

University of Louisville

ThinkIR: The University of Louisville's Institutional Repository

Electronic Theses and Dissertations

5-2006

ARF : an Automated Real-Time Fuzzy Logic Threat Evaluation System.

Jeremy D. Gray
University of Louisville

Follow this and additional works at: <https://ir.library.louisville.edu/etd>

Recommended Citation

Gray, Jeremy D., "ARF : an Automated Real-Time Fuzzy Logic Threat Evaluation System." (2006). *Electronic Theses and Dissertations*. Paper 526.
<https://doi.org/10.18297/etd/526>

This Master's Thesis is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.

ARF: An Automated Real-Time Fuzzy Logic Threat Evaluation System

By

Jeremy D. Gray
B.S. University of Louisville, 2005

A Thesis
Submitted to the Faculty of the
University of Louisville
Speed School of Engineering
in Partial Fulfillment of the Requirements
for the Professional Degree of

MASTER OF ENGINEERING

Department of Computer Engineering and Computer Science
University of Louisville

May 2006

Copyright 2006 by Jeremy D. Gray

All Rights Reserved

ARF: An Automated Real-Time Fuzzy Logic Threat Evaluation System

By

Jeremy David Gray
B.S. University of Louisville, 2005

A Thesis Approved on

May 2005

By the following Thesis Committee:

Dr. James Graham, Thesis Director

Dr. Ibrahim Imam

Dr. Suraj Alexander

DEDICATION

To Janet and David Gray:
The two people I want to be like when I grow up.

ACKNOWLEDGEMENTS

The author would like to thank all the mentors that have given so much over the past half a decade, the faculty and staff at the Speed Scientific School, as well as the students who have provided much needed support and encouragement. Special thanks must be said to my Thesis Director and entire Thesis Committee for taking the time and patience to help another human being achieve his dreams.

ABSTRACT

Intrusion Detection has emerged as a powerful component of network security systems. A wide range of hardware and software components exist to meet most basic security needs on all platforms. These systems log system usage that could be considered as a breach of security in many networks. However, signature based intrusion detection systems have one catastrophic downfall, in that the number of alerts being logged can quickly outgrow the amount of resources necessary to investigate this anomalous behavior. This thesis explores the use of a fuzzy logic based analysis engine that gives an overall threat level of an intrusion detection sensor, prioritizing alerts that are the most threatening. This application gives security personnel a launching point to determine where security holes exist and a snapshot of the threats that exist in a system.

The fuzzy logic system is based on a set of membership functions that define certain metrics from an alert dataset and a set of rules that determine a threat level based on the defined metrics. This application functions as a proof of concept prototype for an administrative tool that can analyze multiple sensors across multiple networks and give a reasonable output of the threat level across a series of intrusion detection sensors on a network. Initial testing indicates promising performance results for testing the threat level of a remote sensor using this methodology.

TABLE OF CONTENTS

DEDICATION.....	iii
ACKNOWLEDGEMENTS.....	iv
ABSTRACT.....	v
TABLE OF CONTENTS.....	vi
LIST OF TABLES.....	ix
LIST OF FIGURES.....	x
CHAPTER I - INTRODUCTION.....	1
CHAPTER II - LITERATURE SEARCH.....	2
2.1 Intrusion Detection.....	2
2.1.1 Host Based Intrusion Detection Systems.....	3
2.1.2 Network Based Intrusion Detection Systems.....	5
2.1.3 Anomaly and Misuse (Signature) Based Models.....	6
2.1.4 The Snort Intrusion Detection System.....	7
2.2 Threat Evaluation.....	10
2.3 Fuzzy Logic.....	12
2.3.1 Fuzzy Logic Definitions and Operations.....	15
2.3.2 Fuzzy Logic Controllers.....	19
2.4 Other Methodologies for Intrusion Detection.....	20
2.5 Fuzzy Logic Approaches to Intrusion Detection.....	22
CHAPTER III – DESIGN CONSIDERATIONS.....	26
3.1 The Problem.....	26
3.2 Goals.....	26
3.3 Setup Requirements.....	28
3.3.1 Network Setup Requirements.....	28
3.3.2 Database Setup Requirements.....	29
3.3.2.1 Remote/Sensor Databases.....	29
3.3.2.1 Application Database.....	31
3.4 Application.....	31
3.4.1 Introduction.....	31
3.4.2 Application Flow.....	32
3.4.3 Application Objects.....	33
CHAPTER IV – IMPLEMENTATION.....	34
4.1 Implementation Details.....	34
4.1.1 XML Documents.....	34
4.2 System.Windows.Forms Objects.....	35
4.2.1 Purpose.....	35
4.2.2 Add Network Object.....	35
4.2.3 Add Host Object.....	37
4.2.4 ARF_Main Object.....	38
4.2.5 Database Connectivity and Synchronization.....	40
4.3 Fuzzy Logic Modules.....	43
4.3.1 Purpose.....	43
4.3.2 Snort Adapter.....	43
4.3.3 Fuzzification Interface.....	45

4.3.4 Fuzzy Inference Machine.....	50
4.3.5 Defuzzification Interface	52
4.4 Diagrams	53
4.4.1 ARF Class Diagram	53
CHAPTER V – TESTING.....	56
5.1 Introduction.....	56
5.2 Testing Design Parameters	56
5.3 Case Study 1 (All Experimental Data).....	57
5.3.1 Overview of Alerts.....	57
5.3.2 Rule-Sets Defined	58
5.3.3 Membership Functions Defined.....	60
5.3.4 Experimental Results	61
5.3.5 Analysis.....	62
5.4 Case Study 2 (Low Threat Dataset).....	62
5.5 Case Study 3 (Medium Threat Dataset).....	63
5.6 Case Study 4 (High Threat Dataset)	64
5.7 Final Analysis	64
CHAPTER VI – CONCLUSIONS AND FUTURE RESEARCH.....	67
REFERENCES	69
APPENDIX I – TESTING DOCUMENTS	73
AllAlerts.xls	73
Totals.....	73
Experiment 1 – Application State.....	73
Experiment 2 – Application State.....	74
Experiment 3 – Application State.....	76
Experiment 4 – Application State.....	77
HighThreatExperiment.xls.....	78
Totals.....	78
Experiment 1 – Application State.....	79
Experiment 2 – Application State.....	79
Experiment 3 – Application State.....	80
Experiment 4 – Application State.....	81
MedThreatExperiment.xls	81
Totals.....	81
Experiment 1 – Application State.....	81
Experiment 2 – Application State.....	82
Experiment 3 – Application State.....	82
Experiment 4 – Application State.....	83
LowThreatExperiment.xls	83
Totals.....	83
Experiment 1 – Application State.....	84
Experiment 2 – Application State.....	84
Experiment 3 – Application State.....	85
Experiment 4 – Application State.....	85
APPENDIX II – SOURCE CODE	87
ARF_Main.cs	87

AddHost.cs.....	99
AddNetwork.cs	103
RevTreeNode.cs.....	104
Interfaces.....	106
Objects	107
Membership Functions.....	113
Specific Membership Functions	113
General Membership Functions.....	116
Rules	119
Specific Rules	120
General Rules.....	126
FuzzyEngine.cs	128
SnortAdapter.cs.....	129
RulesBasedInference.cs	133
MOMDefuzzify.cs	134
MaxDefuzzify.cs.....	135
Vita.....	136

LIST OF TABLES

Table 5.1: Experimental Parameters	57
Table 5.2: Experimental Results of Case Study 1 (Entire Alert Log)	62
Table 5.3: Experimental Results of Case Study 2 (Low Threat Alerts)	62
Table 5.4: Experimental Results of Case Study 3 (Medium Threat Alerts)	63
Table 5.5: Experimental Results of Case Study 4 (High Threat Alerts).....	64

LIST OF FIGURES

Figure 2.1: Basic Architecture of the Snort Intrusion Detection System [33].....	8
Figure 2.2: Crisp Representation of Temperature.....	14
Figure 2.3: Fuzzy Representation of Temperature	15
Figure 2.4: Fuzzy Union Operation	16
Figure 2.5: Fuzzy Intersection Operation	17
Figure 2.6: Fuzzy Complement Operation.....	18
Figure 2.7: Laplace Transform vs. Fuzzy Logic Methodology [36].....	20
Figure 2.8: RETISS Architecture [34].....	23
Figure 2.9: FIRE Architecture [31].....	24
Figure 3.1: Typical Intrusion Detection Sensor Placement	28
Figure 3.2: Event Table.....	30
Figure 3.3: IPHDR Table.....	30
Figure 3.4: Signature Table.....	31
Figure 4.1: The AddNetwork object	36
Figure 4.2: The AddHost object.....	37
Figure 4.3: The ARF_Main object.....	39
Figure 4.4: Membership Function: Number of Occurrences	47
Figure 4.5: Membership Function: Timespan Between Attacks	47
Figure 4.6: Membership Function: Average Time Between Attacks	48
Figure 4.7: Membership Function: Attacks Per IP	49
Figure 4.8: Fuzzy Logic Class Diagram	54
Figure 4.9: High Level Function of the ARF.....	54
Figure 5.1: Trend for Set of General Membership Functions.....	61
Figure 5.2: Experiment 2 Data (General Membership Functions, Specific Rule-Set)	65
Figure 5.3: Experiment 4 Data (Specific Membership Functions, Specific Rule-Set).....	66

CHAPTER I - INTRODUCTION

In an age where the amount of information being stored and accessed is growing rapidly, the need for secure environments becomes more than an academic exercise. In an attempt to increase the speed of service, more and more businesses are turning toward digital solutions. These solutions can store medical data, control ballast mechanisms on ships, and operate a dizzying array of other objects that people use in their day-to-day lives. In these critical systems, a secure environment is a fundamental need to ensure data integrity and public acceptance of the system.

Intrusion detection systems are a major component of securing any network. Analyzing information on the network layer is imperative for realizing and recovering from security breaches. Most research until now has focused on creating a better intrusion detection system. Reducing the number of false positives and false negatives has been of the utmost concern. The focus of the problem at this point shifts toward finding an intermediate solution until a perfect solution can be found.

Robust, open-source enterprise level applications exist that can give an extremely high number of alerts; these alerts can take a vast amount of time to analyze and do not give an adequate level of threat for a system or network. This document discusses the feasibility of an analysis engine residing on the application layer that uses fuzzy logic to determine a threat level. The application in question is designed to be portable, scalable, and intuitive. The application is described in more detail in chapters 3 and 4, with preliminary research shown in chapter 2. Lastly, the testing, analysis, conclusions, and possibilities for future research are documented in chapters 5 and 6.

CHAPTER II - LITERATURE SEARCH

2.1 Intrusion Detection

Intrusion detection systems monitor electronic traffic and compare that traffic against a pre-existing security policy to confirm that the system is adhering to that policy. In more general terms they are a burglar alarm for computers and networks that look for criminals breaking into a computer system and send notification of the intrusion [1]. Just like a physical burglar alarm that would protect your home, there are many different types of intrusion detection systems. These different types of systems take into account specialized security needs.

Intrusion detection had humble beginnings; the detection engine was comprised of an individual sitting behind a terminal looking for anything out of the ordinary, or in some cases audit logs were printed on fan-fold paper that often reached heights of five feet while waiting for human analysis [2]. It should be noted that these were not even intrusion detection techniques. More realistically, they were system recovery or forensic techniques since it was highly unlikely that any attacker would be caught in the act.

The first work on a formal intrusion detection system was published in 1980 by James P. Anderson. His work set a framework for the general design of a system which provided an initial set of tools to computer system security officers [3]. This paper set the wheels in motion for computing professionals to look toward the future and realize that security practices of the day would not be adequate for the computing activities in the future. This is particularly evident in today's access control mechanisms that are

installed directly into operating systems that help monitor everything from the power grid to medical information.

Later, Dorothy Denning published a paper describing a system whose model is based on the hypothesis that security violations can be detected by monitoring a system's audit records for abnormal patterns of system usage. The model included profiles for representing the behavior of subjects with respect to objects. These objects represented metrics, statistical models, and rules for acquiring knowledge about behavior from audit records and for detecting anomalous behavior [3]. This framework for an intrusion detection system described an implementation-specific solution to a problem that had and will plague computing professionals for years to come, and this framework is the foundation of almost every intrusion detection system.

Intrusion detection has several flavors, many of which will be described throughout this document. Host based and network based are the main types of intrusion detection systems and these systems can be created as anomaly based or misuse (signature) based. However, since all of these systems are designed for the same purpose, they all share the same basic components. These universal mechanisms monitor an information source, detect a threat using an analysis engine, and respond to the perceived threat [4].

2.1.1 Host Based Intrusion Detection Systems

The first formally developed intrusion detection systems were host-based systems. Their defining characteristic is that they collect information about a service running on a host computer. Once the data is collected, it must be processed; this can be done locally

or remotely and still satisfies the requirements for a host-based system because the data being processed is only pertinent for a single host [7].

