can be used for similarity comparisons to help describe previously unseen attacks.

Anomaly based systems can identify new attacks and alert users. However, there have been few proposed mechanisms for delivering alert information in a meaningful way to system operators. Specifically anomaly systems can report whether or not traffic is anomalous or if it closely matches a known signature. The signature match requires either training a system with tagged data or generating rules based on known attack characteristics. The later method is similar to the HISA/CeNISA approaches. However the algorithms presented in this chapter makes use of community defined rule sets, instead of sample network data, which is unique.

4.1.1 Intrusion Detection Systems

There are two primary types of IDS's, signature and anomaly based. The primary purpose of both is to detect and possibly react to illicit network intrusion activity. The signature based systems provide monitoring and alert services based on static rule sets. Static rule sets perform well on known signatures but rely upon human experts to recognize an issue, perform analysis and develop a detection rule. As is shown in virus protection products, small variances in behavior can bypass static rules. This necessitates constant and expensive updates by the vendors. Rule based systems are analogous to virus protection software resident on personnel computers. Predefined rule sets capture characteristics of attack vectors. These sets perform well on known signatures but generally do not recognize novel attacks. In addition, minor variations in the signature of a known attack may not be noticed by the system. However, rules are developed and distributed by an active community and widely distributed. It is even possible to convert rules between different formats (Gosh, Schwartzbard, & Schatz, 1999). This provides a rich base of historical information that is readily available for exploitation by HISA.

An anomaly IDS is based on recognizing deviations from a learned model of normal behavior (Gosh, Schwartzbard, & Schatz, 1999). A representative model is built primarily on historical data. The features of future intrusions are not assumed a priori and anomaly decisions are based on profiling current activity, in contrast to the stored normal behavior (Linda, Vollmer, & Manic, 2009). Such a system is capable of detecting previously unknown and dynamically changing intrusion instances. This is most effective when the intrusions are distinctively different from the learned model of acceptable behavior.

Anomaly based IDS's usually report events via one of three different mechanisms. The first is simply if an anomaly event was detected or not (Zanero & Savaresi, 2004). This can be based on passing some threshold and may contain a confidence factor. Second, when labeled data is available, a supervised network can be trained to distinguish between different input vectors (Khan, Awad, & Thuraisingham, 2007). The output then can present the appropriate label if it closely matches any of the known categories. Third, a new set of rules for identifying attacks can be created (Huang & Wenke, 2003). These rules use a derived set of attributes from the data sets that are identified as anomalous. This requires analyzing the known attack attributes and choosing those that are general enough to identify a group of attack types.

IDS rules are used as a basis for network anomaly detection reporting in the HISA algorithm. A simple similarity algorithm was developed mapping the network packet characteristics to an IDS rule fields. The details of each anomaly packet are compared to all rule fields and a match is recorded as a Boolean value. After an exhaustive evaluation of all rules and packets, a summation of matches for each rule is computed. The category feature of the rule(s) with the largest match values is then presented as a classification.

According to (Analoui, Bidgoli, & Rezvani, 2007), IDS implementations primarily make use of the previously mentioned static rule configuration. Most of these IDS systems are generic in application and have rules designed for more typical information technology infrastructures. Control systems networks have similar communication infrastructures, but different behavior patterns. Research detailed in this section may help drive the control industry to make use of both types of systems working cooperatively together. These systems used in combination may be robust enough to cover a significant portion of both novel and known intrusions.

Even though current IDS systems implement a different approach to attack recognition, they have a common base in regards to input and intent. In turn, this can provide a common basis for notification output. As is shown in this chapter, the power of anomaly detection can be combined with the inherent knowledge representation of rule-based systems to provide attack information to an operator. The information is presented in terms of similarity to well defined attack classes, such as denial of service, and references to known vulnerabilities.

4.1.2 Snort Rule Analysis

Snort is an open source IDS created by Martin Roesch (1999). It is capable of performing protocol analysis, content searching/matching and many other abilities including using rule sets. Several rule sets are available for use including those officially approved by the Sourcefire Vulnerability Research Team (VRT) and those contributed by other communities (Emerging threats, 2012). Snort supports a simple rule language that matches against network packets, generating alerts or log messages. Rules are broken into two logical sections: rule headers and rule options. Because of these factors it was chosen as a starting point for our algorithm implementation.

Rule headers contain necessary protocol fields that every rule must have and rule options contain a list of optional information used to refine a match. The rule field format and an example rule is as follows:

<action> <protocol> <source IP> <source port> <direction> <destination IP> <destination port> (<rule options>)
Alert tcp any any -> 192.168.1.0/24 any (content:”ELHO”)

The rule action tells Snort what to do when a match occurs. A common action is to log information to an alert file. The protocol field specifies one of four possible values: TCP, UDP, ICMP and IP. Each value has options specific to the protocol available for use in the option section. The source IP address section can contain the keyword any, a single IP address or a CIDR (Classless Inter-Domain Routing) block. CIDR blocks allow for specifying ranges of IP addresses. Port numbers may be specified as a single static port, a range or use the keyword any.

There are two direction operators. One specifies that the source and destination sections of a rule must match the appropriate items from a packet. The bidirectional operator indicates that the source and destination sections can match either portion of a packet. This allows for tracking two way conversations (e.g. FTP sessions).

Rule options provide further refinement of matching parameters and tie the rule to a rule identification system. There are four major categories of rule options: general, payload, non-payload and post-detection. General options provide knowledge about the rule such as reference information, rule identification, and specific log messages. Payload options examine data contained in the packet data such as content matching expressions. Non-payload options provide matching specifications against packet header data outside of ports and IP addresses. Options include fragment offsets, time-to-live values and specific IP options.

An important option keyword is *classtype*. This keyword is used to mark a rule as belonging to a specific attack class. The attack classes are predefined in a configuration file. This makes for a consistent description and format for the attack information. Although it is an optional field, all rules provided for general consumption include a classification. These classifications can be provided to operators as they convey meaningful information without requiring a lot of in-depth technical knowledge about cyber incident specifics. The default classifications are show in Table 11. The classtype names are descriptive but further details can be found in (Sourcefire Inc., 2009).

Table 11. Snort Classifications

Classtype	*Classtype*
attempted-admin	rpc-portmap-decode
attempted-user	successful-dos
kickass-porn	successful-recon-largescale
policy-violation	successful-recon-limited
shellcode-detect	suspicious-filename-detect
successful-admin	suspicious-login
successful-user	system-call-detect
trojan-activity	unusual-client-connection
unsuccessful-user	web-application-activity
web-application-attack	icmp-event
attempted-DOS	misc-activity
attempted-recon	network-scan
bad-unknown	not-suspicious
default-login-attempt	protocol-command-decode
denial-of-service	string-detect
misc-attack	unknown
non-standard-protocol	tcp-connection

Snort builds a tree structure used to compare against packet features. Each mandatory field in a Snort rule is stored in a Rule Tree Node (RTN). An OTN (Optional Tree Node) is associated with an RTN and used to store optional rule fields. If multiple rules have the same RTN fields, a single node represents them. This optimization feature allows for removal of multiple rules from consideration once a negative match occurs. This is a detrimental aspect to the proposed HISA algorithm and it is important to recognize the impact on the processing time.

The intrusion rule can be seen as a Boolean truth statement. In order for Snort to identify a match, a logical *and* of all positive field matches is necessary. Upon discovery of a false condition in the *and* evaluation, further processing of that rule is halted. The set of all rules would then be equivalent to a logical *or*. Short circuit of rule evaluation prevents using a modified Snort source as a base. As the Snort engine processes a rule section that proves false for a rule that rule is no longer considered for a possible match. This is a logical performance enhancement that speeds the execution of the rule engine. However this inhibits the implementation of the proposed algorithm.

In the HISA process, all parts of a rule will be evaluated to see if it matches the packet data regardless of a previous rule section match. A value of 1 for each header field indicates a match, 0 otherwise. The header values are a necessary match condition but are not always a strong indicator of similarity. For instance, a TCP packet on port 80 is a very common packet as this is traditionally where the http protocol takes place. However, a large amount of attacks of varying types can take place via http. A stronger indicator of an attack type is in the packet header and payload details (i.e. Snort rule option fields). Consequently, these details are required for a stronger response upon match. Detailed rules that provide precise indicators are typically a more relevant match indicator.

Three sources for acquiring rules were used. Sourcefire VRT certified rules and community rules are available online from the Snort repository. The third set was obtained from emerging threats and is available online at www.emergingthreats.net. All of these sets combined to define 16,181 rules covering 31 of the 34 class categories. Table 12 describes the protocol and number of related rule sets available in these sets.

Table 12. Snort Rule Protocol Counts

Protocol (number)	***Protocol (number)***
TCP (12,325)	UDP (890)
IP (2,808)	ICMP (158)

4.1.3 IP/ICMP

The Internet Protocol (IP) layer is a connectionless datagram delivery service defined in RFC 791. The transport layer, i.e. TCP, is responsible for any reliability or ordering of traffic. Routing of the datagram is the responsibility of the IP layer as each instance contains a source and destination address. IPv4 defines a data structure containing this information (Stevens, 2003). Other protocol versions are not considered in this evaluation. The header portion of the datagram has many different fields. Those that are relevant to this discussion are described next.

Enabling fragmentation and reassembly, the IP identification field is a 16 bit value set to different values for each datagram. A datagram may be broken into smaller datagram's (fragmented) if its size exceeds the Maximum Transmission Unit (MTU) of a path. This field is not always set if there is no fragmentation possible.

Time to live (TTL) is an 8 bit field set by the originator. Each time the datagram is forwarded by a router the value is decremented by 1. Given the field size the maximum value is 255. If this value reaches 0 then the packet is discarded.

Forty bytes of optional data are allowed to follow the fixed size twenty byte header. Ten different options are defined for use: no-operation (NOP), end-of-list (EOL), loose source and record route (LSRR), strict source and record route (SSRR), Timestamp, Record route, Basic security, Extended security, Stream Identifier, Router alert.

The Internet Control Message Protocol (ICMP) is defined in RFC 792. ICMP messages are encapsulated within an IP datagram. The first four bytes of these messages have the same format. These bytes are composed of 3 fields: 8-bit type, 8-bit code and 16-bit checksum. Composition of the remaining bytes depends upon the message type.

The type field has 15 different possible values. This field in conjunction with the code field defines the message type. ICMP messages belong to one of two categories that encompass all the type and code combinations: query and error. Some messages that belong to the error category are handled differently.

The sequence number and identifier field can appear in address mask request and reply messages. They are both 16-bit fields and can be set to any value that fits in the range. The intent is for the responder to provide these fields back to the sender to allow for synchronization of messages.

4.1.4 Work Related to HISA/CeNISA

Rule based IDS's are being used to provide an important layer of security for computer systems and networks. The work in (Aickelin, Twycross, & Hesketh-Roberts, 2007) states that an IDS's responsibility is to detect suspicious or unacceptable system and network activity and to alert a systems administrator to this activity. Their intent was to identify a way in which Snort could be improved by generalizing rules to identify novel attacks. The conditions and parameters of a sample set of rules were modified to include broader ranges. Empirically the approach was shown to be effective using a set of network capture data obtained from the Massachusetts Institute of Technology. Previously undetected variants of attacks were identified with a subsequent increase in false positive rate.

Reduction of false positive rates is important in rule based systems. Because the definition of a rule can be general, it is increasingly likely that false positives will occur given a large volume of network traffic. Long, Schwartz and Stoekclin (2006) made use of Snort alerts in XML format to form clusters. The clusters were formed using an XML distance measure. This proved effective in discriminating between normal sessions that raised false alerts and those that contained real attack information.

Many IDS's use rule based signature solutions. Rule formats used by most of these systems are not standard. This leads to duplication of effort making definitions for the same attack in multiple expressions. In (Eckmann, 2001) the authors proposed and implemented an automated rule conversion system. The results showed that Snort rules could be generalized for use in other systems. A system such as this could be used by the HISA/CeNISA algorithms to broaden the scope of rules.

Lee, Stolfo and Mok (1999) describe data mining techniques to construct network behavior models that are both accurate and efficient doing real time processing. These techniques were used to build network intrusion models. Patterns of normal network traffic were created and stored. As new network traffic was presented to the system, a pattern was created. A comparison of this pattern to the stored patterns determined if the traffic was considered abnormal. The concept of comparing a normal baseline to real time data is similar to HISA/CeNISA. However CeNISA uses signature rules to establish a normal baseline and a neural network for pattern comparison.

4.2 Human Interface for Security Awareness

The increased threat of cyber attacks is well documented and has been acknowledged by many governmental, commercial and academic entities world wide (Meserve, 2007). Computer based systems used within many critical infrastructures to monitor and control physical functions are not immune to this threat and may potentially be more vulnerable than common information technology systems (Taylor, Oman, & Krings, 2003). Despite large expenditures of effort, systems will continue to be penetrated or threatened by human failings. These issues provide incentive to develop capabilities that enhance operator’s awareness and understanding of system security measures.

Comprehensive state awareness of safety and security is a preeminent concern for critical infrastructures. Modern implementations of these infrastructures rely heavily on networked communication systems. Network intrusion attacks and anomalies can lead to high financial costs and the endangerment of public safety. Connecting operators to cyber anomalies in an understandable manner is one part of a resilient design (Rieger, Moore, & Baldwin, 2013). Therefore it is imperative that control operators receive relevant and comprehensible cyber health information from systems like AICS, which is described in Chapter 3.. In this section a solution for presenting cyber incident information from anomaly based Intrusion Detection Systems (IDS) to human operators is presented. The stored knowledge inherent in rule based intrusion systems is used to classify newly identified attack vectors. A similarity algorithm, mapping novel to known attack classifications, is presented.