An undeniable advantage of a host based intrusion detection system is its ability to detect the malicious use of a system by an insider. These systems can reside within the trusted network and can be deployed easily for a variety of services. If a trusted user attempts an unauthorized action, a host based system has a high probability of detecting this action. In addition to detecting unauthorized insider activity, host based systems are also effective at detecting unauthorized file modification [4].

As with any system, there are also downsides to host based systems. First of all, the amount of raw data to be processed could weigh so heavily on the system that the service, as well as host, could be rendered useless. This would be apparent in high traffic services utilized by multiple users. Internal web servers often experience this problem. Another scenario would be if the system was compromised by an unknown attack, the audit/security logging functions could be disabled without the knowledge of security personnel.

The information source for host based intrusion detection systems are the audit/security logs generated by the service being monitored, usually at the operating system level [1]. The analysis engine, located on the physical machine, is a set of pre-programmed rules. The response to an attack is usually to notify security personnel. These components constitute the main elements of a host based intrusion detection system: an information source, an analysis engine, and a method of response for alerting purposes.

2.1.2 Network Based Intrusion Detection Systems

While host based systems monitor traffic for a specific service on a network, network based systems monitor and analyze packets that are sent over a network, usually with network devices operating in promiscuous mode [1]. Since this type of system monitors traffic throughout the network, there are more considerations to take into account. First of all, placement of the system becomes important because of the nature of the network based system. These systems examine every packet, therefore prudent placement could mean the difference between the intrusion detection system delivering malicious packets or successfully analyzing them and reporting a threat. Because of the nature and amount of the information to be processed, network based systems tend to be distributed to handle the analysis of a greater volume of traffic.

Data utilized by network based systems is usually “sniffed” out of network traffic because TCP and UDP traffic is usually not stored for any noticeable duration of time at its endpoint. As a result it is imperative that the intrusion detection system analyze packets en-route.

Unlike host based systems, network based systems are best at detecting unauthorized outsider access. They also address a flaw in the logic of host based systems; in a distributed attack, there could be several target hosts. Each, if compromised, would provide a useful piece of information necessary to compromise the network or a host within the network. A host based system might not see this iterative attack as a threat because of the relatively small steps taken over a large number of hosts, whereas a network-based system looks at the network as a whole.

There are, however, downsides to network-based intrusion detection systems. High speed networks are not conducive to packet sniffing because of the speed of the packet going through the network. Examining every packet can prove time consuming and can undermine the high investment put into high speed data networks due to the latency issues caused by this examination.

2.1.3 Anomaly and Misuse (Signature) Based Models

Anomaly and misuse models function in very different ways. Anomaly based models sense “abnormal” system or user behavior to detect anomalies. Behavior is characterized as “abnormal” if it goes against a pre-existing statistical model that represents normal user and system behavior. This requires system administrators and security officers to build and periodically update a profile that corresponds to normal behavior. This can be advantageous because no prior knowledge of attacks needs to be known; the model is catered to a specific network with specific needs. However, these needs are certain to change over time and the model must be updated. In addition to a need for constant monitoring, imprecise and imperfect models are susceptible to a high rate of false alarms or a high rate of undetected anomalous behavior [7].

Misuse or signature based models monitor network traffic searching for activity patterns that match a known attack or other violation of security policy [1]. For example, an obsolete attack known as the “Land Attack” creates an impossible packet with the same source and destination IP address, which is the IP of a target machine. In some older systems, this illegal packet causes an unhandled error and will cause the target system to crash. Human intervention would be needed to reboot the machine. A single packet arriving at the target is all that is needed to execute this attack [8]. A misuse

based model with an existing rule for all illegally structured packets would catch this attack. Since only known vulnerabilities have known signatures and can be recorded into a rule set, misuse models cannot detect new or unknown attacks [7].

2.1.4 The Snort Intrusion Detection System

Snort is an open source network based intrusion detection system, capable of performing real time traffic analysis and packet logging on IP networks. The varieties of tasks performed by this application are limited only by the imagination. Some of the basic functions include protocol analysis as well as content searching and matching. It should be noted that any necessary analysis could be performed on network traffic because of the architecture, but the main functions to date are designed to detect network intrusions. These functions allow the application to detect numerous kinds of attacks including, but not limited to, buffer overflows, CGI attacks, and OS fingerprinting attacks.

The custom rules language that allows network traffic to be screened with a detection engine allows for a lightweight, flexible product that can be configured to meet the needs of almost any host. The real-time alerting capabilities of Snort can be configured to output through a number of different mechanisms including a UNIX socket, any user specified file, or WinPopup messages.

Snort has three primary uses. It can be used as a straight packet sniffer like tcpdump(1), a packet logger, or as a full blown network intrusion detection system[28]. This self-proclaimed de-facto standard of intrusion detection systems was first envisioned by Martin Roesch in November of 1998. The early version of the system was meant to

be a promiscuous mode packet sniffer, but has grown into the most widely deployed intrusion detection and prevention technology in the world [29].

The basic architecture of Snort is shown below in Figure 2.1

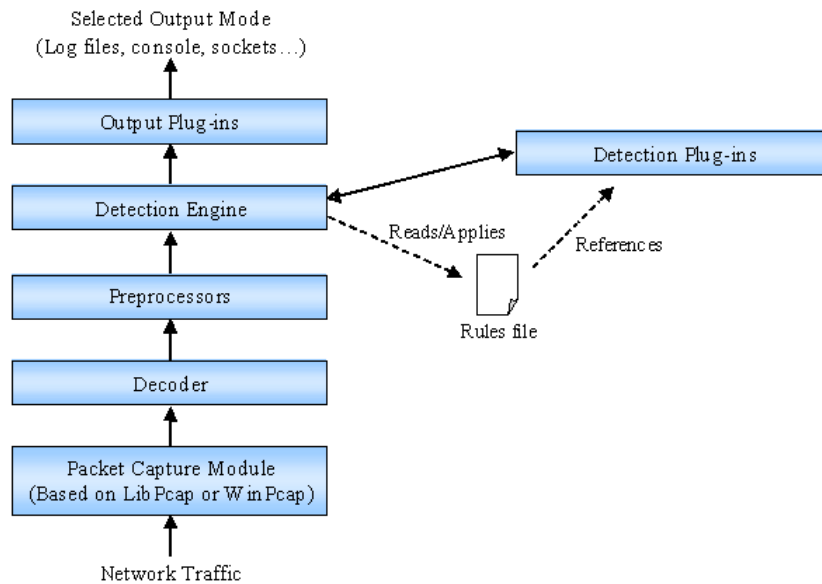


Figure 2.1: Basic Architecture of the Snort Intrusion Detection System [33]

Whenever network traffic comes into contact with a host running Snort, the execution path is as follows. First, the packet capture module pulls unprocessed layer 2 ethernet frames using LibPcap in a LINUX environment or WinPcap on a Windows system. Next, the decoder takes the raw layer 2 data packets and decodes each frame. Each frame is classified by the LAN technology, such as Ethernet, Token Ring, Wi-Fi, PPP, or MPLS. After the LAN technology is defined, Snort decodes the IP protocol and the TCP or UDP packet.

The preprocessing module of this application takes decoded data and performs some basic transformations. Preprocessors can create an alert on, classify, or drop a packet before sending it to the more computationally expensive detection engine. The

“out of the box” preprocessors check each packet to ensure that its format is valid based on the RFC standard. An improperly formatted packet is usually meant to expose network vulnerabilities, such as the obsolete “Land Attack” [8] and will most likely be considered malicious.

Up until this point, the only actions taken on the input data are its capture and basic transformation in order to create a standard form for the information to be processed. The detection engine does the majority of this processing by comparing the captured and transformed data with the rules based detection plug-in. These rules are created by the Snort development team with help from public and private sector security personnel throughout the world and are offered for free download at <http://www.snort.org>. It is also possible to create custom rules based on the predefined format of the application. After the appropriate rules are applied to the data, the output plug-in is responsible for placing the information in a specified location in a specified format. This customizable module allows users to output to any desired datasource.

Based on the architecture and nature of this open source application, a customized intrusion detection system can be deployed in less than an hour as a quick fix or as a long term solution in a network. As stated above, the architecture allows simple customization in three separate nodes: the preprocessors, the detection plug-ins, and the output plug-ins. This powerful customization feature allows security personnel to have complete control over their project from start to finish and provides an ideal environment for research applications.

2.2 Threat Evaluation

On 8 May 1980, the thirty-third World Health Assembly declared that smallpox had been eradicated globally [9]. The World Health Organization realized that, across the world, smallpox was an epidemic that could be handled through an aggressive monitoring and vaccination program. For the first time in history, mankind had vanquished a disease. It must be borne in mind, however, that this was not the first attempt at global disease eradication, but the fifth [10]. The idea of evaluating threats to computer systems holds as many complications as dealing with a world-wide spread of an infectious disease. There are several important lessons to be learned from the program to eradicate smallpox; first of all, the threat must be identified. The news of this identification must propagate through all affected areas and preventative measures should be in place to deal with the spreading of this threat. The major difference between a human disease and a computer vulnerability or threat is the timeline; these activities do not take place over the course of years, they take place over the course of minutes. A threat, in general, is regarded as a possible danger [11]. Therefore, all traffic moving in and out of a host within a network is a threat to the network itself. Disgruntled employees, uneducated users, or compromised machines all pose potential hazards to the functional status of a network.

Servers and networks receive an almost constant bombardment from outsiders; automated scripts ranging in complexity from port scanning to distributed denial of service attacks. Information about these automated tasks changes from day to day and is generally over-simplified or disregarded. Within the field of network security, there is often a great amount of debate over acts that seem as simple as an automated port scan.

Numbers and figures are hard to come by and are convoluted at best, but what is known is that computer systems are compromised by outside attackers.

Another type of threat to computer systems is insider abuse. According to WarRoom Research's 1996 Information Systems Security Survey, 62.9% of the companies surveyed reported insider misuse of their organization's computer systems [12]. These attacks happen within the network, and in most cases the data does not pass through any security mechanism such as a firewall. They are carried out by motivated, educated employees who know the systems they are attacking and the security measures in place.

The Computer Security Institute's 1998 Computer Crime Survey, conducted jointly with the FBI, reported the average cost of an outsider penetration was \$56,000, while the average insider attack cost a company \$2.7 million [13]. These types of threats, along with unintentional damage, pose a giant problem to the safety and scalability of a modern computer network, and many approaches have been designed and implemented to mitigate these threats. The most interesting of these methodologies is called the human immunology method, which is based on the human immune system.

By far, the most complex physical system that exists to date is the human body; thus the ideas encapsulated during human evolution can be used as a basis for research into many areas. The most intriguing idea, with respect to Network Security, is the multitiered approach the immune system takes for protecting the body. The skin provides a defense against external pathogens and is relatively successful in restricting pathogens from entering the body. But people still get sick.

Because the skin is not 100% successful in restricting access to the body, there are internal systems in place. These internal systems must decide whether or not what they are analyzing belongs in the body. In other words, the system must classify organisms as “self” or “non-self” in order to be effective. If the organism is classified as “self” it is left alone, because it belongs in the body. However, if the entity is classified as “non-self”, the immune system attempts to purge the body of the unauthorized presence.

Some notable features of the human immune system relevant in the study of computer threat evaluation are diversity across different systems and inexact detection [14]. Each copy of the immune system is unique. Each individual in a population has a different set of protective cells and molecules. Computer security often involves protecting multiple sites or networks. In these environments, once a way is found to avoid detection in one network, all networks become vulnerable. A better approach would be to provide each protected location with a unique set of detectors. This concept makes sense with respect to intrusion detection systems because of the vast differences in user behavior from one network to another. Thus if one were compromised, others would likely remain secure [15].

This approach to mitigating threats through a network will be discussed in more detail in later phases of this document. However, the basic goal is clear: creating a stable, adaptable definition for “self” and using that definition to categorize traffic through a network as a threat.

2.3 Fuzzy Logic

One of the principal difficulties encountered by people in technical sciences is the need to bridge the gap between human thought and the way machines are told to think.

For instance, it is very easy for a human to look at a group of people and consciously say, “Most people here are young,” but this same simple observation would be relatively hard for a machine to deduce. This difficulty lies in a traditional machine’s inability to assign a grade of membership to an entity without rigidly defined boundaries. If the above statement was given the form “Fifty percent, or more, of the people are over the age of 45,” a computer could very easily give a true or false assertion to the statement.

Zadeh says that humans have a remarkable ability to assign a grade of membership to a given object without a conscious understanding of how the grade is calculated [17]. This holds true in cases such as “Is it warm outside?” or with human constructions such as *heavy*, *light*, *good*, *cold*, or *somewhat*. In other areas of computing, rigid boundaries are defined for imprecise statements like cold. The definition of cold would vary from person to person, and no precise boundaries could be established for exactly what is cold and what is not.