4.2.1 HISA Algorithm Description

The goal of HISA is to present information characterizing an unknown attack to a human user. An approach utilizing prior rule definitions to find a close match is described. The computational process necessary to do so is described as pseudo code in Fig. 4-1. Each major functional area is described in detail in the sections following.

The network packet data is identified by an external anomaly detection routine and is passed to the HISA algorithm for possible identification. The identification process consists of traversing all rule sets looking for those rules that match as closely as possible by matching each part of a rule.

```
Initialization:
find rules and load them into a rule structure.
open network packet file.

Check for matches:
Loop through packets
    Decode packet;
    Loop through rules
        initialize matches to 0;
        check for match on IP address;
        check for match  on protocol;
        switch on protocol type
            check matches on protocol specific;
        store match information;
    end rule loop
end packet loop

Process results:
Loop through packet records
    Loop through rule match record
        sum match results;
    end rule match loop
    identify largest match count rule(s);
end packet loop
```

Figure 4-1. HISA Pseudo Algorithm

4.2.1.1 Initialization

The rules are defined in several different file sets. These files are opened and loaded into memory. The Snort specific rule format is well defined in and is the only rule format currently supported (Sourcefire Inc., 2009). Each rule is parsed into its component parts and stored in a record for fast search and retrieval. The record itself is a hash, or associative array, of rule part names to textual values. These rule parsing and record population routines were implemented using a Perl module called Net::Snort::Parse (Caswell, 2012).

The record matches are stored in the records as text by default. An additional field is included to handle IP addresses that can be stored using Classless Inter-Domain Routing (CIDR) notation. The CIDR record field maintains an object reference to handle these types of checks. The Perl module Net::CIDR, available on CPAN, has functions for dealing with IP address range checks.

Special handling of port and IP sections is required. Snort rules allow for variables or the keyword 'any' in these sections. The variables can be defined once with specific values that are replaced in the rules when encountered. When processing rules, these variables need to be accounted for, in both destination and source, by replacing them with legal values. Tables 13 and 14 illustrate the mapping from variable name to replacement value that is used.

Table 13. IP Variable Definitions

Variable	*Definition*
any	0.0.0.0/0
$AIM_SERVERS	!0.0.0.0/0
$*_NET	0.0.0.0/0

Table 14. Port Variable Definitions

Variable	*Definition*
$HTTP_PORTS	80
$SHELLCODE_PORTS	!80
$ORACLE_PORTS	1521
$SSH_PORTS	22

The anomaly network data is stored in the industry standard pcap format. The packet data consists of anomalous packet data only. The system assumes that each packet is of interest and attempts to match each packet to the stored rule information. The packet can be seen as a feature vector $\vec{v}$ where each feature v is a unique data point in the vector such that $\vec{v} = v_0 ... v_i$. The set **S** contains all feature vectors $\vec{v}$ in a collection of network packet data.

4.2.1.2 Match Check

The match check is a O(n^2) portion of the algorithm as is shown in (1) where n is the number of packets and rules (n approaches infinity). The match function maps the attributes of rule *(r)* to those of packets *(p)*. The function *T(n)* is a constant time operation with a fixed set of comparison operations.

$$T(n) = \sum_{p=1}^{n} \sum_{r=1}^{n} match(r, p) \tag{4-1}$$

Each packet in the stored file is compared against the specifics of each rule definition. A match record stores an integer value for each match item defined. If a match occurs, a 1 is stored 0 otherwise. This is a simple method to indicate a match. The value is stored as an integer despite the current storing of a binary value. A future improvement may be to weigh specific matches differently with a unique value.

The current system matches on the following items: source IP address, source port, destination IP address, destination port, protocol, payload content, ICMP id, ICMP sequence, ICMP type, ICMP code. These are a subset of the possible match fields with a focus on ICMP values as is explained in the results section.

4.2.1.3 Process Results

The match check records are processed individually. Each match item value is summed and tracked. The resulting sums are sorted and grouped from largest value to smallest. The top matching item is then considered to be the closest match. The class type, as previously defined in Table 11, and message ID of the winning sum is presented as the closest match to the user. When the largest sum value has multiple rule matches, a weighted class list is presented.

The weighted class list shows the percentage of each class that matched. For instance, if five rules match and three of them are of the denial-of-service class and two are policy-violation the resulting values of 0.60 and 0.40 respectively are presented. The intent is to provide the user with high level information about the possible nature of the attack.

4.2.2 Experimental Results

ICMP rules and characteristics were the primary focus of the test data set creation to show a proof of concept. ICMP packets have fewer options in both packet details and rule match items. In addition, as can be seen in Table 15, the number and class type representations are reduced. This simplification could have had a negative impact on the results. With fewer data points to work from, the similarity measure may not have produced meaningful results. However this does not appear to have been the case as is shown in this section.

Table 15. ICMP Rule Classifications

ClassType	*Count (total 145)*
denial-of-service	1
misc-activity	103
bad-unknown	3
attempted-recon	11
attempted-user	1
trojan-activity	9
attempted-dos	16
network-scan	1

As a base case, to prove program correctness, the system was exercised with test packets against all ICMP rules. These test packets were crafted to cause specific rules from different classes to exactly match the rule parameters. It was surmised that if the system could not match known vectors than it might be fundamentally flawed. The system correctly identified 100% of test packets with the appropriate static rule definition.

Nemesis is a network packet crafting and injection tool (Nathan, 2012). Implemented as a command line tool, it is well suited for reproducing test scenarios. Nemesis can create and inject ARP, DNS, ETHERNET, ICMP, IGMP, IP, OSPF, RIP, TCP and UDP packets. It was used to produce the ICMP test packets. An example command line used to create a packet is shown below:

```
> nemesis -i 8 -s 0 -d 666 -d lo
```

This command will create an ICMP packet with a type of 8, a sequence number of 0 and ICMP-ID within the header of 666. Subsequently, the packet will be placed on the *lo* interface of the machine.

The packets created using the nemesis tool were captured and stored as a pcap data file. Five different test packets were created to trigger five different Snort rules. Each rule was chosen for its membership in a different class type. The packet nemesis command line specifics and class types are presented in Table 16.

Table 16. Nemesis Command Line and Class Types

Packet Details	*Class Type*
-i 0 -s 0 -e 667	attempted-dos
-i 8 -s 0 -e 666	attempted-recon
-i 5 -c 0	bad-unknown
-i 3 -c 2	attempted-user
-i 8 -s 14611 -c 123	misc-activity

The 145 ICMP rules had representation from all three sources mentioned in section 3. After proving the correctness via the previously mentioned base case, the matching rule definitions were removed from operation. The intent was to run the test packets through the system without the matching rules. This simulates the availability of unknown attack vectors and tests the systems ability to identify similarities with known attacks. It also provides a known set of results to compare the categorization results against. The results are shown in Table 17.

Table 17. ClassType Test Results

Correct ClassType	*Identified ClassType*	*% Match*
attempted-dos	attempted-dos (3)	100%
attempted-recon	attempted-recon (4) network-scan (1)	80%
bad-unknown	bad-unknown (2) attempted-recon (3) misc-activity (28)	6%
trojan-activity	attempted-user (1)	0%
misc-activity	misc-activity (1)	100%

The first column in the table is the class type that the test packet originally matched before removal of the associated rule. The second column shows the system classification output. The class type is followed by the number of rule matches that had the highest similarity score. As is illustrated by the number in parenthesis, it was possible to have multiple hits with the same score. In the final column a percentage value is recorded. This value represents a match percentage of the output results. For example, the attempted-dos test had three results with the same class type. All of the class types matched the correct value resulting in a 100% score. The attempted-recon test had four correct hits and one miss. This resulted in a score of 80%.

Overall, if a match score of 80% or greater is considered as successful, the system correctly identified three of the five (60%) of the test cases. Veracity and detail of rules makes a difference. The quality of the answer is only as good as the basis from which it is drawn. In this case, the Snort rule set and matching algorithm are the basis features. It was observed that 71% of the ICMP rules are in the misc-activity category. There may be an opportunity for refining these rules into a more useful category. However even with this limitation the results are positive and provide useful information.

4.3 Computationally Efficient Neural Network Intrusion Security Awareness

The work presented in this section is based on the previously presented HISA algorithm. An entirely new and improved solution to the same problem is detailed. This section describes the performance improvements of the Computationally efficient Neural Network Intrusion Security Awareness algorithm (CeNISA) as well as comparing results to the original. CeNISA is a pattern recognition algorithm that maps novel attack vectors recognized by anomaly Intrusion Detection Systems (IDS) to known attack classifications. A multi-layer feed forward Error Back Propagation network provides the primary performance improvement for CeNISA.

Signature and anomaly based systems both have strengths and weakness when used in isolation. One issue with anomaly based systems is obtaining labeled data that is not artificially generated (Stolfo, Lee, Chan, Fan, & Eskin, 2001). As was pointed out earlier, signature based solutions can miss new signatures or variations on known attacks. The focus of this section is on combining the capabilities of both to overcome these issues. Additionally, a classification is produced that is suitable for cyber security awareness. The previously described Snort rules are used as training input. The use of features from these rules as training vectors is a unique aspect of the CeNISA algorithm. To show a proof of concept, the ICMP network attack packets used to test HISA were evaluated on the new algorithm.

4.3.1 CeNISA Algorithm Description

The goal of CeNISA is to present information characterizing an unknown attack to a cyber security consumer. A unique approach utilizing historical rule definitions to find a close match is described. The algorithm consists of two critical phases: 1. Building the EBP network. 2. Extraction of data points from network data and subsequent presentation to the trained network. The Neural Network consists of three fully connected layers: input, hidden and output. Each major functional area is described in detail in the following sections.

4.3.1.1 Network Design

There are numerous options available for rule definitions but an analysis of existing ICMP rules showed that in practice only eleven are in use. Of these possible values, three were excluded byte_test, content and threshold. The first two specify specific comparisons of payload information that are inappropriate inputs for a neural network. The threshold option specifies Snort specific behavior on rule alerts and does not provide useful information. Table 18 lists the nine features (eight plus the classification) chosen for inclusion in the training feature vector.

Table 18. CeNISA Training Features

Feature	Description
dsize	packet payload size
icmp_id	ICMP ID field
icmp_seq	ICMP sequence
icode	ICMP code field
id	IP id field
ipopts	IP option field
itype	ICMP type field
ttl	IP time-to-live
class	Snort classification

4.3.1.2 Training Features

Normalization of these input vectors is accomplished according to (1) where x_i is a feature instance and x represents the set of a feature type.

$$x' = \frac{x_i - \min(x)}{\max(x) - \min(x)} \qquad \textbf{(4-2)}$$

This normalized input is passed to the next layer in the network. The net input of node *i* in layer *k+1* is calculated as

$$n^{k+1}(i) = \sum_{j=1}^{Sk} w^{k+1}(i,j)a^k(j) + b^{k+1}(i) \quad (4\text{-}3)$$

Here *Sk* denotes the number of nodes in layer *k*, $w^{k+1}(i,j)$ is the weight of the connection from neuron *j* in layer *k*, $b^{k+1}(i)$ is the bias of neuron *i* and $a^k(j)$ is the output from neuron *j* in layer *k*. The output of node *i* in layer *k*+1 is

$$a^{k+1}(i) = f^{k+1}(n^{k+1}(i)). \quad (4\text{-}4)$$

where f^{k+1} is the activation function of neuron *i*. In CeNISA the hidden node layer cardinality matches that of the input layer, which is nine. Fig. 4-2 shows the relationship of inputs and layers. It should be noted that the nodes are not fully connected in the figure to avoid cluttering the image.

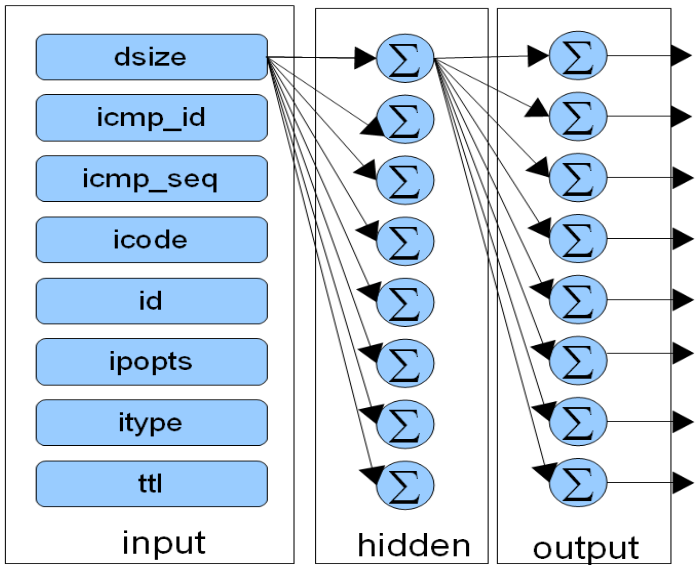


Figure 4-2. Neural Network Diagram

The training set only contains 8 of the possible 38 classifications therefore the output layer consists of 8 nodes. Each output node represents a possible classification. An error term is calculated using the sum of squares function where *o* is the network node output and d is the desired output.

$$E = \tfrac{1}{2}\sum_{n=1}^{c}(o_n - d_n)^2 \quad (4\text{-}5)$$

After the value for the output and error is computed, weight adjustments for the hidden and output layers are calculated.