A value within a fuzzy set can include all real numbers in the interval $[0, 1]$, giving a more exact measurement than typical boolean models. The measurements given are more exact because they are not restricted to the values within boolean logic of true or false. This concept, at first, resembles ordinary probability theory. An ordinary probabilistic methodology defines the probability of a true assertion within a set over the course of time. Membership values within a probability based model are restricted to either true or false, showing the value is within the set or not. This methodology requires a crisp definition of entities, usually a square wave or piecewise square function of some type. On the other hand, fuzzy logic assigns a grade of membership for each specific

item within the set or the set itself to determine membership. Fuzzy logic membership functions can use any mathematical model whose values lie in the interval $[0, 1]$.

A typical, crisp machine representation of temperature would look similar to figure 2.2, shown below.

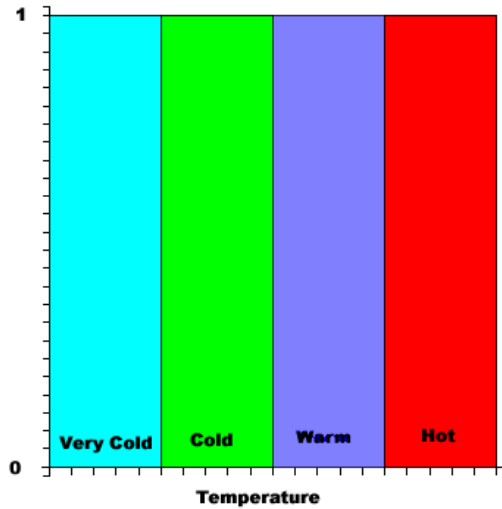


Figure 2.2: Crisp Representation of Temperature

However, a representation for temperature using a fuzzy approach would be very different due to the gradual changes between membership areas. This is illustrated in figure 2.3, shown below.

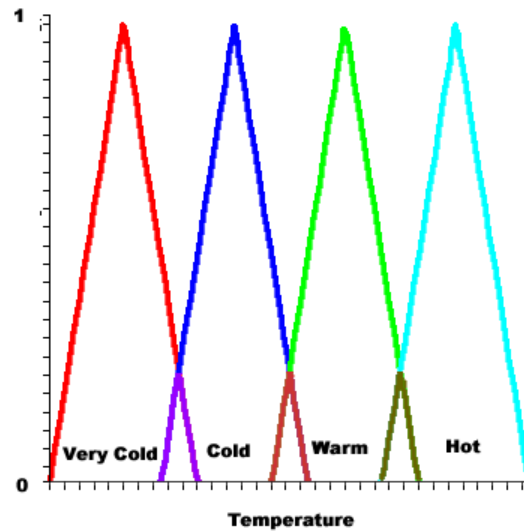


Figure 2.3: Fuzzy Representation of Temperature

The goal of fuzzy logic is to bridge this gap between human thought and machine function, and it can be defined as a kind of logic using graded or qualified statements rather than ones that are strictly true or false. The results of fuzzy reasoning are not as definite as those derived by strict logic, but they cover a larger field of discourse [17].

In order for an effective fuzzy approach to be utilized, a membership function must be defined. A mathematical function is made to decide on the representation of the grade of membership of an item within a fuzzy set. This function can be defined in many ways, such as an expert system or a panel decision [19].

2.3.1 Fuzzy Logic Definitions and Operations

This section presents a set of definitions for fuzzy logic concepts and operation and the contents are integral in the development of fuzzy logic systems.

Universe of Discourse: The collection of elements under consideration.

Membership Function: A function that defines the degree of membership of the elements in a particular set.

Fuzzy Set: A set that can contain elements with only a partial degree of membership, this set is derived from a particular membership function and each element maps to a real number on the interval $[0,1]$ representing the elements grade of membership.

Union Operation: Represented as $A \cup B$, the result of this operation on two fuzzy sets A and B is a fuzzy set denoted by $A \cup B$ whose membership function is given by $\mu_{A \cup B} = \max(\mu_A(x), \mu_B(x))$. This operation is equivalent to the OR operation in Boolean algebra and can be visually interpreted as shown in figure 2.4.

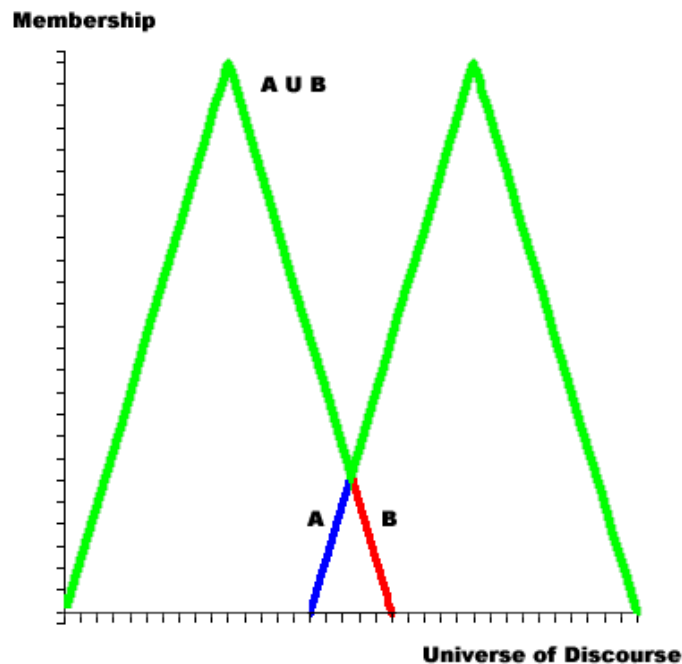


Figure 2.4: Fuzzy Union Operation

Intersection Operation: Represented as $A \cap B$, the result of this operation on two fuzzy sets A and B is a fuzzy set denoted by $A \cap B$ whose membership function is given by $\mu_{A \cap B} = \min(\mu_A(x), \mu_B(x))$. This operation is equivalent to the AND operation in Boolean algebra and can be visually interpreted as shown in figure 2.5

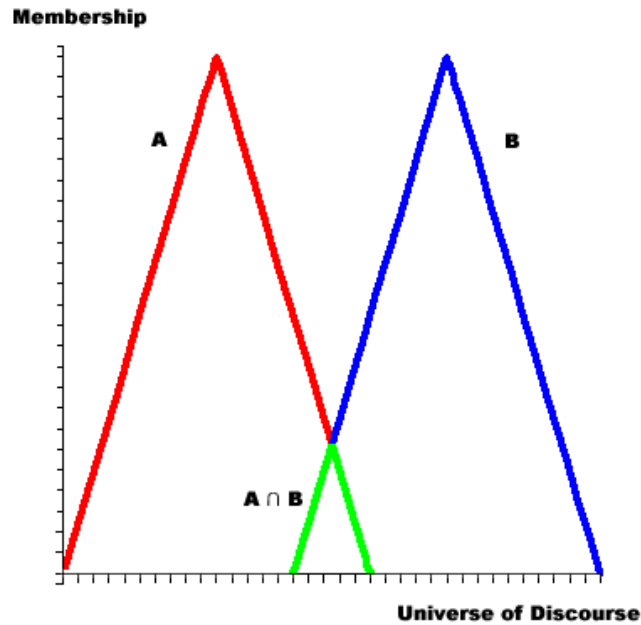


Figure 2.5: Fuzzy Intersection Operation

Complement Operation: Represented as \bar{A} where A is the fuzzy subset in which the complement operation is applied. The result of the operation is a fuzzy set whose membership function is given by $\mu_{\bar{A}} = 1 - \mu_A$. This operation is equivalent to the NOT operation in Boolean algebra and can be visually interpreted as shown in figure 2.6.

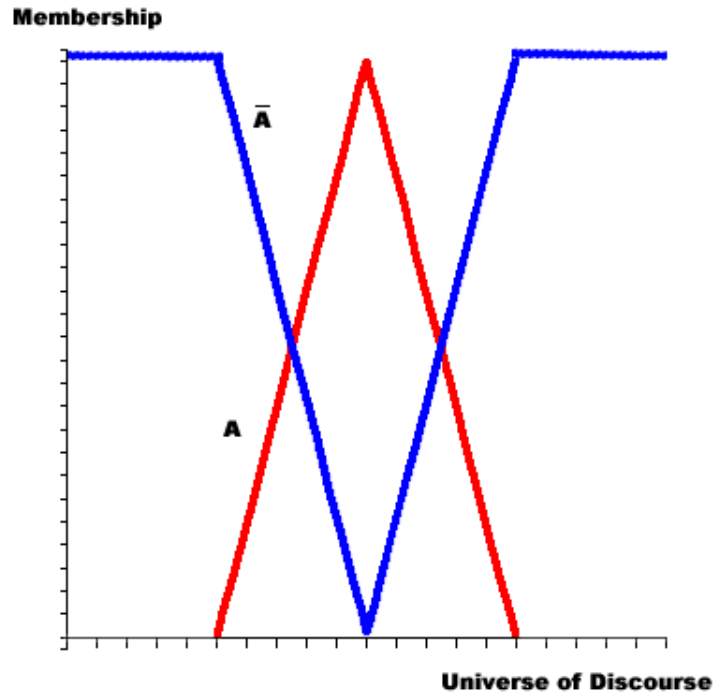


Figure 2.6: Fuzzy Complement Operation

Cartesian Product: Represented as $A \times B$ where A and B are fuzzy subsets. The Cartesian product is the binary variable represented by $\mu_{A \times B}(u,v) = \mu_A(u) * \mu_B(v)$. For example, $A = \{a,b\}$ and $B = \{c,d,e,f\}$ then the Cartesian product of A and B would be equivalent to $\{ (a,c),(a,d),(a,e),(a,f), (b,c),(b,d),(b,e),(b,f) \}$

Fuzzy Singleton: A fuzzy set, A , whose support is a single point x in the universe of discourse X .

Height of a Fuzzy Set: The height of a fuzzy set, A , is the set of elements defined in the universe of discourse X at which the membership function $\mu_A(x)$ equals one.

Normal Fuzzy Set: A fuzzy set is called normal if and only if maximal values of its membership function are equal or larger than α .

Crossover Point: The Crossover point of a fuzzy set is the element in U at which its membership function is 0.5.

It is noted that DeMorgan's Law, Associativity, Commutivity, and Distributivity can also be applied in the realm of fuzzy logic.

2.3.2 Fuzzy Logic Controllers

The basic ideas of what fuzzy logic is, along with the operations listed in the previous section are used to create fuzzy logic controllers that provide solutions to a vast number of problems. The applications for fuzzy logic based systems grows by the day but specifically are used in water quality control [20], automatic train operations [21] and automotive transmission controls [22].

A fuzzy logic control system is composed of four principal elements: fuzzy rule base, fuzzification interface, fuzzy inference machine, and defuzzification interface [23].

These components work together to create digital control devices that allow a human description of a physical system and of the required control strategy to be simulated in a reasonably natural way [19]. A brief description of these components is listed below:

1. The fuzzy rule base is a collection of rules that define specific actions of the expert-system. These rules follow a basic IF THEN format.
2. The fuzzification interface maps crisp inputs into their corresponding fuzzy values based on the appropriate membership function.
3. The fuzzy inference machine performs any number of fuzzy logic operations to ascertain control action for fuzzy inputs.
4. The defuzzification interface converts manipulated fuzzy values and creates a crisp output for a control device.

The idea of fuzzifying crisp inputs, doing calculations to create an output, and defuzzifying those outputs into crisp values is similar to the methodology of solving a linear time-invariant system with a Laplace transform as shown in Figure 2.7

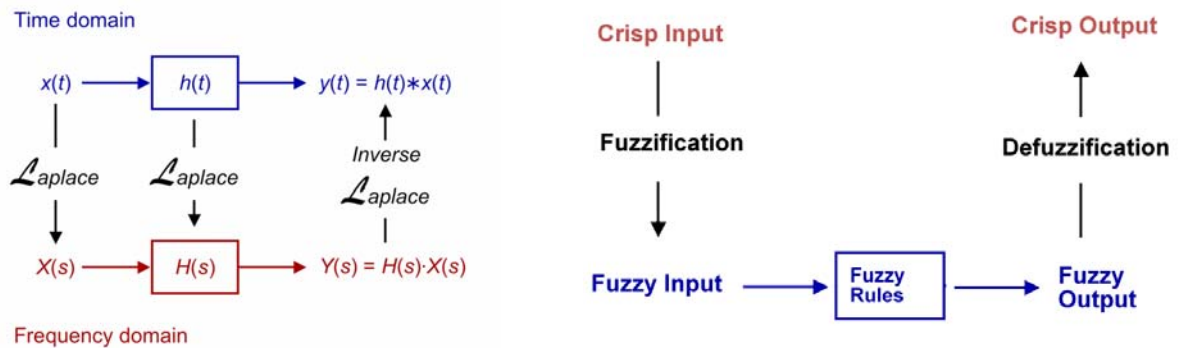


Figure 2.7: Laplace Transform vs. Fuzzy Logic Methodology [36]

2.4 Other Methodologies for Intrusion Detection

There have been many attempts to reduce the number of false alarms within intrusion detection systems. Outside of using fuzzy logic, there are two notable approaches: data mining and the use of neural networks.

Data mining is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data [24]. The key ideas are to use data mining techniques to discover consistent and useful patterns of system features that describe program and user behavior, and use the set of relevant system features to compute inductively learned classifiers that can recognize anomalies and known intrusions [25]. This approach to intrusion detection sifts through large amounts of data collected using the audit capabilities of the host machine and uses this information to create detection models for the intrusion detection system. The creation of detection models is based on

the fundamental principals of data mining that have been inspired from a wide variety of statistically related fields.

Since this approach to intrusion detection is based on a statistical model, it can be approached in an implementation free environment. A widely used, hypothetical, example of data mining would use the transaction history of any large supermarket chain. Analysis of transactions of goods bought over a defined period of time could find that two seemingly unrelated products such as beer and diapers are often bought together. Drawing this conclusion is not intuitive and would be very difficult without data mining in a large set of data; however, taking advantage of this relationship would be very easy. Herein lays the strength and weakness of an approach to intrusion detection using data mining. It requires a large amount of data to be processed for correlations to be drawn. The necessity for this large amount of data could pose a vulnerability to systems that have not collected a sufficient audit trail, and the system could be compromised before the audit trail is of sufficient size. This would make the audit trail itself vulnerable and taint the results of any analysis.

Another weakness of a data mining approach to intrusion detection is the foundation of data mining itself. Data mining finds correlation between two seemingly unrelated entities. Correlation between entities does not imply causation. It is not uncommon for a data mining set to find over five hundred correlations between data, while less than ten of them are worthwhile [26]. A typical correlation that could be found is as follows:

*Ice-cream sales are strongly correlated with crime rates.
Therefore, ice-cream causes crime.*

While ice cream sales and crime might be strongly correlated, this type of thinking does not take into account the element of coincidence or other confounding factors. While the fundamentals of data mining are important in many aspects of computer science, there are serious flaws for applications involving security.

Neural networks are also an active area of research in the field of computer science and intrusion detection. The basic idea of a neural network based intrusion detection system is that a user leaves a specific print when using a system. The neural network is utilized to identify this print and identify a user based on their specific behavior [27]. Most neural network based systems are computationally expensive and run in offline mode. A single job run at the end of day processes audit logs left on host computers, finding suspicious behavior and alerting personnel to launch an investigation.

It is also widely speculated that the scalability of neural network based systems would present a problem when the amount of users exceeds a typical small business. However, the method of neural networks does an excellent job in building a real time profile of user behavior that can adapt over time [27].

2.5 Fuzzy Logic Approaches to Intrusion Detection

The first notable intrusion detection system to utilize fuzzy logic is the real time security system, RETISS [30]. The system is based on the idea that a correlation between anomalous user behavior and threats. Each rule corresponds to a weight table that expresses a level of danger for a specific anomaly. These weights, based on level of danger, are piped through a fuzzy inference module and combined using fuzzy logic to connote the probability of threat. RETISS uses the following architecture:

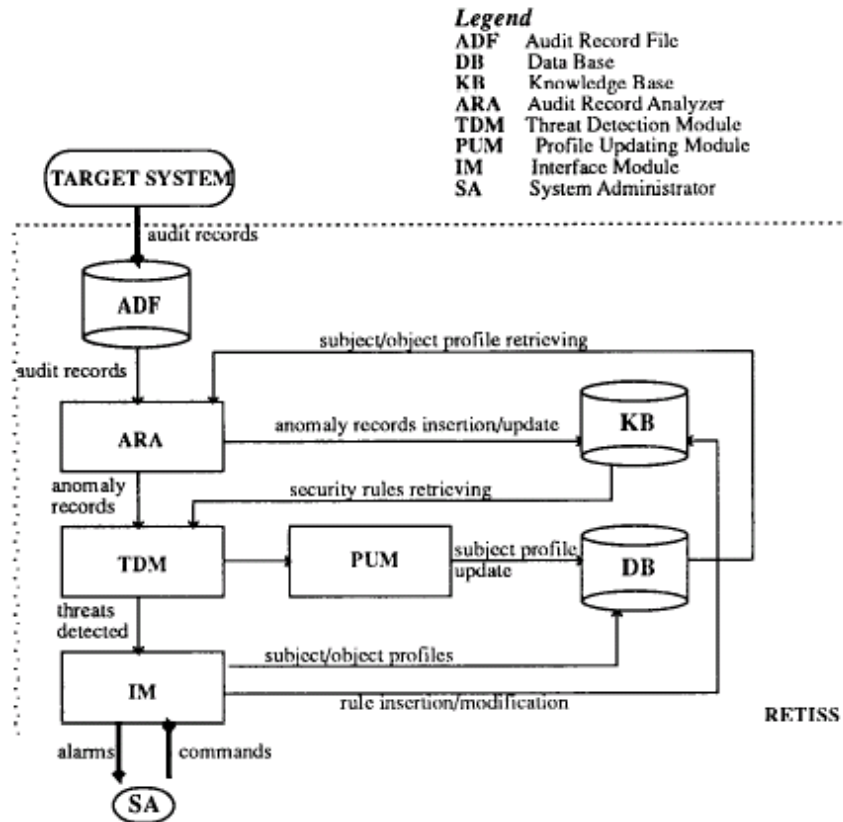


Figure 2.8: RETISS Architecture [34]

This system was implemented on a Sun 3/50 workstation with the Common List programming language and runs on a machine separated from the target system. This separation avoids overloading the activity of the target system and protects RETISS from target system users [30]. Even though this platform was chosen by its designers, this implementation can be adapted for any platform.

Another notable intrusion detection system is the fuzzy Intrusion Recognition Engine, FIRE, which is a network based intrusion detection system that uses an agent based methodology [31]. This agent based methodology, based on AAFID developed at Purdue by Zamboni, et al., allows each individual agent to perform as a separate fuzzification interface while communicating with a fuzzy inference engine.

As shown below, this agent based system uses three main modules: agents (A), transceivers (T), and monitors (M). Agents monitor processes of the host, while transceivers control local agents and act as communication tools between agents and monitors, while monitors control local and remote entities. Monitors handle correlation of data and send alarms to the user interface (UI). Agents a_1 - a_5 , shown below, would be assigned to different types of traffic such as TCP, UDP, or specific categories of threat. These agents would pass information, through a transceiver to a monitor. This diagram dictates that M_1 is the monitor for FIRE and M_2 is the monitor for AAFID [31].

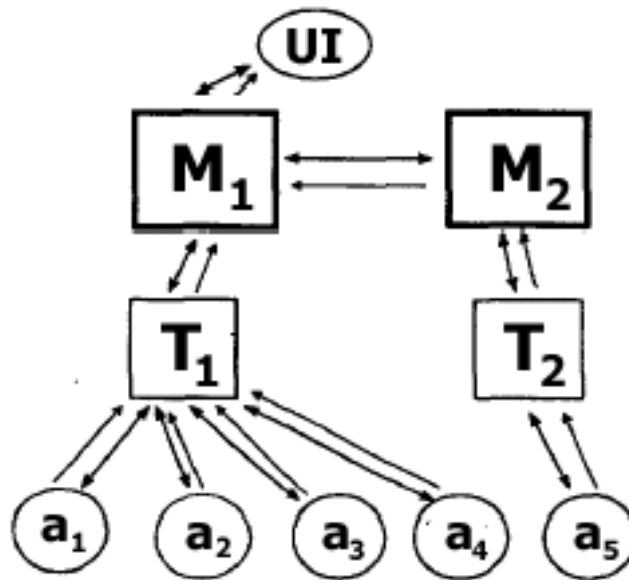


Figure 2.9: FIRE Architecture [31]

This process, while complex, is an active area of research because of system expandability. Expanding an agent based system to encompass new threats could be done by deploying a new agent based on expert knowledge of the threat.

Another approach to the use of fuzzy logic in intrusion detection was introduced by Yingbing Yu in a dissertation titled, “Anomaly Intrusion Detection and Threat Evaluation using Artificial Immunity Model and Fuzzy Logic” [37]. This work proposes

the detection of threats using a hierarchical fuzzy threat evaluation mechanism. First, a finite automata is created to model the behavior of users and the system, based on a behavior profile. These behaviors are applied to measure similarity as well as deviation for behavior. After behavior is categorized as self or non-self, a hierarchical fuzzy reasoning system compares calculates generalized fuzzy number with a weighted value. This model can detect masqueraders and intrusion scenarios in a relatively short timespan.

There have been other approaches using fuzzy logic, such as the approach proposed by Jianxiong Luo and Susan Bridges. Their approach integrates the mining of fuzzy association rules and frequency episodes with fuzzy logic to produce more abstract and flexible patterns for intrusion detection [32]. These approaches to fuzzy logic all introduce elements of architecture and design that allow the achievement of specific goals. All of the goals, however, are aimed at reducing the number of false alarms.

CHAPTER III – DESIGN CONSIDERATIONS

3.1 The Problem

Intrusion detection systems, especially those that are signature based, present security personnel with vast amounts of alert data. In the case of the open source intrusion detection tool, Snort, alerts are sent to an output plug-in which presents information to an authorized person in any manner they wish to implement. It will be assumed that in most systems, alerts are logged in a MySQL database. Even though a relational database is an efficient storage mechanism with production grade read and write times, the number of logged alerts can quickly grow beyond the analytical skills of any human. The questions at this point become, “how secure is this network?” and “what alerts are the most critical, so they can be prioritized for patching and upgrades?”

How can someone easily analyze this collected data and give a meaningful output with a high probability that the alerts in question do not represent a false positive? Approaching this problem using fuzzy logic can significantly reduce the workload of security personnel by evaluating alerts in parallel and giving a reasonable snapshot of a threat level. An automated real-time fuzzy logic threat evaluation system, the ARF threat evaluation system, will be implemented to address this problem. This software package, will from here on be referred to as, ARF. This chapter addresses design considerations for ARF. Implementation details will be presented in chapter four.

3.2 Goals

ARF is to be written using an interface for every step of the fuzzy process as well as an interface for the fuzzy rules used in the inference machine and membership functions used in the fuzzification process. This will allow for a scalable, tunable application that can be easily adapted to allow the addition of specific rules and more complex mathematical calculation.

The GUI should provide an avenue for experienced security personnel to get a snapshot of the level of threat in a single sensor as well as a comparative view of all sensors in a network. While little or no knowledge of fuzzy logic is needed to make inferences about the level of threat in a network, knowledge of pertinent metrics and the ability to create and understand fuzzy rules based on the metrics is required to tune the application to a certain environment. This application, for instance, would employ a separate rule base if it were inside the protected network, than if it were outside the network.

Due to the proliferation of technology within our society and the variety of applications and platforms available, the application uses the Extensible Markup Language (XML) for all non-database persistent storage. This feature will undoubtedly increase the availability of the analyzed output data for future applications and reporting mechanisms and provide a simple format for the application to parse the information and display it through the GUI.

3.3 Setup Requirements

3.3.1 Network Setup Requirements

Intelligent sensor placement should be the first step in deploying any intrusion detection system. A typical intrusion detection sensor placement, according to Earl Carter of the Cisco Press, is shown below in Figure 3.1.

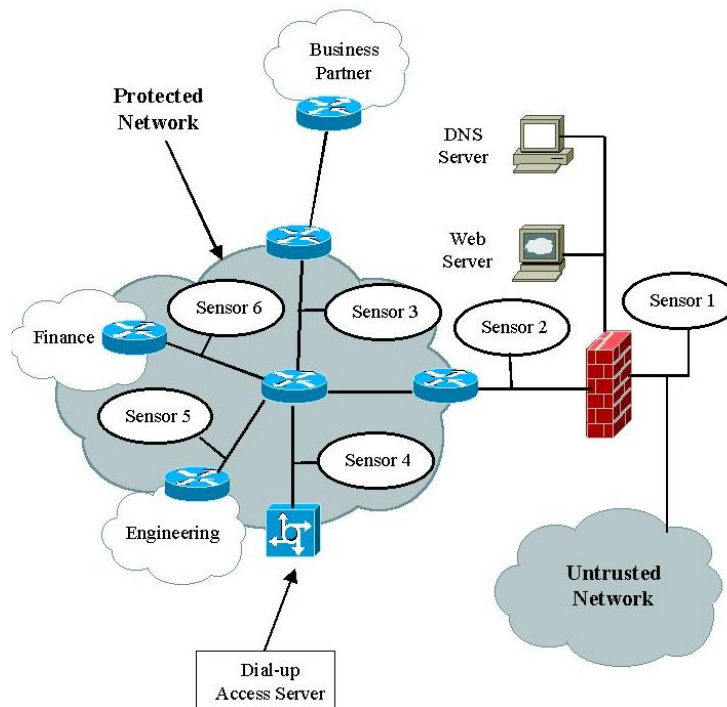


Figure 3.1: Typical Intrusion Detection Sensor Placement [35]

Notice that even within a trusted network there are multiple sensors. In a commercial setting, it is prudent to use a separate machine on the network acting as a pass-through device for a sensor. In reality, a sensor is software and could be placed on any machine, including desktops and servers. This is not common practice in

commercial servers because the fear of placing extra computational load on a production machine directly affects the performance of a server as well as the expense of upgrading hard drives in a redundant array. It is also not considered a best practice to use a sensor on a desktop because the data collected would be specific to the traffic on a single computer. The optimal situation for a sensor would be a dedicated machine with two network interface cards acting as a gateway on a branch of a network.