Equation 4-6 provides the delta to apply to each weight; alpha is the learning rate *np* is the number of input patterns. An alpha of 0.3 was used for training.

$$\Delta w_p = \alpha \bullet \sum_{o=1}^{1} \sum_{p=1}^{np} \left[\left(d_{op} - o_{op} \right) F'\{z_p\} f'(net_p) x_p \right] \tag{4-6}$$

After a complete set of updates for each training input pattern, the sequence of calculating outputs and feeding error information back through the network layers is reiterated 1000 times. These weights are saved and used for evaluation of extracted network features after the training period is complete.

In order to validate the recognition power of the described network a 10-fold cross-validation scheme is used. The input vectors are divided into 10 subsets of approximately equal size, which was 13 for this instance. The network is trained 10 times, each time leaving out one of the subsets. The subset left out is subsequently used as the test data to judge the network error.

The network packet data is identified by an outside anomaly detection routine and is passed to the CeNISA algorithm for classification. The classification process consists of extracting the features found in Table 18 and presenting the values as input to the EBP network. The processing of the packets is described next in a pseudo coding style.

```
Open Pcap file
Initialize feature structure
Loop
    strip Ethernet encapsulation
    decode IP content information
    decode ICMP content
    store values in feature list
    call EBP Network function
    store results
End loop
```

Figure 4-3. CeNISA Pseodo Algorithm

4.3.2 Experimental Results

The original ICMP rule set of 145 was reduced to 129 instances. Duplicate and ambiguous rules (due to feature selection) were removed. Table 19 details the class types and associated counts of the rules. A fully trained network using these rules correctly identified 75% of the snort rule classifications. The confusion matrix shown in Table 20 provides the classification details. The class values have been abbreviated for space considerations. As can be seen from the matrix, a large number of rules are marked as miscellaneous. This may have caused over fitting of the solution to this classification.

Table 19. ICMP Rule Classifications

Class Type	*Count (total 129)*
denial-of-service	22
misc-activity	81
attempted-recon	11
trojan-activity	9
bad-unknown	4
attempted-user	1
network-scan	1

Table 20. Confusion Matrix

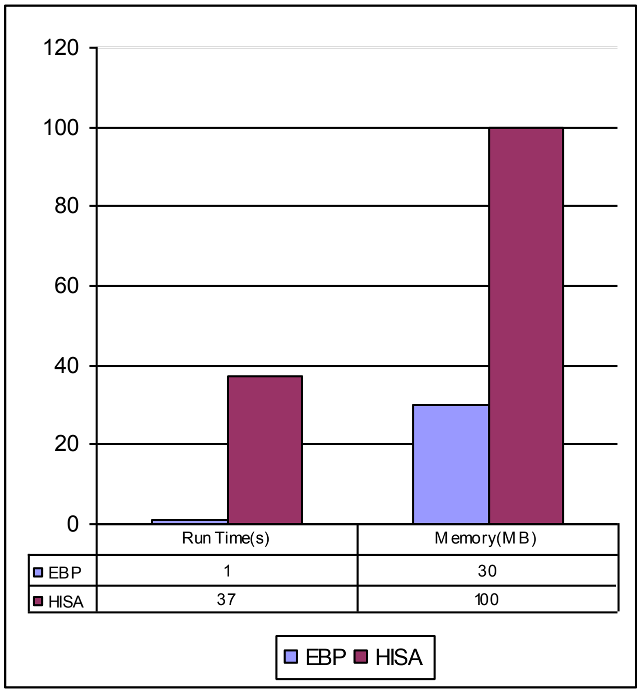
	a	**b**	**c**	**d**	**e**	**f**	**g**
a:misc	74	2				1	4
b:trojan	5	4					
c:network	1						
d:bad	3	1					
e:a-user	0						1
f:recon	6					4	1
g:dos	7						15

The confusion matrix only shows the results from utilizing the Snort rules. The next step involved presenting the features from the five ICMP test network packets. The network correctly identified denial-of-service, misc-activity and attempted-recon for a 60% success rate. These classes represent the majority of the training vectors and should be the most recognizable to the network.

4.3.2.1 CeNISA versus HISA

In comparison, using the test cases as a basis, the modified algorithm using an EBP network provided the same accuracy (60%) as the original. However, a large improvement in efficiency with regards to run time and memory usage was realized as can be seen in Fig. 4-4. The HISA algorithm had a runtime of 37 seconds and utilized 100 MB's of RAM. The enhanced version took less than 1 second and required 30 MB's of RAM. This execution time does not take in to consideration the training phase, which is a one-time cost.

HISA stores the rule information as a data structure in memory and compares the network characteristics to values in the structure. The equivalent functionality in CeNISA is the EBP network. The rule attack knowledge is effectively stored within the weights between layers.



	Run Time(s)	Memory(MB)
▪ EBP	1	30
▪ HISA	37	100

Figure 4-4. Runtime Performance Comparison

4.4 Chapter Summary

This chapter's contribution described two algorithms: 1) HISA and 2) CeNISA. Both algorithms are solutions for presenting anomaly based intrusion detection alerts based on similarity to static rules. Rules developed for the Snort IDS and their default classifications were used as input features. For HISA, a similarity algorithm was developed that utilized a simple match summation process. This process utilized native Perl functionality and custom modules. For each match found in a rule, a positive value was noted. The rule(s) with the max sum was used to identify the attack class membership. This result was subsequently presented to the user as an indicator of system cyber security status. An identification rate of 60% demonstrated the effectiveness of the proposed algorithm on test ICMP data.

Subsequently CeNISA, an enhanced algorithm for presenting anomaly based intrusion detection alerts based on an error back-propagation neural network, was presented. Snort rule information was used as training input vectors. The results from the trained network were compared with a previously derived similarity algorithm utilizing the input vectors gleaned from network traffic. The resulting output for both designs was subsequently presented to the user as an indicator of system cyber security status. An identification rate of 60% demonstrated the effectiveness of the proposed algorithm on test ICMP data. Although the test identification rates were identical, the EBP network based solution required 70% less main memory and executed 37 times faster.

Chapter 5. Autonomous Rule Creation and Distance Metrics

The **first** contribution of this chapter examines the performance of three distance measures Mahalanobis, Euclidean and a learned metric in the context of CI routines. The **second** contribution is the presentation of a multi-modal optimization solution that maps anomaly traffic features to rule sets. The rule sets are distributable to a wide range of software solutions.

Distance measures are a key component in several Computational Intelligence (CI) solutions such as multi-modal optimization problems. Optimization can be used in support of autonomic processing of anomaly data. This process determines the similarity of combined anomalous network traffic features to an idealized recognition formula. Therefore examining the performance of distance or similarity metrics when applied to autonomic computing solutions is important (Vollmer, Soule, & Manic, 2010).

Extensions of simple Genetic Algorithms, particularly types of crowding, have been developed to help solve multi-modal types of problems. Within the context of the experiments presented in the first section of this chapter, empirical evidence shows that the statistical based Mahalanobis distance measure when used in Deterministic Crowding produces equivalent results to a Euclidean measure. In the case of Restricted Tournament selection, use of Mahalanobis found on average 40% more of the global optima, maintained a 35% higher peak count and produced an average final best fitness value that is 3 times better.

The final part of this chapter explores a multi-modal genetic algorithm solution for autonomous rule creation (Vollmer, Alves-Foss, & Manic, 2012). Many network anomaly behavior techniques have been proposed as solutions for anomaly based network intrusion detection. However, little work has been done on communicating the anomaly characteristics to other systems. Translating a newly discovered intrusion recognition criteria into a distributable rule can be a human intensive effort. The last section presents an automated solution to create detection rules once an anomaly has been identified. Empirical tests using captured ICMP network packets from an anomaly detection system are used as input and Snort rules are output. Candidate output rules are sorted according to a fitness value and any duplicates are removed. When run against ten test cases, a 100 percent rule alert rate was achieved as expected. Out of 33,804 test packets 3 produced false positives. Each test case produced a minimum of three rule variations that could be used as candidates for a production system.

5.1 Distance Metric Comparison for Multi-Modal Genetic Algorithm Solutions

Problems, in which a number of points are potentially good solutions, while not necessarily optimal, are defined as multi-modal. Genetic Algorithms (GA), including crowding approaches such as Deterministic Crowding (DC) and Restricted Tournament Selection (RTS), have been developed to maintain sub-populations that track these multi-modal solutions. For example, multi-modal GA's have been used in the design of a nuclear reactor core (Mishra, Pandey, & Chauhan, 2009). In addition, two surveys highlight the multiple uses of GA's in control systems and power engineering tasks (Rajkumar, Vekara, & Alander, 2008), (Fleming & Purshouse, 2002). These tasks include optimization for controller design and model identification, fault diagnosis, reliable systems, robustness analysis and robot control.

The basic idea of a multi-modal GA is to encourage the evolution of subsets representing diverse solutions in a single population during the evolutionary process. In order to measure this diversity, distance measures are employed. Given better distance measures, improved results may be realized. Thus, the effectiveness of Mahalanobis distance in comparison with Euclidean distance in two real value encoded Genetic Algorithm solutions is examined.

When optimizing multi-modal functions a conventional GA's population tends to converge to just one of the optimal, or near optimal points. This characteristic occurs because of Genetic Drift and is an artifact of the application of random selection processes to finite populations (DeJong, 1975). This convergence to only one solution is undesirable in multi-modal optimization of real problems. Because a GA utilizes a population of many (hundreds, thousands or more) possible solutions, modifications to the algorithm can enable maintaining several optima.

One such modification, Fitness Sharing, lowers each individual's fitness by an amount relative to the number of similar individuals in the population (Goldberg & Richardson, 1987). Similarity is determined by evaluating a distance measure between population elements. Another GA modification, Deterministic Crowding, is an improved version of De Jongs Crowding. After crossover and mutation, each resulting child individual replaces the most similar parent used to create it if it has a higher fitness value. Similarity is typically computed using phenotypic distance (Mahfoud, 1995). Restricted Tournament Selection, similarly to DC, creates two new children (Harik, 1995). These children then compete with a fixed number of randomly chosen individuals from the population; the number of competing individuals is defined by the Crowding Factor (CF). The nearest individual chosen for the competition is selected from the CF population using a distance measure.

Distance measures as a means of determining similarity or closeness are a common algorithmic feature for many GA implementations such as those used in Crowding. The distance measure used is dependent on the population definition but typically solutions like Euclidean distance or Hamming distance are implemented. In addition, these distance measures can occur in different domains of the problems set i.e. Phenotypic, Genotypic and Fitness (Natalia, Hugo, & Raul, 2000).

5.1.1 Algorithm Description

This section describes the implementation details of a Genetic Algorithm and the two population diversity algorithms. Equations for the two distance metrics and their computational complexity are discussed. The final section discusses the test functions used for evaluation.

For evaluation purposes, solutions to minimize five benchmark functions with a range of different dimensions were evolved utilizing two forms of a crowding Genetic Algorithm (GA). A pseudo-code implementation of a GA is shown in Fig. 5-1 below and lines with numbers are described more fully later.

```
1 Create an initial population
  For Each Selection Process
      2 LOOP while below execution count
          3 select individuals as parents
          4 create children from parent (crossover/mutation)
          5 select and replace individuals with children
          6 update fitness values
      END LOOP
  End For Each
```

Figure 5-1. GA pseudo code

One of the first tasks in building a GA is to decide upon a representation of the solution population (line 1 in Fig. 5-1) and create a number of individuals in that population. All of the functions defined for this project make use of real values as inputs. This made for a natural definition of the population representation and floating point numbers were used to represent each of the n positions in a given individual solution. Therefore an individual can be seen as a vector $\vec{v}_i = (\mathrm{x}_0, \ldots, \mathrm{x}_\mathrm{n})$ of floating point values x. The total population is a set of vectors $P = \{\vec{v}_0, \vec{v}_1, ..., \vec{v}_T\}$. For each test function, the initial population size was set to 60. The initial generation of individuals is populated by randomly generating values uniformly in the domain range of the given function. The test function section provides the ranges of each function used in testing.

After an initial population is created a series of variations and selections must take place on the population individuals. This can continue until some acceptable solution or predefined resource limit is reached. For this project, each function has an associated max number of iterations value and is shown in Table 21 as iterations. When this number was reached, after incrementing by one each pass through the loop (line 2 in Fig. 5-1) and initially starting at zero, processing terminated and the fittest individual in the population was isolated as the final best solution. The fitness, in this case, consisted of the value returned from exercising the test function with the real value parameters represented by the individual. The fittest individual produced the smallest function output.

Table 21. GA Execution Variables

	Sphere	**Rastrigin**	**Ackley**	**Griewangk**	**M6**
Iteration^2	400	400	400	400	400
Iteration^3	500	500	500	500	-
Iteration^5	600	600	600	600	-
Optima^2	1	4	1	5	25
Optima^3	1	8	1	5	-
Optima^5	1	32	1	5	-
niche	0.2	0.1	1	0.9	0.5

Superscripts in column 1 indicate the dimensionality of the solution population.

A steady state population model was implemented. This means that, for each iteration through the loop, only a small part of the original population is changed. This is in contrast to a generational model where the entire population is replaced by the offspring. In this case, two individuals were selected as parents and two offspring were created from them. Subsequently, two more individuals were then selected for replacement by the new offspring creating a new generation.

The two candidate parent individuals were randomly selected independent of any fitness or distance measure (line 3 in Fig. 5-1). After crossover of the parents occurs, the two resulting offspring are evaluated to replace candidates in the population. This is a critical step for the maintenance of a multi-modal solution set. The determination of how replacement individuals are selected is discussed later in the section on replacement strategy.