For academic and experimental purposes, a standalone sensor using Snort, Apache, SSL, PHP, MySQL, and BASE on a computer running an updated version of Fedora Core 4 is utilized as a data collection agent. The only requirements for this sensor to operate properly with the proposed application are:

1. Snort must use MySQL to log alerts.
2. MySQL must accept TCP/IP connections on port 3306.
3. For security purposes, MySQL should include a user account with SELECT privileges for the application to view the alert logs.

3.3.2 Database Setup Requirements

3.3.2.1 Remote/Sensor Databases

Applicable tables in a sensor database require no modification. However, the tables that will be used for analysis by the application are shown below. Only fields applicable to the function of the application are shown.

The EVENT table, shown in Figure 3.2, is the most commonly referenced table in the MySQL database created by Snort. The primary key two-tuple is made up of a sensor

ID, allowing multiple sensors on a network to report to the same database as well as a command ID that identifies the alert with an auto-incrementing integer. The MySQL data types for these fields are, respectively, integer(10), integer(10), integer(10), and datetime, where all integers are unsigned and all fields are required.

event	
PK	<u>sid</u>
PK	<u>cid</u>
	signature
	timestamp

Figure 3.2: Event Table

The IPHDR table, shown in Figure 3.3, has a one to one row relationship with the event table and holds all IP header information as specified in RFC 791. IP addresses are stored as integers to reduce space required and can be transformed using the MySQL methods IP address = inet_ntoa(integer) and Integer = inet_aton(IP Address).

iphdr	
PK	<u>sid</u>
PK	<u>cid</u>
	ip_src
	ip_dst

Figure 3.3: IPHDR Table

As alerts are inserted into the MySQL database, Snort checks to see if the alert signature already exists within the SIGNATURE table, shown in Figure 3.4. If the signature does not exist, it inserts a row to identify the integer signature recorded in the EVENT table. In other words, this table shows the textual name of numerical signatures referenced from other tables.

signature	
PK	<u>sig_id</u>
	sig_name

Figure 3.4: Signature Table

*Note: non-applicable fields have been omitted from Figures 3.2 – 3.4

3.3.2.1 Application Database

A computer running the application to be described requires the presence of a MySQL server instance. The tables to be created are exact replicas of the tables mentioned in the previous section with the exception of the primary keys in the event and IPHDR table. The primary key for the application, or local database is a three-tuple consisting of (sid,cid,nid) where nid is an alias or other identifier to identify the sensor with a network. This is necessary because all data from all sensors will be copied to the application database. Creation of these tables is not a function of the application.

3.4 Application

3.4.1 Introduction

ARF is implemented using the .NET framework 1.1 and the C# (C Sharp) programming language. Using this framework, it is possible to make use of not only the object oriented nature of the language and the powerful debugging tools, but also a vast array of inheritable built-in objects. The application includes one external reference to the MySQL Connector/.NET which implements ADO.NET interfaces allowing for simple, secure, and high-performance data connectivity to a MySQL database from any

.NET application. MySQL Connector/.NET is well documented and available for free download at <http://www.mysql.org>.

A fuzzy logic approach to analyzing data collected from Snort is used to work towards the goal of reducing the number of false positives in an intrusion detection sensor running Snort. This approach needs to display pertinent and detailed information about the data being analyzed and output a level of threat based on alerts of the host in question. These outputs should be organized on a graphical user interface (GUI) and create an intuitive “snapshot” of many sensors over many networks.

3.4.2 Application Flow

The application exists in two logically separated modules. The GUI with its associated properties and methods need to take care of creating, connecting to, displaying, and synchronizing databases between the application and a selected sensor. Another main module should be the fuzzy logic module that accepts a set of data from MySQL and returns a floating point number between 0 and 1 that indicates the level of threat for the sensor.

The fuzzy logic module is comprised of four separate classes, dictated by the logical steps in the fuzzy logic process.

1. A class to adapt a DataTable, representing alerts in a remote database, sent by the GUI into predefined metrics.
2. A class to fuzzify the metrics calculated from the previous class based on predefined membership functions.

3. A class to make inferences about the fuzzy values from a base of rules and create an XML file to log specifics of the process for display in the GUI.
4. A class to defuzzify the output of the previous class, which gives a threat value on the interval [0,1]. This outputted value will be displayed on the GUI.

Each of these classes needs to implement an interface designated for the particular step in the fuzzy process in which they reside. This process will create an environment in which the effort to add new functionality and perform basic maintenance will be greatly reduced.

3.4.3 Application Objects

Every step in the fuzzy logic module of the application should output a typed collection of custom objects for experimental and debugging purposes. The objects hold all applicable output calculations from the previous class and serve as an input to the next class. A typed collection of custom objects is implemented in order to reduce the possibility of running the fuzzy logic classes in any order except the order in which they are designed to be run. The collections should implement the methods:

```
public virtual void Add(Object_Name c)
public virtual Object_Name this[int Index]
```

These methods allow adding of objects to a collection and referencing the objects within the collection with an integer index. Any other functionality will not be permitted on custom collections, as they are meant to be containers for data.

CHAPTER IV – IMPLEMENTATION

4.1 Implementation Details

The implementation of the application, ARF, uses two main categories of software components. The first component handles all administrative functionality of the object and is encapsulated into several instances of `System.Windows.Forms` objects. The second component handles all fuzzy logic operations and is logically placed inside its own namespace `arf.fuzzy`. These components, to be described fully in later sections of this chapter, compose the framework for this application.

4.1.1 XML Documents

The `hosts.xml` file was created as a logical container to hold applicable information about sensors. Using this format, it is possible to hold information about many different networks containing many different sensors. This would be achieved by adding a new XML element named `network` with a single attribute, `name`, that denotes an alias for that network. Within each network it is possible to have zero or many hosts with all applicable information. The format of this document is shown in the code segment below.

```
<network name="419winkler.com">
  <host>
    <alias>Thesis Box</alias>
    <ip>localhost</ip>
    <dbname>home_alerts</dbname>
    <dbuser>root</dbuser>
    <dbpass>password</dbpass>
  </host>
  <host>
    <alias>Test 2</alias>
    <ip>localhost</ip>
    <dbname>poopie</dbname>
    <dbuser>rewt</dbuser>
    <dbpass>password</dbpass>
  </host>
</network>
```

4.2 System.Windows.Forms Objects

4.2.1 Purpose

The System.Windows.Forms objects are in charge of collecting input and sending that input to the fuzzy logic engine for analysis. After the analysis is complete, the System.Windows.Forms objects are responsible for displaying any necessary analysis. These tasks are completed using native .NET objects with the exception of a specialized tree node class and the MySQL Connector/NET reference. These System.Windows.Forms objects also contain powerful database connectivity and transaction management objects.

4.2.2 Add Network Object

This System.Windows.Forms object, AddNetwork, inherits from the .NET object System.Windows.Forms.Form and has a single purpose. This purpose is to create a new network alias in the hosts.xml configuration file. The desired output of this function will insert the following description into the hosts.xml file:

```
<network name="AnyNetwork.com">
</network>
```

AnyNetwork.com denotes an alias which refers to a network containing sensors with MySQL databases holding alerts generated by Snort.

The output from the System.Windows.Forms object, AddNetwork is shown below in figure 4.1



Figure 4.1: The AddNetwork object

The functionality necessary to append this XML element at the end of a list of networks within the hosts.xml file is listed below. This method is triggered by a button click event on the form object, and the name of the network is stored in a text box named txtNetworkName.

```
if(this.txtNetworkName.Text.Trim() != "" && this.txtNetworkName.Text != null)
{
    // Load Existing hosts.xml file
    System.Xml.XmlDocument networks = new XmlDocument();
    networks.Load(xmlfilename);

    // Create a list of xml nodes with the tag name "network"
    System.Xml.XmlNodeList nl = networks.GetElementsByTagName("network");

    // Create New XML Element
    XmlElement newcatalogentry = networks.CreateElement("network");
    XmlAttribute newcatalogattr = networks.CreateAttribute("name");
    newcatalogattr.Value = this.txtNetworkName.Text;
    newcatalogentry.SetAttributeNode(newcatalogattr);

    // Add XML element at end of node list
    networks.DocumentElement.InsertAfter(newcatalogentry,
    networks.DocumentElement.LastChild);

    // Save the new XML file to disk
    FileStream fsxml = new FileStream(this.xmlfilename, FileMode.Truncate,
    FileAccess.Write, FileShare.ReadWrite);

    networks.Save(fsxml);
    fsxml.Close();
}
// Close the form
```



```
this.Close();
```

The code segment shown above loads the hosts.xml document and adds a new network element to the document. The element added will have the attribute that was typed into the form shown in figure 4.1.

4.2.3 Add Host Object

This System.Windows.Forms object, AddHost, inherits the .NET object System.Windows.Forms.Form and has a single purpose. This purpose is to create a new host within a network in the hosts.xml configuration file. The desired output of this function will insert the following into the specified file:

```
<host>  
  <alias>Sensor_Alias</alias>  
  <ip>Address of MySQL Database Holding Sensor Data</ip>  
  <dbname>Name of Database</dbname>  
  <dbuser>Restricted Privilege User</dbuser>  
  <dbpass>Restricted User Password</dbpass>  
</host>
```

This information represents all necessary connection information to access the MySQL database that is being used by snort on a specific sensor. The initial state of the System.Windows.Forms object, AddHost is shown below in figure 4.2.

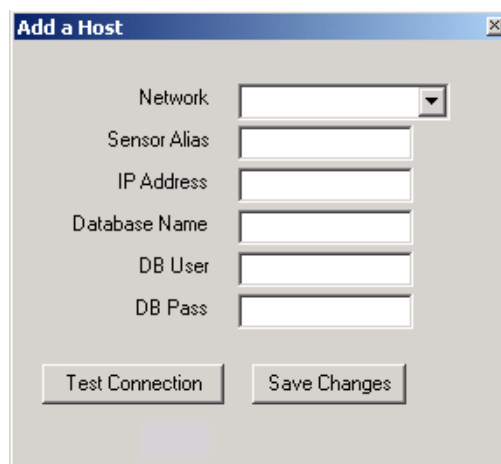


Figure 4.2: The AddHost object

The functionality of this object is slightly more complex, but nonetheless a variation on a theme. The list of networks in this form is populated with a list of all the networks listed in the hosts.xml file, while all other information is provided by the user. After all information is entered, there are two separate functions necessary to complete the AddHost operation. The first is a method to test the connection to the sensor whose information was entered. This function creates a connection to the server and instantly closes it. If these operations are successful, a message is shown to the user indicating success. If the operations fail a message is shown to indicate failure.

The second function of this object is to save the information typed in each field into an XML document. It should be noted that a valid connection is not required for insertion into the XML file and by default, the “Save Changes” functionality prompts the user if the connection is not valid before inserting it into the XML file. Since the functionality is basically identical to the AddNetwork object, shown in the previous section, code has been omitted in this section and is provided in its entirety in the appendix.

4.2.4 ARF_Main Object

This System.Windows.Forms object, ARF_Main, inherits the .NET object System.Windows.Forms.Form and contains the main functionality of the application and is shown below in Figure 4.3.

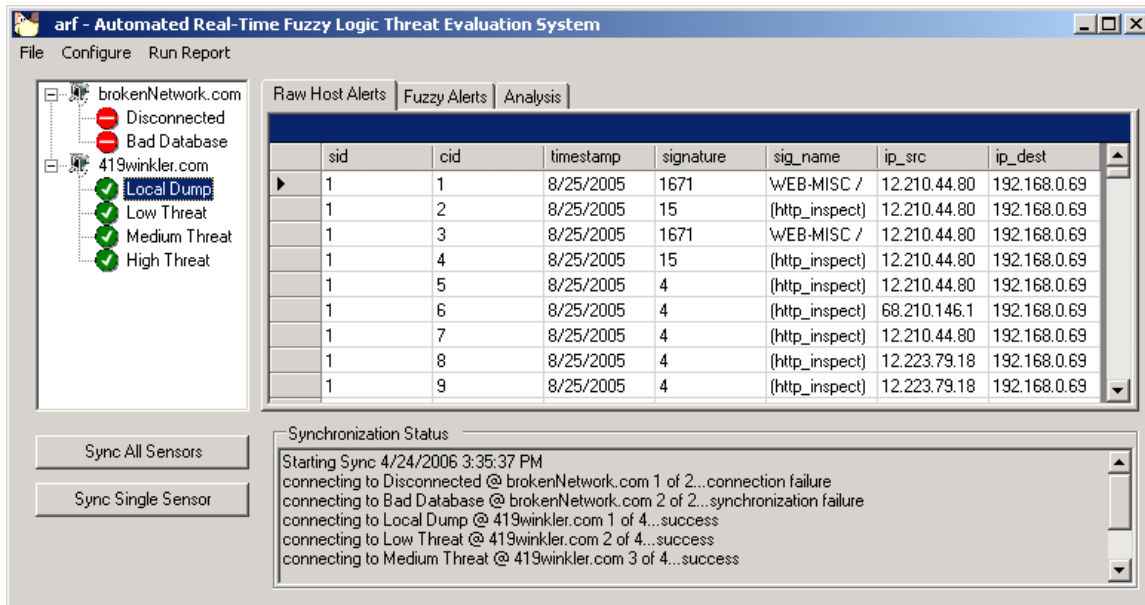


Figure 4.3: The ARF_Main object

The basic components of this object are a System.Windows.Forms.TreeView object that shows a collapsible list of networks and sensors as well as a System.Windows.Forms.TabControl which organizes data representing a sensor in a logical format. Data shown in the TabControl is changed when a sensor within the TreeView object is clicked with a mouse.