Crossover between the two parents to create two new offspring utilized whole arithmetic recombination (line 4 in Fig. 5-1). Each position of the child is a new value created from the values at the same position of the parent vectors. This new value lies between that of the two parents and is created for each child using equations (5-1) and (5-2). Where *x* and *y* are the values at *i*th position in the parent vectors and α is a weight adjustment. The weight adjustment value α used for all functions is 0.6.

$$Child1 = \alpha * x_i + (1-\alpha) * y_i \tag{5-1}$$

$$Child2 = \alpha * y_i + (1-\alpha) * x_i \tag{5-2}$$

After crossover, mutation occurs every iteration on both child individuals prior to replacement selection (line 4 in Fig. 5-1). This mutation is non-uniform with a fixed Gaussian distribution which means most of the changes made will be small. For each position in the selected individual, a value drawn randomly from a Gaussian distribution with mean zero and user defined standard deviation of 0.1 is added to it. If this operation results in a value outside of the acceptable function range, defined in the test function section, the value is set equal to the closest boundary value.

Finally the fitness values for the mutated individuals are updated (line 6 in Fig. 5-1). This is the last step before the next iteration is performed following the steps already outlined with a population set that contains the new individuals.

This entire process is repeated for both replacement selection algorithms on a copy of the same population. An exact copy of the initial population is carried over from one replacement strategy trial to the next ensuring the same starting data sets are used. In addition to the multiple iterations of both selection routines on the same data set, an outer loop is executed 100 times (i.e. 100 independent trials). Because of the stochastic nature of crossover and mutation this is done to obtain a higher confidence in the performance measures.

5.1.1.1 Replacement Strategy

In order to maintain a diverse solution set within the population, two methods of choosing replacement candidates were considered: Restricted Tournament Selection and Deterministic Crowding. As mentioned in the introduction, both of these techniques are capable of maintaining a diverse population of solutions. The implementation of line 5 in Fig. 5-1 makes use of only one of the replacement selection techniques at a time. For comparison purposes, both were implemented and exercised in independent trials.

Uni-modal GA solutions using tournament selection randomly pick two individuals for replacement. These values are replaced if the new children have better fitness values. Restricted Tournament Selection (RTS) picks a candidate replacement individual that is closest to the new child from a subset of the population of window size w. The size of the population contained in w is defined by empirical testing (30 for this project in all cases) and each member is drawn from the original population using a uniformly random selection process. Closeness is determined by a distance function. After determination of the closest individual to the candidate child a competition is held based on fitness between the child and selected individual. The one with the best fitness is selected for inclusion into the solution population.

Deterministic crowding introduces competition between the children and the individuals used to create them. After crossover and mutation each child replaces the nearest parent if it has a higher fitness. Given two parents (P1, P2) and two related children (C1, C2), two of the four possible tournaments are executed. Selection of the tournaments is determined by the smallest distance value between a parent and child. The pseudo algorithm for this procedure is shown in Fig. 5-2 where F is a fitness function and D is the distance function. It should be noted this is for optimizing a minimization problem where a smaller fitness value is better.

```
IF (D(Pi,Ci) + D(Pj,Cj)) <= (D(Pj,Ci)+(D(Pi,Cj))
        If (F(Cj) < F(Pj) then replace Pj with Cj;
        If (F(Ci) < F(Pi) then replace Pi with Ci;
ELSE
        If (F(Ci) < F(Pj) then replace Pj with Ci;
        If (F(Cj) < F(Pi) then replace Pi with Cj;
```

Figure 5-2. Tournament Selection Pseudo Code

As just described, both RTS and DC require a distance measure. This distance measure occurs in the genotype domain in our experiments. The two distance measures, Euclidean and Mahalanobis, that were evaluated are presented in the following section.

5.1.1.2 Distance Measures

Gray coding, Hamming distances and similar algorithms can be used when binary encodings are utilized. However these type of distance measures are not directly applicable to real value encodings. In general, a Euclidean or Hamming distance measured is used in genetic algorithms whenever a distance measure is needed (Herrera, Lozano, & Verdegay, 1998), (Drezewski & Siwik, 2007). This section focuses on two distance measures: Euclidean and Mahalanobis.

The Euclidean distance is the familiar geometric distance based on the Pythagorean formula. This distance measure is relatively simple to calculate using the following formula where x and y are n dimensional vectors representing points:

$$d(x,y) = \sum_{i=1}^{n} \sqrt{(x_i - y_i)^2} \qquad (5\text{-}3)$$

This distance measure has a straightforward geometric interpretation, is computationally inexpensive and simple to code. However it does have two drawbacks expanded upon in the following paragraphs.

First, in geometric problem domain variables are typically measured utilizing the same units of length. Data values from real world problems may have different scales. For example a regression problem making use of class information such as age, test scores and time are all on a different scale and therefore not directly comparable. The Euclidean distance is sensitive to the scales of the variables involved and may not perform optimally. A standardized or weighted Euclidean distance that incorporates variances but not covariances can overcome this problem. A Mahalanobis distance incorporates both variances and covariances.

Second, the Euclidean distance does not compensate for correlated variables. Given a test data set containing multiple variables where one variable set is an exact duplicate of another set, these sets are highly correlated. The Euclidean distance calculation will weigh the duplicate variables more heavily than the others. It has no method of accounting for the fact that the duplicate provides no new information.

Mahalanobis distance was introduced by P.C. Mahalanobis in 1936. It is based on both the mean and variance of the variables in addition to the covariance matrix. The iso-surface formed around the mean is an ellipse in two dimensional space or an ellipsoid or hyper-ellipsoid when more variables are used. It is a multivariate quantitative method that can solve for multiple dimensions simultaneously. The covariance among the variables is taken into account when calculating the distance. Because of this, the problems of scale and correlation inherent in the Euclidean distance are not an issue. Given an individual as a vector $\vec{x}_i = (x_0, \ldots, x_n)$ of floating point values x, a vector representing the mean of a data set $\vec{\mu} = (\mu_0, \ldots, \mu_n)$ and a covariance matrix C of size n x n representing the covariance values between all dimensions n, the Mahalanobis distance is calculated with the given formula:

$$md(\vec{x}_i) = (\vec{x}_i - \vec{\mu})C^{-1}(\vec{x}_i - \vec{\mu})^T \tag{5-4}$$

This function produces a distance value for the $\vec{x}_i$ vector. This vector is either a parent or child individual. The steady state population is used to compute the mean $\vec{\mu}$. In effect the distance measure utilized is not the distance between two vectors but the distance of a vector from the GA population. Hence the population is used as a reference point for all distance measures.

5.1.1.3 Algorithmic Complexity

Computational complexity of the Mahalanobis Distance measure is **O**(n^2) for n dimensional data vectors in the solution population domain (Pinho, Manuel, Tavares, & Correia, 2006). Without any optimizations, the Euclidean distance computational complexity is **O**(n). In this implementation, the Euclidean distance was computed in the genotype domain. Given a minimization problem with 0.0 being the global optimal solution this computational complexity is reduced to a constant time in one dimension.

5.1.2 Experimental Results

The intent of this analysis is to gauge the performance of the distance measures when used in GA crowding multi-modal solutions. The performance of the test runs was measured by three metrics: 1. Peak count = Average number of peaks found. 2. The number of times the global optimum was found in the 100 repeated runs. 3. The average best fitness of the final solution for the 100 repeated runs.

Five functions were used to evaluate performance. These functions have been used frequently in GA evaluations. For completeness these function are described next. The functions were evaluated with 2, 3 and 5 dimension value sets.

Sphere (5-5) is a continuous, convex uni-modal, n-dimensional function constrained to real values -5.12, 5.12. A global minimum occurs at 0.

$$F_s(x) = \sum_{i=1}^{n} x_i^2 \tag{5-5}$$

The generalized Rastrigin (5-6) function is n-dimensional function with a large number of local minima whose value increases with the distance to the global minimum. The function was constrained to real values in the range -1.5, 0.5 where A=10 and w=2π. This limited the number of optima to 5 including the global.

$$F_{ra}(x) = nA + \sum x_i^2 - A\cos(wx_i) \tag{5-6}$$

Ackley (5-7) is a highly multi-modal n-dimensional function. A large number of local minima are spread evenly over the space. One global minimum occurs at 0.

$$F_{ak}(x) = 20 + e - 20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) \tag{5-7}$$

Greiwangk (5-8) has a product term that introduces interdependence among the variables. It is a continuous multi-modal function that has a global optimum at 0.0. In addition it has four relatively large optima at [±π, ±π * 1.414, 0.0, …]. It is constrained to -600,600.

$$F_g(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{400} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{5-8}$$

M6 (5-9) is called the Shekels Foxhole problem (DeJong, 1975). This is a 2-dimensional problem with 25 optima. The two variables are restricted to the range -65.536, 65.535. The maxima are located at (16*i*, 16*j*) where *i* and *j* are integers in the range [-2,2]. They are all of differing heights with a global optimum at (-32, -32). M6 is defined below:

$$F_{m6} = 500 - \frac{1}{.002 + \sum_{i=0}^{24} \frac{1}{1 + i + (x - a(i))^6 + (y - b(i))^6}} \tag{5-9}$$

where $a(i) = 16[(i \bmod 5) - 2]$ and $b(i) = 16(\lfloor i/5 \rfloor - 2)$.

For a solution to be considered as an optima it was not necessary to match the exact value. It should be noted that the goal was minimization and most of the test functions had a global minimum at 0. If all the points of the data vector fell within a small offset from the correct value it was considered to have been a match.

In order to empirically show that the replacement selection method, using two distance measures, was choosing different individuals the specific individual selection results were tracked during a trial run of RTS. At each point in the algorithm, where a distance measure was required, both distance measures were calculated for the two children. In a run with 200 iterations, the individuals selected were identical 10% of the time. 46% of the selections had one individual in common. In the remaining 44% the individuals chosen were unique

A similar process for evaluating replacement selection in DC was implemented. The test consisted of 400 iterations on the five test functions. Given that DC can select to replace at a maximum two parents the total number of possible replacement considerations the algorithm had to make was 4,000. The Mahalanobis distance made a change 2,314 times and Euclidean 2,320. Of these changes all but 152 were identical replacements. This shows that while there were differences in the distance measures they were in agreement on 96% of the decisions.

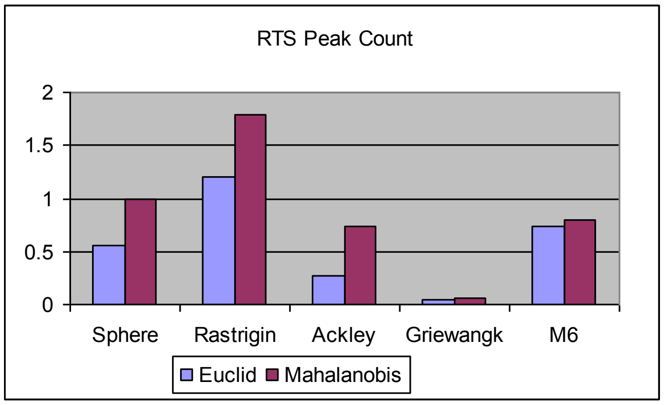


Figure 5-3. RTS Peak Count

Fig. 5-3 contains graphs of the Peak Count for both replacement solutions with a higher value indicating more optimal solutions are found. The data is from the two dimensional test sets. It shows that Mahalanobis matches or outperforms Euclidean for RTS. Performance results for the DC implementation showed on average the performance for both distance measures was equivalent and consequently the results are not shown in the figure.

The number of global optimum discovered in each of the 100 independent runs is found in Table 22. The bold face numbers indicate the better score for each distance measure. If they performed equally well then both values for the distance measure are in

Table 22. Global Optimum Count

	DC		RTS		
Functions	*Euclid*	*Mahal*	*Euclid*	*Mahal*	*Dim.*
Sphere	**100**	**100**	97	**100**	2
Rastrigin	**98**	**98**	**91**	89	2
Ackley	**100**	**100**	70	**95**	2
Griewangk	**20**	11	0	**1**	2
M6	**3**	2	3	**4**	2
Sphere	**100**	**100**	61	**100**	3
Rastrigin	**49**	42	**35**	**35**	3
Ackley	**98**	93	17	**73**	3
Griewangk	0	**1**	0	0	3
M6	2	**3**	2	**3**	3
Sphere	**100**	**100**	5	**97**	5
Rastrigin	0	**1**	1	**3**	5
Ackley	35	**36**	0	**33**	5
Griewangk	0	0	0	0	5
M6	0	**3**	2	**5**	5

bold unless both values are zero. Results for all five test functions in the three different dimensions are available. Out of the fifteen RTS test runs, Mahalanobis scored better eleven times and equivalent once with two instances of both finding no optimums. In the fifteen DC runs, five runs were equivalent, Euclid scored better five times, Mahalanobis scored better five times and there was one instance where no optima were found. This is the second result that shows little difference in performance between the two distance measures in regard to DC output.

Finally Fig. 5-4 depicts the average final best fitness value of the 100 independent trials of the RTS selection function for five dimension solutions. The fitness value is the result of executing a test function with the values of an individual of the population. For this experiment the goal is to find a minimum value. Consequently smaller fitness values are better. Mahalanobis consistently produced better (smaller) results in the four test functions. The M6 test function is defined for only two dimensions and therefore is not shown.