The TreeView object displays a hierarchical collection of labeled items, each represented by a System.Windows.Forms.TreeNode or a custom object arf.RevTreeNode. A network is represented as a TreeNode that has children. The network can expand as well as collapse, it can indicate itself with a picture of a computer, and it has an alias. All of these properties are native to this built-in .NET object. A sensor, however, is represented by a custom object that stores all applicable information about the sensor. This information is pulled from the hosts.xml file and is used for all operations referring to a sensor on this form. The object, RevTreeNode, inherits from the TreeNode object

and serves as a container for applicable information about a sensor. The image property of this object can either be set to a red circle or a green check, indicating whether or not the application has connected to the sensor within the current session.

The first tab, in the TabControl, shows an exact copy of the alerts generated by Snort for the specific sensor in a System.Windows.Form.DataGrid control. The fields in this DataGrid are modeled after a basic join of the Snort 'event' table and 'IPHDR' table. This textual name for this tab is 'Raw Host Alerts' since it shows only unformatted alerts from a MySQL database.

4.2.5 Database Connectivity and Synchronization

A major portion of the functionality of the ARF_Main object is connecting to, and synchronizing data between, the MySQL database server installed on the application machine and the MySQL database server installed on the sensor. Excerpts from some of these methods are shown below.

The first of these database connection methods is ConnectSingleMySQL. This function takes a connection string, stored in a RevTreeNode, and connects to a MySQL database. If the connection is successful, a MySqlConnection object is returned to the calling function, as shown in the segment below.

```

/// <summary>
///     Connect to a remote database
/// </summary>
/// <param name="conn">The Connection string</param>
/// <returns>The MySQL connection object or null if connection is not
available</returns>
private MySqlConnection ConnectSingleMySQL(string conn)
{
    try
    {
        MySqlConnection connection = new MySqlConnection(conn);
        connection.Open();
        connection.Close();
        return connection;
    }
    catch
    {
        return null;
    }
}

```

This functionality allows the application to know, with a high degree of certainty, which sensors are available for analysis.

The SyncSingle function has been shown in a pseudo format because of the length of the function. This function is executed on a RevTreeNode if a valid connection has been made with the ConnectSingleMySQL function. This function is shown below in pseudo format.

```

///<summary>
///     Sync a remote snort databases alert data to the local MySQL Server
/// </summary>
///<param name="connection">Connection String For Remote MySQL Server</param>
///<param name="SensorName">Alias of Remote Server to reference in local
database</param>
///<returns>True if the host syncs</returns>
private bool SsyncSingle(MySqlConnection connection, string SensorName)
{
    // Find Timestamp of Last Alert Logged Locally for this Sensor
    // Create transactions of the alerts from Remote DB to be copied locally
    // --> events table
    // --> iphdr table
    // Start the transaction to copy the remote records locally
    // If the transaction succeeds, commit the transaction and return true
    // if the transaction fails, rollback the transaction and return false
}

```

This function executes all transactions necessary to synchronize the application databases.

The ConnectSyncSingle, shown below, function acts as a wrapper function for the previously listed functions. Its purpose is to call other functions, if necessary. This

method is necessary for the functionality, if desired, to synchronize a single host within a network.

```
/// <summary>
///     Connects to the specified RevTreeNode and synchronizes data if need be
/// </summary>
/// <param name="m">The RevTreeNode of the sensor to synchronize</param>
private void ConnectSyncSingle(arf.RevTreeNode m)
{
    string conn = "SERVER=" + m.IP + "; DATABASE=" + m.DatabaseName + "; UID="
+ m.DatabaseUser + "; PASSWORD=" + m.DatabasePassword + ";";

    MySqlConnection SingleConnection = ConnectSingleMySQL(conn);

    if(SingleConnection != null)
    {
        // Sync the databases
        // change the icon

        if(SyncSingle(SingleConnection, m.Alias))
        {
            m.ImageIndex = 0;
            m.SelectedImageIndex = 0;
        }
    }
}
```

The final relevant database connection method, `MySQLHostConnect`, is triggered when a button titled ‘Sync Hosts’ is pressed on the `ARF_Main` form object. This function connects to all sensors that are available and synchronizes the data from their alert databases to the local database. If a connection was made and data was synchronized, then the picture next to the sensor name changes to a green check mark to show the user success or a red symbol to show failure. This function is shown below.

```
/// <summary>
///     Connect and Sync all sensors listed for all sites in hosts.xml
/// </summary>
private void MySQLHostConnect()
{
    string conn;
    MySqlConnection SingleConnection;

    foreach(TreeNode n in treeView1.Nodes)
    {
        foreach(arf.RevTreeNode m in n.Nodes)
        {
            ConnectSyncSingle(m);
        }
    }
}
```

These database functions create an easy way to connect to and synchronize data between multiple sensors over multiple networks.

4.3 Fuzzy Logic Modules

4.3.1 Purpose

The fuzzy logic section of this application creates a standard system of functions and objects to assign a level of threat to a sensor on a network. There are four separate steps required in any fuzzy control system, all of which are implemented and outlined below. The controlling class for the fuzzy logic portion of the application, FuzzyEngine, receives two inputs: a System.Data.DataSet and a System.Windows.Forms.ListBox. The DataSet holds applicable data about the sensor and the ListBox serves as mechanism to report the status of the fuzzy logic computations during analysis. With this in mind, realize that the entire fuzzy logic component of this application can be invoked with a single command as shown in the following code segment:

```
double d = HostThreatLevel(RawAlertDataset, lstXMLOutput);
```

4.3.2 Snort Adapter

Before any type of fuzzy logic operation can be performed, the input data must be transformed. The SnortAdapter class takes the input DataSet and performs operations to ascertain the value of predefined, calculable metrics that can be calculated.

The following command from the FuzzyEngine class invokes these operations and returns a typed collection of custom objects, to be explained in a later section.

```
SnortAdapter s = new SnortAdapter(RawAlertDataset);  
CrispThreatCollection c1 = s.CrispThreats;
```

Upon receiving the DataSet, the constructor for the class sorts the dataset based on threat signature and performs calculations on each collection of rows that is returned. The following metrics were chosen on during the design of the application:

1. The number of occurrences of the signature, which is calculated from a simple counting of the rows.
2. The time span of the alerts, which is calculated from the last alert subtracted from the first alert and expressed as System.TimeSpan object.
3. The average time between alerts, calculated from the time span divided by the number of occurrences.
4. The number of alerts per source IP address, calculated from the number of occurrences divided by the unique source IP addresses.
5. The severity of the alert: this metric is reserved for future versions of Snort that implement this functionality. The application uses a multiplier of one to account for future releases of Snort. Currently severity measures do not exist within the Snort application for all signatures but can be implemented by security personnel by modifying a configuration file within the application. At the time of writing, this feature is a planned enhancement for future versions of Snort.

After all calculations are completed, information is stored in a custom object named CrispThreat. This structure contains the properties:

```
private int occurrences;  
private System.TimeSpan timespan;  
private int severity_multiplier;  
private double src_ip_frequency;  
private System.TimeSpan avg_time_between;  
private int signature;
```

The CrispThreat object also contains two constructors, one of which takes all of the calculated metrics and their respective data types as input. The other takes a

CrispThreat object and creates an exact copy. Since there will be a CrispThreat object created for every unique signature that occurs in a host, it is necessary to create a container for these objects. The implementation of a collection of CrispThreat objects, CrispThreatCollection, is shown in the following set of declarations.

```
public class CrispThreatCollection : System.Collections.CollectionBase
{
    public virtual void Add(CrispThreat c)
    {
        this.List.Add(c);
    }

    public virtual CrispThreat this[int Index]
    {
        get
        {return (CrispThreat)this.List[Index];}
    }
}
```

This implementation provides the necessary protection for data and application flow as set out in the design document.

4.3.3 Fuzzification Interface

The fuzzification interface transforms the CrispThreat objects within their respective collection. Each metric within a CrispThreat object is processed by a predefined membership function and returns a fuzzy value in the interval [0,1]. The fuzzification interface is defined as a step in the fuzzy logic process and should not be confused with a .NET interface.

A programmatic interface is defined for this section of the fuzzy logic module and listed in the following code segment.

```
public interface IFuzzification
{
    FuzzyThreat MakeFuzzy(CrispThreat c);
    FuzzyThreatCollection MakeFuzzy(CrispThreatCollection i);
    FuzzyThreatCollection GetFuzzyThreats();
}
```

The methods described in the interface shown above must be implemented in the class that is used in the fuzzification process. For example, every method for fuzzification would be able to:

1. Take a CrispThreat, apply the MakeFuzzy function, and return a FuzzyThreat
2. Take a collection of CrispThreats, apply the MakeFuzzy function, and return a collection of FuzzyThreats.
3. Access the calculated FuzzyThreatCollection from anywhere in the scope of the IFuzzification object.

With this concept in mind, a FuzzificationMembershipFunction class is created that implements the interface IFuzzification. Functionality in the FuzzificationMembershipFunction class is not relevant at this point because actions are taken on the IFuzzification object. A sample of how this would be achieved is shown in the code segment below.

```
IFuzzification f = new FuzzificationMembershipFunction(c1);  
FuzzyThreatCollection f1 = f.GetFuzzyThreats();
```

For future changes of membership functions, all a programmer would need to change in the fuzzy logic modules of this application would be the name of the class after the `new` keyword listed above.

A key aspect of the functionality for this module of the fuzzy logic components is creating membership functions for each metric deemed pertinent in the design phase. The metrics and their associated membership functions are listed below, starting with the number of occurrences of a specific alert signature. The membership function created to model the level of threat based on occurrences of an alert is shown below. Notice the gradual change in the dependant variable as the independent variable rises.

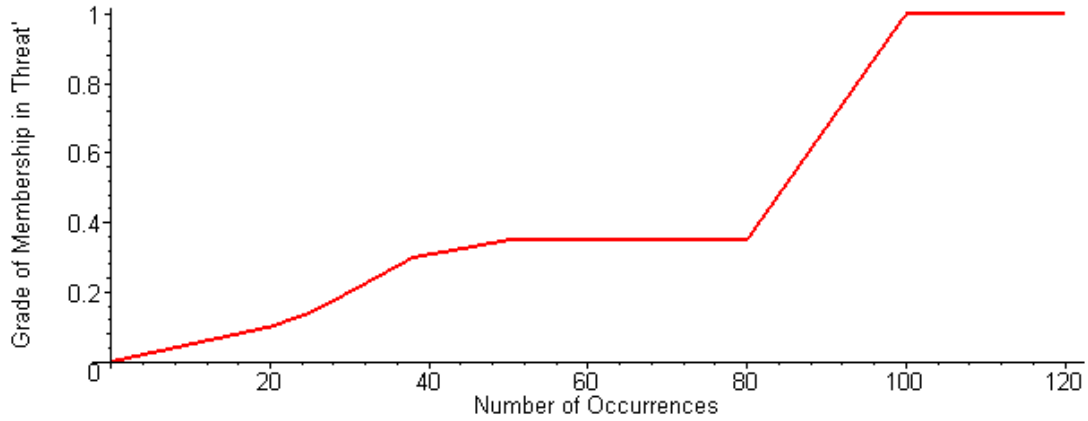


Figure 4.4: Membership Function: Number of Occurrences

The next membership function represents the span of time from the first occurrence of a signature and the last signature. This function assigns a greater level of threat if logged alerts occur in a relatively short period of time.