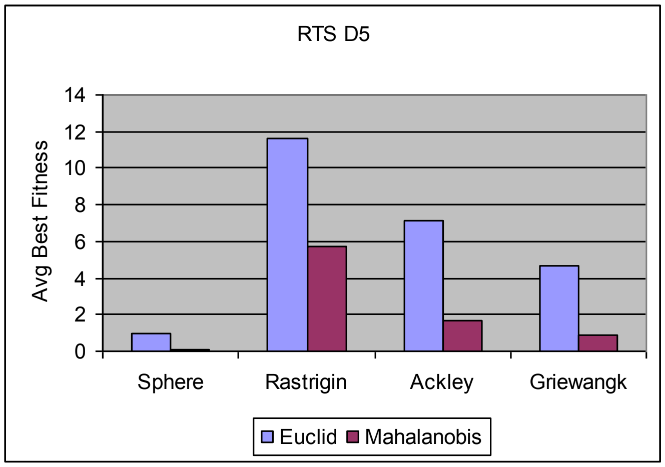


Figure 5-4. Five Dimension RTS

In the DC implementation it was observed that only small differences existed in all of the experiments with the two distance measures. The DC algorithm uses the distance measure (in addition to fitness) as part of a replacement strategy to choose between two parents and two children. The conjecture is that this small population choice accounts for the similar performance when utilizing the different distance functions. In RTS the crowding factor provides a larger population (30 individuals) for distance comparisons. As was shown in the previous section, the two distance measures used in

RTS select at least one different individual 90% of the time. However for DC the difference in selection was only 4%.

5.2 Autonomous Rule Creation for Anomaly Detection

In the voluminous amounts of research on anomaly based intrusion detection systems, little consideration has been given to communication mechanisms (Gogoi, Borah, & Bhattacharyya, 2010). Network based anomaly systems typically produce two types of output. The network traffic may simply be identified as anomalous or not, perhaps with a companion confidence score. Other systems provide a label associating the data with a known attack type i.e. Denial of Service (Vollmer & Manic, 2009). This label is predicated on the system having been trained to identify a specific anomaly.

If the ability to identify a novel intrusion is valuable in one network it seems reasonable that knowledge gleaned from those efforts would be useful in another. Given the numerous different implementations of anomaly detection, and the resulting unique data representations used to find anomalies, there does not appear to be an automated solution to translate the newly discovered detection criteria into a more widely accessible format. A human expert, after arduously examining the traffic and creating a rule, typically performs this knowledge transfer. The rule can subsequently be used as input to any system capable of deciphering the syntax.

There are two fundamental approaches for Network Intrusion Detection Systems (IDS): behavior and rule based. Behavior based systems typically maintain a model of normal system behavior and raise exceptions when parameters fall outside the norm. Computational Intelligence algorithms such as neural networks and clustering have been shown to be effective solutions at identifying anomalous behaviors (Linda, Vollmer, & Manic, 2009), (Taylor & Alves-Foss, 2002). Rule based systems use widely distributable predefined signatures to detect known network issues.

The cost of developing and maintaining rule sets is an important issue for the rule based systems. Human experts are required to create, test and distribute the rules. Given a network trace containing anomalous packets, an expert must investigate the numerous attributes that uniquely identify the attack. This involves laboriously examining the packets for information and creating a candidate rule. Correct rule creation is then a manual process of trial and error where each trial run is examined for a proper alert on a test file of captured network data. Finally, if the expert chooses to do so, the rule is submitted to a rule repository where it may be accepted into a public distribution system.

This section explores a solution to autonomously create IDS rule sets utilizing evolutionary computation techniques. This is accomplished by implementing a Genetic Algorithm (GA) to autonomously create rules from identified network packets that are indicative of system misuse. These packets, used as input, originate from network traffic identified by a behavior based IDS. The resulting rules from processing the packets may not always be optimal for direct distribution but should provide a basis for reducing subsequent expert analysis effort. The system described here can be considered as a one way communication mechanism bridging the two types of intrusion detection systems.

The GA chosen for implementation provides multiple optimal or near optimal unique rules that are made available for further evaluation by a human expert. The effect on accuracy of using different distance measures in this GA was explored in the previous section. It would be a straightforward process to simply provide a single rule that contains all possible attributes of a given network packet. This could lead to over fitting of the rule to a single attack instance. In general, the more specific a rule is the more likely it is to eliminate false positives. However if a rule is too specific it may become brittle in the sense that any minor variation in the attack may be missed. In addition, the more detailed a rule becomes the more computational effort is required to process it.

The solution presented here is similar in approach but different in context to past work in network intrusion detection. Previous work as in (Goyal & Kumar, 2007) was primarily concerned with developing a multiple rule set able to separate known behavior from unknown. Our effort is to produce a set of near optimal IDS rules for a single specific anomalous instance previously detected by a behavior based system. This is accomplished via use of a GA that has several unique characteristics differing from previous research efforts. Specifically, the representation of the population as syntactically correct Snort rules, as opposed to binary or researcher created syntax, and a three part fitness function. This function is designed to optimize the resulting rule sets for the Snort rule engine based on published best practices and characteristics of historical rule repositories.

As a consequence of the population representation a unique distance method was required. The cardinality of the rule genotype attributes is variable. The gene features that may be removed or added to Snort rules are not limited to a fixed number of fields. There are optional fields and most production rules contain a variety of them. Because of this, mutation and crossover required special consideration as well. The presented GA solution accommodates these features.

The input data differs from prior work as well. A single instance of labeled data is presented to the system and the rule set is evolved to detect it. The question to be resolved is can this be done without producing a rule that provides a positive response to network traffic that is not an anomaly (false positive).

5.2.1 Background

Genetic Algorithms are an effective heuristic search technique inspired by concepts of evolutionary biology. They became popular with the published work of John Holland in the 1970's. For an evolutionary algorithm to be categorized as a GA it needs a population representation of possible solutions, variation operators, selection and replacement mechanisms. When optimizing multi-modal functions a conventional GA's population tends to converge to one of the optimal, or near optimal points.

A specific implementation of a GA called Restricted Tournament Selection (RTS) provides a solution for maintaining several optimal solutions in the population (Harik, 1995). This modification is possible because a GA utilizes a population of many (hundreds, thousands or more) possible solutions. This is effective in cases where the fitness function is not capable of representing the fitness with a high fidelity. An evaluation function is used to determine the 'fitness' of individuals in a population. This fitness is a measure indicating how well the individual solves a given problem.

The RTS algorithms ability to maintain several fit solutions will be leveraged to produce unique rules that alert on a given anomalous packet. This assumes that the solution surface for generating rules is not uni-modal. There are several syntax models available for the rule format. Because of its widespread use the Snort rule format was implemented. The first section of this chapter provides more RTS details.

5.2.1.1 Snort Rule Processing

For clarification this section briefly describes how Snort internally processes rules and a required modification to that process. The rule can be seen as a Boolean truth statement. In order for Snort to identify a match, a logical and of all positive field matches is utilized. Upon discovery of a false condition in the *and* evaluation, further processing of that rule is halted.

Snort builds a tree data structure used to compare rule values against packet features. Each mandatory field in a rule is stored in rule tree node (RTN). An OTN (optional tree node) is associated with an RTN and used to store optional rule fields. If multiple rules have the same RTN fields they are only represented by a single node. This optimization feature allows for removal of multiple rules from consideration once a negative match occurs.

The optimization aspect of this rule tree structure implementation is detrimental to our proposed GA fitness algorithm discussed in section 3. If the Snort engine processes a rule section that proves false, that rule is no longer considered for a possible match. This is a logical performance enhancement that speeds the execution of the rule engine. However this short circuit of rule evaluation prevents using a modified Snort source as a basis for rule fitness evaluation.

An attempt was made to encourage the GA to craft good rules to reduce human expert analysis effort spent on examining the rules produced. Good being a subjective term is defined here with respect to three measurements. First the rule has to be able to recognize the packet as being an anomaly. Second it should conform to a grammar checker called dumbpig produced by Ward (2012). This tool parses a rule, reports on badly formatted entries, incorrect usage, and alerts to possible performance issues. Finally the rule should be similar in the number and type of fields used in existing rules. The assumption is that these rules have been vetted by the community of experts and are therefore worthy of emulation.

5.2.2 Algorithm Description

This section describes the implementation details of the final solution. The input data processing and its representation are presented. This data is fed to a GA implemented with RTS. A fitness function implemented in three parts is described. The resulting rules produced by the GA are then sorted by fitness and the top three rules are presented as possible solutions.

5.2.2.1 Input Processing

It is assumed that the anomalous network traffic will already have been identified in advance. Systems, such as those described by Linda, Vollmer and Manic (2009) and Taylor with Alves-Foss (2002), are capable of isolating this kind of traffic. The network traffic data is to be contained in a PCAP formatted file. PCAP data files have become industry standard and are the output of an application programming interface library called libpcap. Utilizing the Perl CPAN module (Net::Pcap) based on this library, the information is read into the program memory space. Having network data stored in files, as opposed to real time capture on a network interface, enables offline processing. However the PCAP library is capable of performing both functions.

After the packets are read into memory each one is parsed and stored in a data structure. For ICMP packets, this structure includes the following: source IP address, destination IP address, ICMP id, type, code, sequence number and packet size. These are all fields used in the GA population representation described in the next section.

5.2.2.2 Genetic Algorithm Description

A pseudo-code implementation of a GA is presented in Fig. 5-3 of the previous section. Details specific to this implementation are presented next.

One of the first tasks in building a GA is to decide upon a representation of the solution population (line 1) and create a number of individuals in that population. Each individual is stored as a representation of a variable length list of Snort rule fields. A Perl associative array maintains these mixed type values. The field key, type and acceptable range values are shown in Table 23. Field key values are taken directly from the Snort rule syntax definitions. The ranges may include values that are not allowed according to specifications but are technically allowable within the data type or are specific to Snort rule processing. For instance, the src and dst fields can contain a variable name that Snort will replace with a configurable value at runtime.

Table 23. Field Values

Field	Type	Range
proto	string	‘icmp’
src	string or CIDR	‘$SNET’ or 0.0.0.0/0
sport	string or integer	‘any’
dst	string or CIDR	‘$DNET’ or 0.0.0.0/0
dport	integer	‘any’
itype	integer	0-255
icode	integer	0-255
icmp_id	integer	0-65535
icmp_seq	integer	0-65535
dsize	integer	0-65535
content	string	Any text

Each individual in this population can be represented using a variable length vector $\vec{v}_i = (x_0, \ldots, x_n)$ of mixed data type values x. The total population is the set of vectors $P_T = \{\vec{v}_0, \vec{v}_1, \ldots, \vec{v}_T\}$. For this experiment, the population size T was fixed at 200. This value was chosen as a reasonable tradeoff between time efficiency and solution convergence. The first generation of individuals was populated by randomly generating values in the domain range of the given fields with a random number of options.

The selection of a field’s inclusion in a rule was not uniformly random in all cases. An analysis of the fields present in the 146 ICMP specific Snort rules was performed. Statistics were compiled on the type and frequency of rule option values present. An assumption was made that these rules, having been vetted and accepted into the official repositories by experts, exhibit desirable characteristics worthy of emulation.

Whenever a decision needs to be made about a field’s inclusion a random number *r* is generated in the range 0 to N where N is the number of rules. Given a frequency count *fc* of a given field f from the set of rules N such that *fc* < cardinality (N), the field *f* is included in a rule if the generated *r* is less than *fc*. In other words, a field is randomly selected for inclusion proportional to the relative frequency of its presence in the original 146 ICMP rules. For example, the dsize field occurred 13 times in the set of 146 rules. Therefore close to a 9% chance exists that this field will be picked for inclusion.

After an initial population is created, a series of variations and replacement selections must take place on some of the population individuals. This process repeats until some acceptable solution or predefined iteration limit is reached. For this project a fixed number of 1000 iterations were selected. After looping (line 2) the defined number of iterations, processing terminated and the fittest individuals maintained by the RTS selection algorithm were isolated as the final best rule set. The details of variation and selection that occur in this loop are presented next.

A steady state population model was implemented. This means that for a given iteration in the loop, only a maximum of two individuals in the original population are selected for replacement. This is in contrast to a generational model where the entire population is replaced by the offspring. In this case, two individuals were selected as parents and two offspring were created from them. Subsequently, two more individuals were then selected for replacement by the new offspring creating a new generation.

The two candidate parent individuals were randomly selected independent of any fitness or distance measure (line 3). Crossover between the two parents to create two new offspring utilized uniform crossover. Creation of these two offspring is a two-step process consisting of rule header creation and rule option creation.

Rule header creation consists of randomly choosing a field from either parent and copying that value into the child. For the given ICMP problem, there is not much variability in the header portion and header creation is not that important relative to the option fields.

Each option field of the new offspring rule is a copy of a field selected from a given parent. For each offspring a primary parent of the original two candidates is selected. Each option field of that primary parent is then considered for inclusion in the new child. A random number from a uniform distribution over 0.0-1.0 is generated. If the number is greater than 0.5 than the primary parents value is utilized; otherwise consideration is given to retrieving the information from the second parent. At this point, if the second parent contains the option field, it is copied into the child. However, if the field does not exist in the second parent, then the default action is to revert to the value from the primary parent. The net result of this is a child that contains the same number of options as the primary parent but with potentially different values from the secondary parent. Variation in child option count is left as a possibility in the mutation operator.

After crossover, mutation occurs on both children prior to replacement selection (line 4). As was the case in crossover, rule header mutation and rule option mutation behave differently.

There is a 25% independent chance of mutating the rule header of a single child. Once selected for mutation the header src or dst IP field is randomly chosen for change. This change consists of randomly choosing between the three available options: any, Snort IP variable ($DNET or $SNET) and the test packet IP address (source or destination as appropriate).

Option mutation is considered separately from header mutation and it has a 25% chance of occurring as well on each option in the child. The reason for this separation of mutation is a result of programming convenience and need not occur in this manner. The integer domain values such as icode, itype, icmp_id and icmp_seq are processed the same according to their domain ranges. A random value within a window bracketing the original value is chosen. If this operation results in a value outside of the acceptable domain range defined in Table 23, the value is set equal to the closest boundary value. Content keyword mutation occurs by randomly selecting a range of the test packets data load and transforming any non printable characters into hex representation. Finally, with a 10% chance, it is possible that any of the options are simply removed from the rule. A future enhancement for consideration would be to add a nonexistent parent option field to the rule.

Once mutation is performed the two resulting offspring are evaluated to replace candidates in the population. This is a critical step for the maintenance of a multi-modal solution set. Uni-modal GA solutions using tournament selection randomly pick two individuals for replacement. These values are replaced if the new children have better fitness values. RTS instead picks from a window size *w*, an individual that is closest to the new child (line 5). The size of the population contained in w is defined by empirical testing and each member is drawn from the original population using a uniformly random selection process. For this project *w* is set at twenty-five. Determination of the best value for *w* was not examined exhaustively and there could be a better value. Closeness is determined by a distance function. After determination of the closest individual to the candidate child a competition is held based on fitness between the child and selected individual. The one with the best fitness is selected for inclusion into the solution population.

5.2.2.3 Three Part Fitness Function

In order to rank the individuals' fitness an evaluation function called the fitness function was defined. The fitness function is a critical component of a GA as it is a primary source for determining an individual's selection for survival and evolution. As was described at the end of the background section, three criteria were identified to judge a 'good' rule. The criteria are described as complete rule match, partial rule match and grammar check. Each criterion is implemented as a function that returns a numeric value. This sum of all three values constitutes the fitness value with a larger value indicating a higher fitness (line 6). Details of each criterion function are presented next.

First a rule should be able to recognize the packet as being an anomaly. This is tested by running Snort with a candidate rule on a test packet and evaluating the result. A system call to the Snort command line was created that sends the output to a comma separated file. This output file is then read for existence of an alert related to the rule. If this exists a value of 10.0 is returned, otherwise the value 0.0 is returned. This relatively large value was chosen to promote the importance of rules that cause an actual alert.

Second the rule is checked for a bad format, incorrect structure and possible performance issues. The Perl based dumbpig grammar checker was incorporated into the project code base to perform this check (Ward, 2012). A function call with the rule data results in a floating point value between 0.0 and 1.0. The fewer issues the function finds the greater the return value.

The final evaluation is executed even when a rule does not completely match an evaluation packet. As was described in the background section, Snort rule evaluation short circuits processing a given rule definition whenever it finds a non match. In response to this behavior, a Snort rule evaluation process was recreated without this aspect. Instead, the processing was modified to track matches on all possible rule fields. For each field a Boolean value is maintained with 1 indicating match and 0 not. After a complete pass evaluating all fields in a rule, the final value is computed according to:

$$F(x) = \sum_{i=1}^{N} match(x_i) \tag{5-10}$$

where N is the number of fields to compare and match is the function that returns the Boolean result of the comparison.

The final fitness evaluation of a candidate rule is then computed as a sum of all three criteria. This fitness value is stored with a reference to the rule and only updated as needed. As this is the final computation step, execution resumes at the beginning of the loop and continues as appropriate.

5.2.2.4 Output Processing

The output of the GA described in the previous section is a set of rules along with their respective fitness values. These rules are sorted according to the fitness values with any duplicate rules removed. The resulting top three (highest fitness value) rules are then proposed as possible rule definitions to be distributed. This assumes that the top rules induce Snort to alert on the related packet. It was found in testing that this occurred in all ten tests cases, with an average 28% of the final rules producing a match.

5.2.3 Experimental Results

This section describes the results of running the system on two test data sets. A subset of the resulting rules that compiled the best fitness (largest) values are presented in addition to information on fitness and algorithm progression.

5.2.3.1 ICMP Test Data

ICMP rules and characteristics were the primary focus of a test data set created to show a proof of concept. Three packet creation tools: Nemesis, packETH and ISIC were utilized to provide two sets of test network data. The first two tools provided customized individual packets designed to trigger specific rules. The third tool, ISIC, was used to create a large set of packets composed of random values to test for false positives. The use of these tools and the resulting test data sets are described in this section.

Nemesis and packETH are network packet crafting and injection tools (Nathan, 2012), (Jemec, 2012). Details of an individual ICMP packet can be specified making them well suited for creating and reproducing test scenarios. They are similar in functionality but packETH features a graphical user interface while Nemesis is a command line tool. An example Nemesis Linux command line used to create a packet is shown below.

```
> nemesis -i 7 -s 0 -d 11 -d lo
```

This command will create an ICMP packet with an itype of 7, a sequence number of 0 and an icmp_id value in the header of 11. Subsequently, the packet will be placed on the lo or loopback interface of the machine. With a packet capture tool attached to this interface, the data can then be captured into a static file and reused for later testing by replaying the file.

The individual test packets created using the tools in the manner just described were captured and stored as a PCAP data file. A total of ten different test packets were created to trigger ten different Snort rules. Each rule was chosen for its rule keyword variety and membership in a rule class. The packet specifics and class types are presented in Table 24 in Snort rule format.

Table 24. ICMP Packet Details

	Packet Details	Class Type
0	icmp_id:667;itype:0;content:"ficken"	attempted-dos
1	same as #1 except random IP's and IP identification field.	attempted-dos
2	dsize:0;itype:8;icode:0	attempted-recon
3	icode:0;itype:5	bad-unknown
4	icode:2;itype:3	misc-activity
5	icode:2;itype:3; content:"|28 00 00 50 00 00 00 00 F9 57 1F 30 00 00 00 00 00 00 00 00 00 00 00 00|"	attempted-user
6	icode:0;itype:8;dsize:20; content:"abcde12345fghij6789";	trojan-activity
7	itype:8;icode:0;dsize:32;content:"abcdefghijklmnopqr|0 000|";depth:22;	trojan-activity
8	icmp_id:123;icmp_seq:0;itype:0; content:"shell bound to port";	attempted-dos
9	icode:0; itype:40;	misc-activity

Snort was exercised with the ten hand crafted test packets against 146 ICMP rules from the three sources mentioned in the background section. These test packets were crafted to match all conditions of ten specific rules that were categorized into a variety of rule classes. Each individual packet was run against the original set of rules to ensure that a one to one relationship of packet to rule existed. In other words a single packet matched with a single rule. However this was not entirely possible as some of the rule definitions are broad enough to alert on only a few attributes. For instance test packet 8 triggered an alert on a rule that only specified itype 8 and icode 0. This is a generic ping rule and is technically correct. This type of issue aside Snort positively identified 100% of the test packets with no false positives or false negatives.

A second large test set of random valued ICMP packets was created to evaluate the system for false positives. ISIC is a generic utility used to test the stability of an IP stack (Frantzen & Xiao, 2012). It is capable of creating a large number of test packets containing random values. The random values are correct in that they fall within a field's acceptable data range. However, the value may not be currently in use or match a prerequisite implied by a setting in another field. These packets are then typically sent to a target while observing for any anomalous results. For this project 23 megabytes of data containing 33,794 ICMP packets were created.

In the same manner as was described in the hand crafted packet test creation, the random test set was run against Snort and the original rules. This resulted in a total of 31,929 alerts. Eighty-eight percent (28,293) of the alerts were triggered by a single rule designed to find undefined codes. The remaining 3,636 alerts were produced by 51 unique rules. Of this set, 6 alerts were generated from 4 rules that were used to craft matches in the first test set. These were carefully noted for the evaluation phase as genuine alerts. They should be ignored when testing for false positives, as indeed they are not.

After generating the two data sets, the original ten rule definitions used for the hand crafted packets were saved. These rules then became one basis for the final evaluation of the evolved rules correctness. It was not expected that the created rules exactly match the originals but they should be similar in content and behavior.

5.2.3.2 Complexity Analysis

Rule generation in the manner described was considered to be an offline process so initially minimal consideration was given to runtime performance. However the complexity of the RTS GA is N x *w* where *w* is the size of the selection window. The complexity analysis of our implementation is complicated by the three part fitness test. Specifically the call to the Snort executable is an unknown quantity. The runtime performance of Snort varies greatly depending upon the nature of the rule set and volume of network traffic. In this project, the observed process run time of just the Snort binary was less than 1 second in all test cases on a desktop DELL with an Intel E5430 CPU and 4 GB's of RAM. The average run time of the test cases was 57 seconds. This reflects the heavy usage of string manipulation procedures. In addition, the project was implemented in PERL with a focus on program correctness and process visibility instead of runtime performance.

5.2.3.3 Test Results

Table 25. Fitness and Run Data

	Initial Avg Fitness	Final Avg Fitness	Best Fitness	Replace
P0	5.05	7.95	31	145
P1	5.23	10.49	31	207
P2	2.5	8.92	22	185
P3	2.63	9.06	22	191
P4	2.6	10.35	22	224
P5	4.6	10.24	30.6	209
P6	5.14	11.5	30.6	258
P7	5.22	11.86	30.72	270
P8	4.97	9.12	26.47	180
P9	2.46	9.54	22	218

In order to show the progression of the algorithms search for a set of optimal rules, Table 25 provides average fitness information, largest final rule fitness and the number of times a child was used to replace a member of the population. The columns are labeled with a test packet identifier of P0-P9. It can be seen in all test cases that the average final fitness values are higher than the initial average fitness. The fitness value of the final best solution, or rule, in each case is approximately two or three times that of the final average. The child creation step did produce an individual, on average, every five iterations that improved fitness as indicated by the replacement count.

```
235.130.217.126/32 any -> any any (itype:0; content:"ficken|0A|"; icode:0;)
$SNET any -> 140.53.42.24/32 any (itype:0; content:"cken";)
$SNET any -> any any (content:"fick"; dsize:7;)
(src) $HOME_NET any -> $EXTERNAL_NET any  (icmp_id:667; itype:0; content:"ficken";)
```

Figure 5-5. Packet 0 Generated Rules

Fig. 5-5 shows the top three fittest individuals for test case P0. For clarity the original rule definition used to create the test packet is included as the last line and is labeled as src. For formatting reasons the action, protocol, sid, classtype and rev number have been removed. These values are metadata information and do not contribute to Snorts recognition engine execution. As can be seen there is a variety in the rule header composure. The specific IP addresses were retrieved from the test packet headers. Retaining these in a production rule may not add value but this depends on knowledge not used as input into this system. A rule option commonality can be observed in the inclusion of similar content values and itype fields. The first rule contains an icode field that does not appear in the original rule. As this field was not identified in the original rule a default value was supplied when creating the packet. This value was manually verified as being correct for the test packet. Because of space considerations the other top three rules generated from each of the remaining nine cases are not shown.

A simple test of rule correctness involved running the ten test anomaly packets against the corresponding set of top three generated rules. All of the packets were recognized and generated the appropriate Snort alerts for a 100% positive identification rate for each of the top three rules for the 10 tests. This was expected based on the composition of the fitness evaluation function. It should be noted that all three of the rules contain valid identifying fields. A post-processing step could involve any combination of the defined fields to create a valid rule. A possibly more interesting test concerns the occurrence of false positives. It has been observed that a simple rule definition could contain just one field that matches a large number of packets. This would certainly produce a large positive identification rate. A set of random ICMP test packets was created to test for this scenario. In addition to the random packets, the ten hand crafted packets were used since they were readily available. For a given rule generated by our algorithm, only one of the crafted packets should generate an alert.

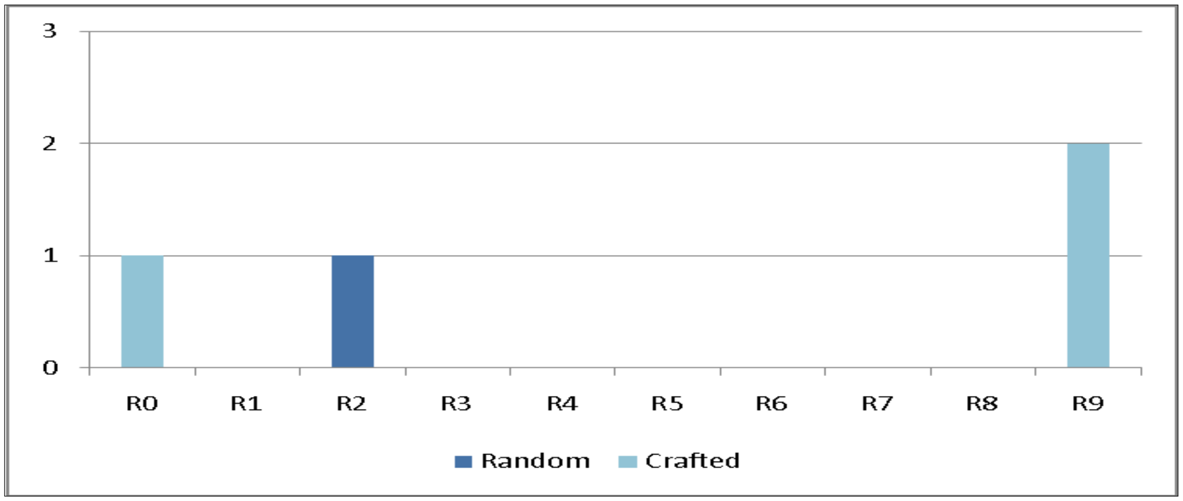


Figure 5-6. False Positive Rate for Rules

As can be seen in Fig. 5-6 the false positive rate was very low. The vertical axis indicates the number of false positives for a given test rule set. A total of four false positives were generated from the set of 30 generated rules over the more than 33,000 test packets. Three false positives came from the hand crafted packets. None of the false positives were generated from the single top fittest rule for each test case.