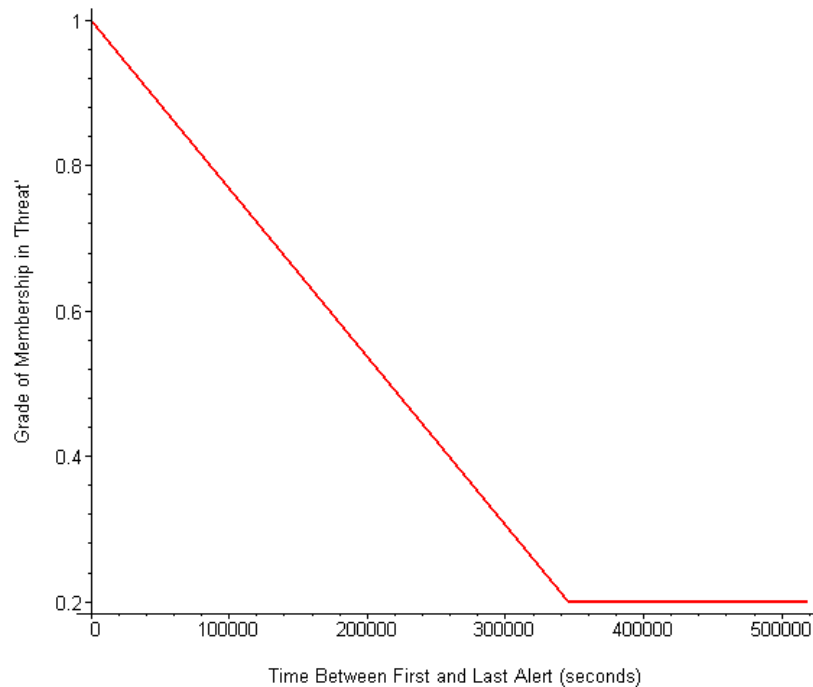


Figure 4.5: Membership Function: Timespan Between Attacks

The average time between attacks is defined as the time span between the first and last alert divided by the number of occurrences.

Another metric that is calculated from the time between attacks and the number of occurrences is defined as the average time between attacks membership function. This membership function is used in conjunction with other functions to create a rule that can be composed of several different metrics. This allows for a greater degree of control within a set of rules. The average time between attacks membership function is shown below.

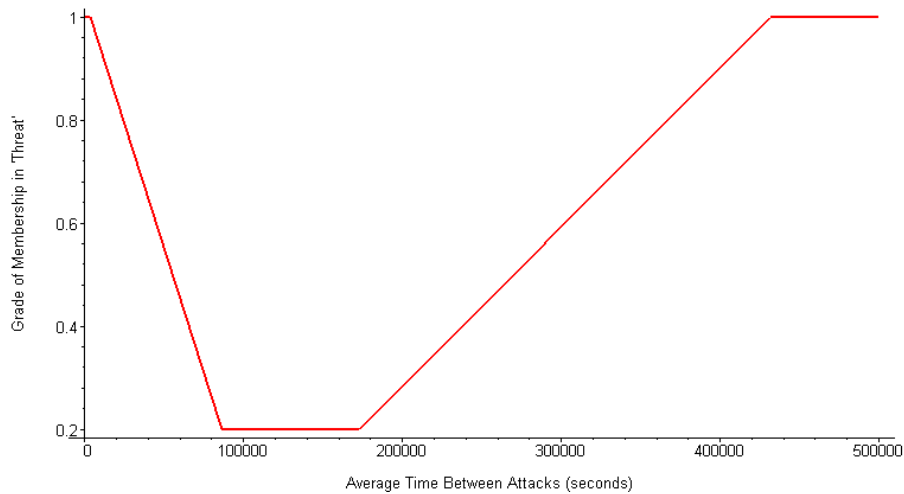


Figure 4.6: Membership Function: Average Time Between Attacks

The final metric is the membership function associated with the frequency of alerts from a specific IP address. This is defined as a stepwise function, shown below. This metric is determines the number of different source IP addresses that are generating alerts based on signature and is used to create composite rules. For example, if the average time between attacks is low and the source IP frequency is low a lower threat level could be generated because it would be assumed that the same entity is trying the same operation numerous times. The source IP frequency function is shown below in figure 4.7.

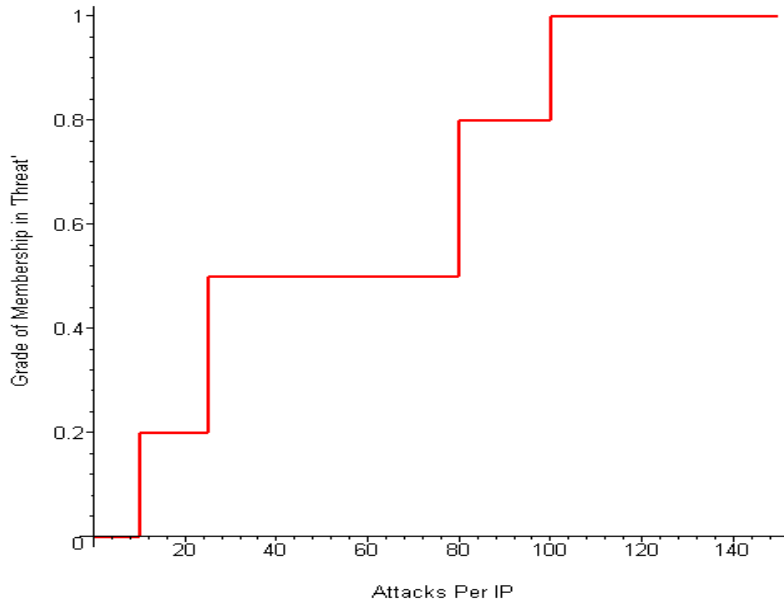


Figure 4.7: Membership Function: Attacks Per IP

The current mathematical capabilities of the application are membership functions defined by a set of points or as a polynomial function. However, this can be easily expanded by creating a new class that implements the custom `IMath` interface.

The output object created by the fuzzification interface is a collection of `FuzzyThreat` objects. This structure is represented by the following properties:

```
private double fuzz_occurrences;
private double fuzz_timespan;
private double fuzz_severity_multiplier;
private double fuzz_src_ip_frequency;
private double fuzz_avg_time_between;

private CrispThreat crisp;
private int signature;
```

These `private double` variables are the output of their respective membership functions, outlined above. This object also holds a copy of the `CrispThreat` object created in the previous function. The purpose for this feature is to allow for future calculations in the inference and defuzzification mechanisms.

4.3.4 Fuzzy Inference Machine

The fuzzy inference machine is the heart of the fuzzy logic modules in this application. It takes the fuzzy objects for each signature, calculated previously, and applies a set of rules to create a level of threat for a sensor. This step also creates an XML log file detailing the threat levels of all the objects before they are defuzzified.

The inference machine must implement the interface listed below

```
public interface IIInferenceEngine
{
    FuzzyThreatLevelCollection
    FuzzyInference(FuzzyThreatCollection i);

    FuzzyThreatLevel          FuzzyInference(FuzzyThreat f);
    FuzzyThreatLevelCollection GetFuzzyThreatLevel();
}
```

The methods described in this interface provide functionality, respectively, to:

1. Take a collection of FuzzyThreat objects and return a collection of FuzzyThreatLevel objects that are created with a rules-based system.
2. Take a single FuzzyThreat object and return a single FuzzyThreatLevel object created with a rules-based system.
3. Access the calculated FuzzyThreatLevel collection from anywhere in the scope of the IIInferenceEngine.

Using this interface, the following function call is performed in the FuzzyEngine class to execute this step of the fuzzy logic process.

```
IIInferenceEngine i = new RuleBasedInference(f1);
FuzzyThreatLevelCollection f2 = i.GetFuzzyThreatLevel();
```

Each IIInferenceEngine engine has an associated set of rules. These rules make all inferences about threat levels. This set of rules maps a threat level to a specific signature

based on the metrics calculated from the set of alerts. An excerpt from a sample rule is shown in the code listed below.

```
// 'WEB-MISC /home/www access'  
// http://www.snort.org/pub-bin/sigs.cgi?sid=1671  
// Low threat  
private double WEB_MISC_ACCESS(FuzzyThreat f)  
{  
    double threat = startlow;  
    if(this.IsHigh(f.Src_Ip_Frequency) && this.IsLow(f.Avg_Time_Between))  
        threat += .5;  
  
    return threat;  
}
```

This alert was evaluated to have a low threat level, however, it states that if the threat from the source IP addresses are high and the threat calculated from the average time between is low, then the threat level should be elevated. These rules, along with all others, form an integral step in the fuzzy logic process. They dictate how alerts should be treated, and allow the application to assign a grade of membership in the fuzzy category, threat.

After the FuzzyThreatLevelCollection is created from the set of rules, the contents of the collection are used to create an XML document for System.Windows.Forms objects to display as needed. This document follows the format shown below:

```
<host name="Insert Sensor Alias Here" time="4/2/2006 4:28:24 PM">  
    <signature sig="1">  
        <fuzzy_value>.64</fuzzy_value>  
    </signature>  
    .....
```

Since the fuzzy logic modules of the application do not and should not necessitate a database connection, the signature is outputted in a numerical format. This functionality gives a more detailed look at exactly what alerts are considered the most threatening with respect to a sensor on a network.

The output object created by the fuzzy inference mechanism is a collection of FuzzyThreatLevel objects. This custom structure is represented by the following properties:

```
double fuzzy_threat_level;  
int signature;
```

This object represents a threat level for a specific signature that has been calculated based on metrics found in previous objects being subjected to signature-based rule.

4.3.5 Defuzzification Interface

The final step in the fuzzy logic process is the defuzzification process. This step makes a calculation on each FuzzyThreatLevel object and returns a number that signifies a level of threat for a sensor. The defuzzification process implements the interface listed below.

```
public interface IDefuzzifier  
{  
    double Defuzzify(FuzzyThreatLevelCollection i);  
    double GetCrispOutput();  
}
```

This interface exists because of the number of methods to defuzzify a fuzzy set. The method chosen to defuzzify the FuzzyThreatLevelCollection object is the Mean of Maximum. This method takes a simple average of all fuzzy threat levels and returns the result.

```
public double Defuzzify(FuzzyThreatLevelCollection i)  
{  
    int count = 0;  
    double dValue= 0.0;  
    foreach(FuzzyThreatLevel t in i)  
    {  
        count++;  
        dValue += t.Fuzzy_Threat_Level;  
    }  
    return dValue/(double)count;  
}
```


The value calculated at by the Defuzzify function is the value that is returned to the calling System.Windows.Forms object, which is designated as the threat level for the application.

There are several methods available for defuzzification, with no clear methodology on how to pick a defuzzification method. Other methods include, returning the maximum membership value for a fuzzy set and a method to compute the center of area for all overlaid membership functions with the dependant variable specified on the appropriate axis. Changing to one of these methods would drastically change the experimental outputs of this application, not necessarily for the better. Returning the maximum value in the fuzzy set would increase the weight on low threat objects in a similar fashion as higher priority threats, causing a reduction of reliability of output data. The center of area method could be an appropriate solution but was not chosen due to the mathematical complexity of the algorithm.

4.4 Diagrams

4.4.1 ARF Class Diagram

The following diagram shows the basic flow between the fuzzy logic classes. The actual implementation varies slightly from what is shown in one respect. The FuzzyEngine class is acting as a controlling class and information is actually sent back to the FuzzyEngine class before proceeding to the next step. For this diagram to be programmatically accurate there would be a call and response between the FuzzyEngine class and each step of the fuzzy logic process. However, it has been determined that the application flow and object manipulation is described with more detail in the following diagram, figure 4.8.

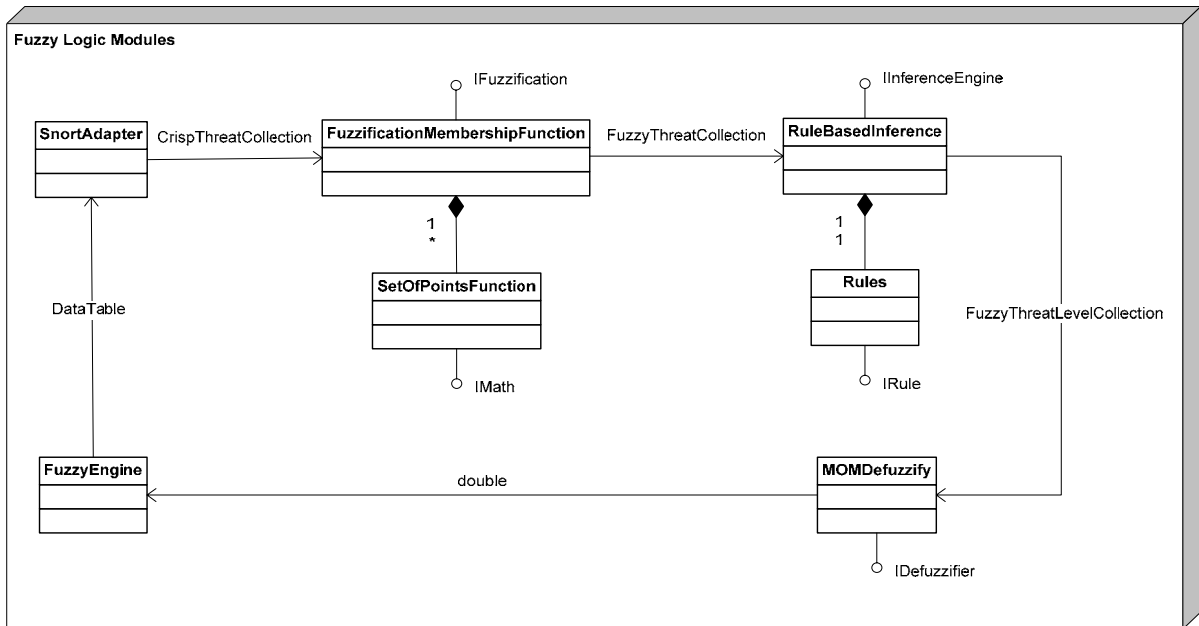


Figure 4.8: Fuzzy Logic Class Diagram

Some typical intrusion detection sensor placements are shown in the following diagram, figure 4.9.