5.3 Chapter Summary

This chapter provides two contributions: 1) A discussion and evaluation of similarity measures in two multi-modal GA solutions. 2) The implementation of a multi-modal GA to autonomously create static rule sets from anomalous network traffic. The algorithm presents multiple rule solutions for distribution thereby reducing the effort required by human experts.

Two crowding type Genetic Algorithm multi-modal solutions with real coded values were evaluated: Restricted Tournament Selection and Deterministic Crowding. The use of Mahalanobis and Euclidean distance measures in selection determination was evaluated. Five frequently used test functions were implemented as benchmarks to evaluate the performance of the selection routines.

Mahalanobis is computationally more expensive but, within the parameters of the experiment, appears to be superior to Euclidean distance in Restricted Tournament selection and equivalent in Deterministic Crowing. The use of Mahalanobis distance, in the case of Restricted Tournament Selection, found 40% more of the global optimum, maintained a 35% higher average peak count and produced an average final fitness that was 3 times better (lower). As is often the case, the results depict a tradeoff of computational complexity versus performance when choosing a distance algorithm to implement.

Finally, the last section devised a solution to decrease human effort in creating Snort rules based on anomalous network traffic was accomplished. All of the ten hand crafted test packets resulted in a set of rules that caused Snort to alert. Not all of the rules were unique, but in each case there were at least three unique rules and as many as eight. Testing showed that these rules were specific to the packets and produced only four false positives from 33,804 test packets. These successful results indicate that analysts can use the generated rules as the basis for production rules.

A key enabling technology was the use of a multi-modal GA introduced in the first section. A critical feature, in the performance of a GA's search capability, is the ability of the fitness function to accurately indicate progress in exploration of the solution set. In the last section a three part fitness function was developed. It was sufficient in aiding generation of rules that caused alerts on test cases. Further refinement of this function and the addition of more rule options may increase the capability of the system to define robust rules. In addition, defining the capability to include session information instead of single packet rules could be explored. Finally expanding the domain of the test packets to UDP and TCP would provide a broader coverage of the anomalous network possibilities.

Chapter 6. Conclusion and Future Work

This dissertation focuses on the problem of protecting control system devices attached to Ethernet networks. Utilizing standardized communication mechanisms and computational intelligence techniques in an autonomic computing structure; the ability to detect and mitigate abnormal behaviors was demonstrated. Further, despite the addition of complex algorithms, the resulting autonomic based system simplified the expression of information to the human operator.

6.1 Summary of Contributions

Summaries of this dissertations contribution by chapter are:

Chapter 3 introduced the Autonomic Intelligent Cyber Sensor (AICS) architecture. AICS utilizes concepts of autonomic computing and a SOAP based IF-MAP external communication layer to create a network security sensor. This approach simplifies integration of legacy software and supports a secure, scalable, self-managed framework of modules. The contributions of this chapter include:

1. A network security and information framework that utilizes a two level flexible communication platform.
2. Intelligent Anomaly Assessment module (IAA): A custom Fuzzy Type 2 implementation that uses clustering and a sliding window technique to model system behavior.
3. Network Entity Identification (NEI): A network host identification scheme using Ettercap and custom logic to continuously examine traffic and maintain a dynamic model of the network.
4. Dynamic virtual Honey Pot (DHP): Information from the NEI model is utilized to automatically create dynamic virtual honeypots with Honeyd. This relieves system administrators from time-consuming configuration duties.

Chapter 4 discussed two communication algorithms for security awareness. The contributions of this chapter are:

1. A Human Interface for Security Awareness (HISA) algorithm for interpreting cyber incident information from anomaly based intrusion detections systems is developed. The similarity algorithm maps anomaly results from behavior

systems to signature based intrusion system rules. Categorizations of attacks found in rules created for the Snort intrusion system were used as a basis of information to present to the user.

2. The second contribution is the Computationally efficient Neural Network Intrusion Security Awareness (CeNISA) algorithm. The work presented in this section builds on the HISA algorithm. An entirely new and improved solution to the same problem is detailed. CeNISA is a neural network based pattern recognition algorithm that maps novel attack vectors recognized by anomaly Intrusion Detection Systems (IDS) to known attack classifications.

Chapter 5 presented an algorithm for autonomously creating snort rules based on identified anomalous network traffic. The contributions of this chapter are:

1. An exploration of distance measures used in multi-modal genetic algorithms. The basic idea of a multi-modal GA is to encourage the evolution of subsets representing diverse solutions in a single population during the evolutionary process. In order to measure this diversity, distance measures are employed. Given better distance measures, improved results may be realized.
2. The second chapter section explores a solution to autonomously create IDS rule sets utilizing evolutionary computation techniques. This is accomplished by implementing a multi-modal GA to autonomously create rules from identified network packets. These packets, used as input, originate from network traffic identified by a behavior based IDS such as the IAA component.

6.2 Future Work

This work has identified several areas of possible future research. The use of virtualized networks and devices derived from the automated system presented could subsequently be used as a standard test bed for a variety of IDS systems. This kind of virtualized system is easily transportable and would provide a consistent background for testing. Subsequent work would be able to appropriately conduct comparisons with related efforts.

One category of future work would be correcting the deficiencies found in the open source support software used in AICS. For instance, the virtual honeypot software

Honeyd could benefit from the handling of IP options. Additionally, network service emulation scripts for Honeyd are manually created. Many control system vendors offer windows based simulations of their controllers for users to test implementations. Integration of these implementations and development of autonomous service behaviors that emulate observed network communications would further the goals of deception, repeatable testing and reduction of human efforts.

The distributed nature of multiple sensors could be more fully exploited. This would include integration of AICS with external components utilizing the IF-MAP standard. These components could include functionality to correlate information from the sensor with physical production data. Distributed sensors could share knowledge in the form of rules and network behaviors. In the particular case of Genetic Algorithms, this could potentially reduce the amount of training needed for a new deploy.

References

(2012, April). Retrieved August 15, 2012, from Nessus webpage: http://tenable.com/products/nessus

(2012, April). Retrieved August 15, 2012, from Nmap webpage: http://nmap.org

Agrawal, R., Grosky, W., & Fotouhi, F. (2009). Virus detection and removal service architecture in digital ecosystems. *IEEE Conference on Digital Ecosystems and Technology* (pp. 301-305). IEEE.

Aickelin, U., Twycross, J., & Hesketh-Roberts, T. (2007). Rule generalisation in intrusion detection systems using SNORT. *International Journal Electronic Security and Digital Forensics, 1*(1), 101-116.

Analoui, M., Bidgoli, B. M., & Rezvani, M. H. (2007). Hierarchical two-tier ensemble elarning: a new paradigm for network intrusion detection. *ACM first Ph. D. workshop in CIKM* (pp. 33-40). ACM.

Auffret, P. (2010, Aug). SinFP, Unification of Active and Passive Operating System Fingerprinting. *Journal of Computer Virology, 6*(3), 197-205.

Barber, D. (2012). *Bayesian Reasoning and Machine Learning.* New York, New York, USA: Cambridge University Press.

Beglarbegian, M., Melek, W., & Mendel, J. M. (2011, April). On the robustness of Type-1 and Type-2 fuzzy logic systems in modeling. *Information Sciences, 181*(7), pp. 1325-1347.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* New York: Oxford Press.

Brodal, P. (1998). *The Central Nervous System.* Oxfort University Press.

Candido, G., Colombo, A. W., Barata, J., & Jammes, F. (2011, November). Service-Oriented infrastructure to support the deployment of evolvable production systems. *IEEE Transactions on Industrial Informatics*, 759-767.

Caswell, B. (2012, April). *Perl-Net-Snort-Parser.* Retrieved August 15, 2012, from http://projects.honeynet.org

Chandola, V., Banerjee, A., & Kumar, V. (2007). *Anomaly Detection: A survey.* University of Minnesota.

Cheminod, M., Durante, L., & Valenzano, A. (2013, February). Review of security issues in industrial networks. *IEEE Transactions on Industrial Informatics, 9*(1), 277-293.

Cheng, Y., Leon-Garcia, A., & Foster, I. (2008). Toward an autonomic service management framework: A holistic vision of soa, aon, and autonomic computing. *IEEE Communications Magazine, 46*(5), pp. 138-146.

de Deugd, R., Carroll, R., Key, K. E., Millett, B., & Ricker, J. (2006, July). SODA Service Oriented Device Architecture. *IEEE Pervasive Computing, 5*(3), pp. 94-96.

DeJong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* Unviersity of Michigan.

Denning, D. E. (1987, February). An intrusion detection model. *IEEE Transactions on Software Engineering, SE-13*, 222-232.

Deri, L. (2012, April). *ntop*. Retrieved August 15, 2012, from http://www.ntop.org

Digital Bond Incorporated. (2012, April). *SCADA Honeynet*. Retrieved August 15, 2012, from http://www.digitalbond.com/tools/scada-honeynet

Dobson, S., Denazis, S., Fernandez, A., Gaiti, D., Gelenbe, E., Massacci, F., et al. (2006). A survey of autonomic communications. *Transactions on Autonomous and Adaptive Systems, 1*(2), 223-259.

Drezewski, R., & Siwik, L. (2007). Techniques for maintaining population diversity in classical and agent-based multi-objective evolutionary algorithms. *7th International Conference on Computational Science.* Beijing.

Duggan, D. P., Berg, M., Dillinger, J., & Stamp, J. (2005). *Penetration testing of industrial control systems.* Sandia National Laboratories. Sandia.

Dussel, P., Gehl, C., Laskov, P., Buber, J., Stormann, C., & Kastner, J. (2010). Cyber-critical infrastructure protection using real-time payload-based anomaly detection. *4th Workshop on Critical Information Infrastructures Security* (pp. 85-97). Berlin: Springer.

Eckmann, S. T. (2001). Translating snort rules to STATL scenarios. *Recent advances in intrusion detection.*

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing.* SpringerVerlag.

Emerging Threats. (2009, 01 01). *www.emergingthreats.net*. Retrieved 05 01, 2009, from www.emergingthreats.net: www.emergingthreats.net

Emerging threats. (2012, April). Retrieved August 15, 2012, from www.emergingthreats.net

Ettercap network sniffer. (2012, April). Retrieved August 15, 2012, from http://ettercap.sourceforge.net

Fleming, P. J., & Purshouse, R. C. (2002). Evolutionary algorithms in control systems engineering: a survey. *Control engineering practice, 10*(11), pp. 1223-1241.

Fonseca, R., Porter, G., Katz, R. H., Shenker, S., & Stoica, I. (2005). *IP Options are not an option.* UC Berkeley.

Frantzen, M., & Xiao, S. (2012, April). *ISIC IP Stack Integrity Checker*. Retrieved August 15, 2012, from http://isic.sourceforge.net

Gagnon, F., & Esfandiari, B. (2011, Mar). A hybrid approach tooperating system discovery based on diagnosis. *International Journal of Network Management, 21*, 106-119.

Garcia-Teodoro, P., Diaz-Verdejo, J. E., Macia-Fernandez, G., & Sanchez-Casad, L. (2007, October). Network-based hybrid intrusion detection and honey systems as active reaction schemes. *International Journal of Computer Science and Network Security, 7*(10), 62-70.

Gellman, B. (2002, June 26). *Cyber-Attacks by Al Qaeda Feared.* Retrieved May 1, 2009, from www.washingtonpost.com: www.washingtonpost.com/ac2/wp-dyn/A50765-2002

Gogoi, P., Borah, B., & Bhattacharyya, D. K. (2010, Feb). Anomaly detection analysis of intrusion data using supervised and unsupervised approach. *Journal of Convergence Informatino Technology, 5*(1).

Goldberg, D. E., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. *2nd International Conference on Genetic algorithms and Applications.* Cambridge.

Gomez, J., Dasgupta, D., Gomez, J., & Kaniganti, M. (2003). An evolutionary approach to generate fuzzy anomaly signatures. *IEEE Information assurance workshop.* IEEE.

Gosh, A. K., Schwartzbard, A., & Schatz, M. (1999). Learning program behavior profiles for intrusion detection. *1st USENIX workshop on intrusion detection and network monitoring* (pp. 51-62). Santa Clara: USENIX.

Goyal, A., & Kumar, C. (2007). *GA-NIDS: A genetirc algorithm based network intrusion detection system.* Northwestern University, Evanston.

Gungor, V. C., & Hancke, G. P. (2009, October). Industrial wireless sensor networks: Challenges, design principles, and technical approaches. *IEEE Transactions Industrial Electronics, 56*(10), 4258-4265.

Gungor, V. C., Lu, B., & Hancke, G. P. (2010, October). Opportunities and challenges of wireless sensor networks in smart grid. *IEEE Transactions on Industrial Electronics, 57*(10), 3557-3564.

Harik, G. R. (1995). Finding multimodal solutions using restricted tournament selection. *6th International Conference on Genetic Algorithms*, (pp. 24-31). San Francisco.

Hay, A., CId, D., & Bray, R. (2008). *OSSEC HIDS Host-based intrusion detection guide.* Burlington, MA: Elsevier.

Hecker, C., & Hay, B. (2010). Securing E-Government Assets through automating deployment of Honeynets for IDS Support. *43rd Hawaii International Conference on System Sciences*, (pp. 1-10).

Hecker, C., Nance, K. L., & Hay, B. (2006). Dynamic honeypot construction. *10th Coll. for information systems security education.* Adelphi.

Herrera, F., Lozano, M., & Verdegay, J. L. (1998). Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial intelligence review 12*, pp. 265-319.

Hieb, J., & Graham, J. H. (2004). *Anomaly-Based Intrusion Detection for network monitoring using a dynamic honeypot.* Intelligent Systems Res. Lab. University of Louisville.

Hu, W., Liao, Y., & Vemuri, V. R. (2003). Robust Anomaly Detection Using Support Vector Machines. *International Conference on Machine Learning*, (pp. 282-289).

Huang, Y., & Wenke, L. (2003). A cooperative intrusion detection system for ad hoc networks. *1st ACM workshop on security fo ad hoc and sensor networks*, (pp. 135-147). Fairfax.

IBM. (2006, June). *An architectural blueprint for autnomic computing 4th edition.* Retrieved 08 15, 2012, from http://www-03.ibm.com/autonomic/pdfs/AC Blueprint White Paper 4th.pdf

Internet Protocol. (1981, September). Retrieved August 15, 2012, from DARPA RFC 791: http://www.ietf.org/rfc/rfc791.txt

Iwao, T., Yamada, K., Yura, M., Nakaya, Y., Cardenas, A., Lee, S., et al. (2010). Dynamic Data Forwarding in Wireless Mesh Networks. *IEEE Smart Grid Communications* (pp. 385-390). IEEE.

Jemec, M. (2012, April). *packetETH - Ethernet packet generator*. Retrieved August 2012, from http://packeth.sourceforge.net

Jiang, X., & Xu, D. (2004). BAIT-TRAP: A Catering Honeypot Framework. Purdue University.

Khan, L., Awad, M., & Thuraisingham, B. (2007). A new intrusion detection system using support vector machines and hierarchical clustering. *The international Journal on Very Large Databases, 16*(4), 507-521.

Klir, G. J., & Yuan, B. (1995). *Fuzzy Sets and Fuzzy Logic Theory and Applications.* New York: Prentice Hall.

Kyusakov, R., Eliasson, J., Delsing, J., Deventer, J., & Gustafsson, J. (2011). Integration of wireless sensor and actuator nodes with IT infrastructure using service-oriented architecture. *IEEE Transactions on Industrial Informatics*.

Lee, W., Stolfo, S. J., & Mok, K. W. (1999). Mining in a data-flow environment: experience in network intrusion detection. *International conference on Knowledge discovery and data mining* (pp. 114-124). ACM.

Linda, O., Vollmer, T., & Manic, M. (2009). Neural Network Based Intrusion Detection for Critical Infrastructures. *International Joint Conference on Neural Networks* (pp. 55-62). Atlanta: IEEE.

Linda, O., Vollmer, T., & Manic, M. (2012). Improving Cyber-Security of smart grid systems via anomaly detection and linguistic domain knowledge. *5th IEEE Symposium on Resilient Control Systems.* Salt Lake CIty: IEEE.

Linda, O., Vollmer, T., Wright, J., & Manic, M. (2011). Fuzzy Logic Based Anomaly Detection for embedded network cyber sensor. *IEEE Symposium Series on Computational Intelligence* (pp. 202-209). Paris: IEEE.

Long, J., Schwartz, D., & Stoecklin, S. (2006). Distinguishing False from True alerts in Snort by data mining patterns of alerts. *SPIE Defense and Security Symposium.* Orlando.

Lyon, G. (2008). *Nmap Network Scanning.* Sunnyvale, CA: Insecure.com.

Macia-Perez, F., Mora-Gimeno, F., Marcos-Jorquera, D., GilMartinez-Abarca, H., Ramos-Morillo, H., & Lorenzo-Fonseca, I. (2011, March). Network intrusion detection system embedded on a smart sensor. *IEEE Transactions on Industrial Electronics, 58*(3), 722-732.

Mahfoud, S. W. (1995). *Niching methods for genetic algorithms.* PhD Thesis, University of Illinois.

McQueen, M. A., & Boyer, W. F. (2009). Deception used for cyber defense of control systems. *2nd IEEE Conference on Human System Interaction* (pp. 624-631). IEEE.

Mendel, J. M. (2001). *Uncerain Rule-Based Fuzzy Logic Systems: Introduciton and New Directions.* Upper Saddle River, NJ: Prentice Hall PTR.

Mendel, J. M., John, R., & Liu, F. (2006, December). Interval Type-2 Fuzzy Logic Systems Made Simple. *IEEE Transactions on Fuzzy Systems, 14*(6), 808-821.

Merentitis, A., Patouni, E., Alonistioti, N., & Doubrava, M. (2008). To reconfigure or not to reconfigure: Congnitive mechanisms for mobile devices deicsion making. *IEEE Vehicular Technology Conference* (pp. 1-5). IEEE.

Meserve, J. (2007). *Sources: Staged cyber attack reveals vulnerability in power grid.* Retrieved April 2012, from CNN: http://www.cnn.com/2007/US/09/26/power.at.risk/

Mishra, A., Pandey, M. D., & Chauhan, A. (2009). Regulation of nuclear power plants a multi objective approach. *Annals of Nuclear Energy, 36*(10), 1560-1573.

Mukkamal, S., Yendrapalli, K., Basnet, R., Shankarapani, M. K., & Sung, A. H. (2007). Detection of Virtual Environments and Low Interaction Honeypots. *Information Assurance and Security Workshop*, (pp. 92-98).

Natalia, F., Hugo, A., & Raul, G. (2000, October). Crowding under diverse distance criteria for niche formation in multimodal optimization. *Journal of computer science and technology*.

Nathan, J. (2012, April). *Nemesis*. Retrieved August 15, 2012, from http://nemesis.sourceforge.net

P0f v3. (2012, April). (M. Zalewski, Producer) Retrieved August 15, 2012, from icamtuf.coredump.cs/p0f3

Pacific Northwest Smart Grid Demonstration Project - 2011 Annual Report. (2012, April). Retrieved August 2012, from http://www.pnwsmartgrid.org/publications.asp

Pinho, R. R., Manuel, J., Tavares, R. S., & Correia, M. V. (2006). Efficient approximation of the Mahalanobis distance for tracking with the Kalman filter. *CompIMAGE*, pp. 84-92.

Pothamsetty, V., & Franz, M. (2012, April). *SCADA Honeynet Project*. Retrieved August 15, 2012, from http://scadahoneynet.sourceforge.net

Provos, N., & Holz, T. (2007). *Virtual Honeypots.* Boston, MA: Addison-Wesley.

Rajkumar, N., Vekara, T., & Alander, J. T. (2008). *A review of genetic algorithms in power engineering.* University of Vassa.

Rieger, C. G., Moore, K. L., & Baldwin, T. L. (2013). Resilient control systems: A multi-agent dynamic systems perspective. *2013 IEEE International Conference on Electro/Information Technology* (pp. 1-16). Rapid City, SD: IEEE.

Rockwell Automation. (2005, August). Micrologix Ethernet Interface User Manual.

Roesch, M. (1999). Snort-lightweight intrusion detection for networks. *Proceedings of the 13th USENIX conference on System Administration* (pp. 229-238). Seattle: USENIX.

Roesch, M. (2001, 01 01). *Writing Snort Rules: How to write Snort rules and keep your sanity*. Retrieved 05 15, 2009, from www.snort.org: http://www. snort. org/writing_snort_rules.htm

Ruan, H., Lacoste, M., & Leneutre, J. (2010). A policy management framework for self-protection of pervasive systems. *6th International Conference on Autonomic and Autonomous Systems* (pp. 104-109). Cancun: IEE.

Rueff, G., Roybal, L., & Vollmer, T. (2013). *SCADA Protocol Anomaly Detection Utilizing Compression (SPADUC) 2013.* Idaho Falls: Idaho National Laboratory.

Shea, D. A. (2004). *Critical infrastructure: control systems and the terrorist threat.* Library of Congress.

Sherry, J. (2010). Applications of the IP Timestamp Option to Internet Measurement. Dept. Comp. Sci. and Eng. University of Washington.

Shuaib, H., Anthony, R. J., & Pelc, M. (2010). *Towards certifiable autonomic computing systems.* University of Greenwich, Autonomic Research Group. Greenwich: CMS.

Sinclair, C., Pierce, L., & Matzner, S. (1999). An application of machine learning to network intrusion detection. *15th annual computer security applications conference.*

Sommer, R., & Paxson, V. (2003). Enhancing byte-level network intrusion detection signatures with context. *Proceedings of the 10th ACM conference on Computer and Communications Security* (pp. 262-271). ACM.

Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. *IEEE Symposium on Security and Privacy* (pp. 305-316). IEEE.

Sourcefire Inc. (2009, September 15). *Snort Users Manual.* Retrieved October 1, 2009, from snort.org: www.snort.org

Stein, G., Chen, B., Wu, A. S., & Hua, K. A. (2005). Decision tree cliassifier for network intrusion detection with GA-based feature selection. *43rd ACM Southest Regional Conference* (pp. 136-141). Kennesaw: ACM.

Stevens, W. R. (2003). *TCIP/IP Illustrated* (Vol. 1). Redwood City, CA: Addison Wesley.

Stolfo, S. J., Lee, W., Chan, P. K., Fan, W., & Eskin, E. (2001). Data mining-based intrusion detectors: an overview of the colombia IDS project. *30(4)*, 5-14. ACM.

Taylor, C., & Alves-Foss, J. (2002). An empirical analysis of NATE: Network Analysis of Anomalous Traffic Events. *2002 workshop on new security paradigms*, (pp. 18-26).

Taylor, C., Oman, P., & Krings, A. (2003). Assessing power substation network security and survivability: A work in progress report. *International Conference on Security and Management*, (pp. 23-26). Las Vegas.

Tcpdump packet analyzer. (2012, April). Retrieved August 15, 2012, from http://www.tcpdump.org

The Edison Foundation. (2010, Apr). *Utility Scale Smart Meter Deployments, Plans & Proposals.* Retrieved August 15, 2012, from http://www.edisonfoundation.net

The Edison Foundation. (2013). *Utility-Scale Smart Meter Deployments.* IEE.

Trust@FHH. (2012, April). *IROND*. Retrieved August 15, 2012, from http://trust.inform.fh-hannover.de/joomla/index.php

Trusted Network Computing Group. (2010, September). *TNC IF-MAP Metadata for Network Security.* Retrieved August 15, 2012, from http://www.trustedcomputinggroup.org/resources/tnc_ifmap_metadata_for_network_security

Trusted Network Computing Group. (2011, November). *TNC IF-MAP Binding for SOAP.* Retrieved from http://www.trustedcomputinggroup.org/resources/tnc_ifmap_binding_for_soap_specification

Tshark Network Analyzer. (2012, April). Retrieved August 15, 2012, from http://www.wireshark.org

Ulieru, M., & Grobbelaar, S. (2007). Engineering Industrial Ecosystems in a Networked World. *5th IEEE International Conference on Industrial Informatics* (pp. 1-7). Fredericton: IEEE.

Valente de Oliveira, J., & Pedrycz, W. (2007). *Advances in fuzzy clustering and its applications.* John Wiley & Sons, Ltd.

Vollmer, D. T., & Manic, M. (2009). Computationally efficient neural network intrusion security awareness. *IEEE Second International Symosium on Resilient Control Systems.* Idaho Falls: IEEE.

Vollmer, D. T., & Manic, M. (2009). Human interface for cyber security anomaly detection systems. *IEEE Second Conference on Human Systems Interactions.* Catania, Italy: IEEE.

Vollmer, D. T., Alves-Foss, J., & Manic, M. (2012). Autonomous rule creation for intrusion detection. *IEEE 2011 Symposium on Computational Intelligence in Cyber Security.* Paris: IEEE.

Vollmer, D. T., Linda, O., & Manic, M. (2013). Autonomic intelligent cyber sensor to support control network situational awareness. *IEEE Transactions on Industrial Informatics*.

Vollmer, D. T., Soule, T., & Manic, M. (2010). A distance measure comparison to improve crowding in multimodal problems. *Third International Symposium on Resilient Control Systems.* Idaho Falls: IEEE.

Vollmer, T., & Manic, M. (2009). Computationally efficient neural network intrusion security awareness. *2nd IEEE Symposium on Resilient Control Systems.* Idaho Falls: IEEE.

Vollmer, T., & Manic, M. (2013). Adaptable autonomous virtual hosts in an industrial control network. *IEEE Transactions on Industrial Informatics*.

Ward, L. (2012, April). *dumbpig*. Retrieved August 15, 2012, from http://leonward.wordpress.com/dumbpig

Witten, I. H., & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann.

Xu, R., & Wunsch, D. C. (2009). *Clustering.* Hoboken, NJ, USA: John Wiley & Sons.

Xu, R., & Wunsch, D. C. (2009). *Clustering.* Hoboken: John Wiley and Sons, Inc.

Yang, Y., Xu, Y., Li, X., & Chen, C. (2011, June). A loss Inference algorithm for wireless sensor networks to improve data reliability of digital ecosystems. *IEEE Transactions on Industrial Electronics, 58*(6), 2126-2137.

Zadeh, L. A. (1965). Fuzzy Sets. *Information and Control, 8*, 338-353.

Zadeh, L. A. (1975). The concept of a linguistic variable and its approximate reasoning. *Information Sciences*(8), pp. 301-357.

Zanero, S., & Savaresi, S. (2004). Unsupervised learning techniques for an intrusion detection system. *2004 ACM sysmposium on applied computing* (pp. 412-419). ACM.

Zhong, S., Khoshgoftaar, T., & Seliya, N. (2007). Clustering-based network intrusion detection. *International Journal of reliability, quality and safety, 14*(2), 169-187.