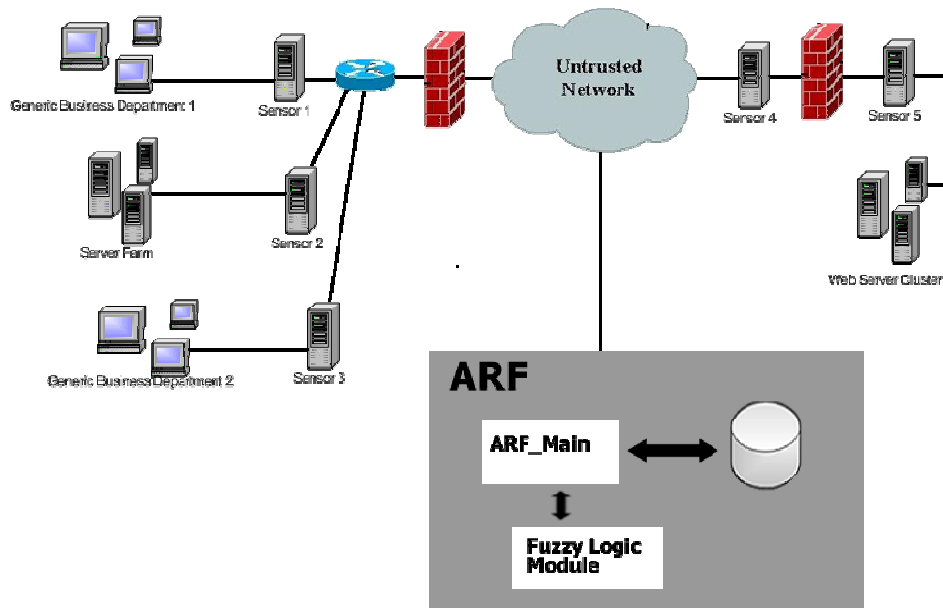


Figure 4.9: High Level Function of the ARF

The application is designed to be able to analyze information in any sensor with which there is a logical connection. This diagram also illustrates the separation between the data and application. With a simple modification, the database used for the application could be positioned anywhere a network connection can exist.

CHAPTER V – TESTING

5.1 Introduction

Testing of the ARF threat evaluation system will be demonstrated in a series of case studies. An experiment will be described in the next section to be executed for each case study. While these case studies will be informative, it should be noted that the performance of this application is only as good as the rule-sets utilized. The purpose of these tests is to gather evidence that will prove the effectiveness of the rule-sets in question. This is necessary because a side-by-side comparison of the Snort alert log and the ARF fuzzy threat level is not feasible.

5.2 Testing Design Parameters

Each case study will analyze the fuzzy threat level generated with several different combinations of membership functions and rule-sets. A separate dataset will be utilized for each case study; however, the experimental parameters will remain the same throughout the testing process. For each case study, four experiments are to be executed. Each of these experiments requires two entities. The first of which is a set of membership functions and the second is a set of rules.

There are two sets of membership functions. The first set of membership functions is very general and would, theoretically, ship with the application. The second set of membership functions is specific and takes measures to handle rules in a more case by case nature. The rule-sets also follow this pattern.

The experiments are to be executed in the following pattern as shown in table 5.1, below.

Table 5.1: Experimental Parameters

Membership Function	Rule-Set	Output
General	General	O ₁
General	Specific	O ₂
Specific	General	O ₃
Specific	Specific	O ₄

After the four experiments outlined above are executed on a dataset defined in each case study, data for analysis needs to be outputted for analysis. This data will be written to an Excel spreadsheet. The table shown above, an output of the alert dataset, and the final state of all application objects are recorded in this spreadsheet. This serves as a long-term record of the experiment and will aid in analysis.

5.3 Case Study 1 (All Experimental Data)

This study represents a dataset collected in just under 27 days. There are approximately two thousand alerts represented in the dataset ranging all levels of threat from low to high.

5.3.1 Overview of Alerts

This dataset represents alerts whose threat levels range from low to high. Low threat alerts are typically defined as actions whose purpose is to collect information about the target machine. These alerts can be as benign as someone searching for a directory structure of a target machine or can act as a first step to a more serious threat. A high threat, however, is typically an attempt to exploit a known vulnerability in an application or system. The outcome of these high threat vulnerabilities varies, but can lead to execution of arbitrary code with privileges on a target machine, denial of service, or a compromised file structure.

5.3.2 Rule-Sets Defined

There are two separate rule-sets that are used throughout the testing process. These sets can be seen in their entirety in the appendices and will be discussed here in a general fashion. Any set of rules defined in this application must implement the `IRule` interface and inherit from the `RuleBase` class. These two objects allow each rule-set to function with the same basic operations and allow effortless integration into ARF.

Each set of rules has a general format that should be followed. First, a variable threat is initialized to a floating point threat level between 0 and 1.0. After that, the threat level is incremented or decremented based on a set of rules. The rules use the `FuzzyThreat` object in question to make a determination of whether or not the threat level will be increased or decreased. After all adjustments have been made for the alert, a function checks the bounds of the threat to make sure it lies within the interval $[0,1]$ and returns the threat level for the set of alerts. This format would be considered best practice, but the rule based inference mechanism used by ARF is flexible enough to cater to any specific changes for each rule.

The specific rule-set defines a rule for every known signature. These rules take into account the severity of the alert as well as the fuzzified input that is calculated in previous portions of the application. Unknown alerts can be handled in a uniform fashion, but for testing purposes unknown alerts do not exist. The general rule-set uses a single rule to infer the threat level for an alert. This rule is more descriptive than a single rule because it must serve a wider spectrum of alerts. Each alert is run through a series of tests that will increment or decrement the threat level in the same fashion that is described above.

Some examples of rules used in this application are shown below. The first example is the heart of the general rule-set, acting as a “catch-all” rule for the application.

```
private double GRule(FuzzyThreat f)
{
    double threat = startmed;

    // If anything is a high threat, elevate the threat level
    if(this.IsHigh(f.Occurances) ||
        this.IsHigh(f.Src_Ip_Frequency) ||
        this.IsHigh(f.Timespan) ||
        this.IsHigh(f.Avg_Time_Between)
    )
        threat += .1;

    // If a lot of different IPs are doing it in a low amount of
    // time...its probably new and needs to be looked at
    if(this.IsHigh(f.Src_Ip_Frequency) &&
        this.IsLow(f.Avg_Time_Between))
        threat += .5;

    // If a lot of different IPs are doing over a large timespan
    // the threat should be elevated

    if(this.IsHigh(f.Src_Ip_Frequency) &&
        this.IsMedHigh(f.Avg_Time_Between))
        threat += .2;

    // If not a lot of people are doing it over not a lot of time,
    // decrement the threat level
    if(this.IsLowMed(f.Src_Ip_Frequency) && this.IsLow(f.Timespan))
        threat -= .3;

    // If it is not happening a lot, decrement the threat level
    if(this.IsLowMed(f.Occurances))
        threat -= .1;

    threat = CheckBounds(threat);
    return threat;
}
```

As shown in the code listed above, the threat level for an unknown alert is initialized to .5, run through a series of tests to increment or decrement the threat level, and checked to make sure the level remains within the interval (0,1). While this rule may seem verbose, it should be noted that it only needs to be implemented one time and not for every available alert.

The specific rules are generally much shorter, as shown in the code listing below.

```

//'WEB-CGI calendar access'
// http://www.snort.org/pub-bin/sigs.cgi?sid=882
// Low / No Threat
private double WEB_CGI_CALENDAR_ACCESS(FuzzyThreat f)
{
    // This application does not exist on the system
    return 0.0;
}

```

This is the rule defined for the specific alert created by the snort development team and numbered 882. This threat exists as a flaw in an open-source calendar application, which does not exist on the target machine, therefore poses no threat. However, the code shown below shows a reasonable rule for a specific alert.

```

//'WEB-MISC Invalid HTTP Version String' (2570)
// http://www.snort.org/pub-bin/sigs.cgi?sid=2570
// High Threat
private double WEB_MISC_INVALID_HTTP_VERSION_STRING(FuzzyThreat f)
{
    double threat = starthigh;
    if(this.IsMedHigh(f.Src_Ip_Frequency))
        threat += .2;
    if(this.IsHigh(f.Timespan))
        threat += .1;
    threat = this.CheckBounds(threat);
    return threat;
}

```

Note that this rule follows the same basic pattern defined previously, an initialization, a calculation, and a bounds checking mechanism.

5.3.3 Membership Functions Defined

There are two separate sets of membership functions used throughout the testing process. The sets can be seen in their entirety in the appendices and will be discussed here in a general fashion. Each set of membership functions need only implement the IFuzzification interface to ensure operability with the application.

Each set of membership functions should also implement functionality to deal with every metric defined in design phase. Function stubs are shown below:


```
private double MembershipOccurrences(int occurrences);  
private double MembershipTimespan(TimeSpan t);  
private double MembershipTimeBetween(TimeSpan t);  
private double MembershipSrcIPFreq(double ip_fre);
```

These function stubs take a crisp input and create a fuzzy output based on the operations performed within the function.

The first set of membership functions to be described is the general membership functions. These functions follow a straight line pattern shown below in figure 5.1. In this example, the maximum threat level plateaus at 0.8.

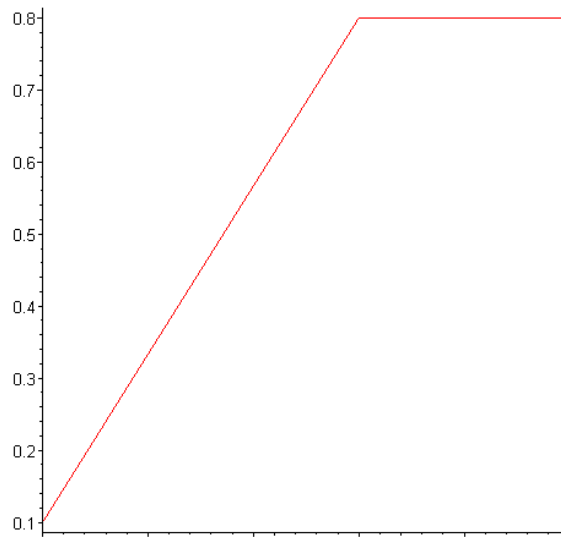


Figure 5.1: Trend for Set of General Membership Functions

The only variation in the general trend shown in figure 5.1 is the boundary where the highest threat starts and the subsequent slope of the line that is calculated from the two points.

For the specific set of membership functions, other graph structures were used and are described in figures 4.4 – 4.7.

5.3.4 Experimental Results

The output of each experiment is shown below, in table 5.2, in the format described in section 5.2 of this document.

Table 5.2: Experimental Results of Case Study 1 (Entire Alert Log)

Experiment Number	Membership Function	Rule Set	Fuzzy Threat Level
1	General	General	0.63
2	General	Specific	0.51
3	Specific	General	0.37
4	Specific	Specific	0.38

5.3.5 Analysis

The results of this case study show a declining threat level for the first two experiments and similar threat levels for the third and fourth experiments. The varying property for the third and fourth experiments is the rule-set and the similarity of the fuzzy threat levels would indicate that a general rule set may be all that is necessary for an effective analysis of data for this mix of results. These results, while unexpected, may serve to simplify the customization process of ARF over a series of sensors.

5.4 Case Study 2 (Low Threat Dataset)

This case study represents all alerts that are deemed “low threat” that exist in the dataset collected for case study 1. As previously stated, these threats are considered to interrogate a system rather than compromise.

All alerts that were deemed as low threats were placed into a separate local database that represents a sensor with all low threat alerts. This database will be used to simulate a remote sensor. The output of the experiment on a set of alerts that represents a low threat is shown below in table 5.3.

Table 5.3: Experimental Results of Case Study 2 (Low Threat Alerts)

Experiment Number	Membership Function	Rule Set	Fuzzy Threat Level
1	General	General	0.5
2	General	Specific	0.24
3	Specific	General	0.4
4	Specific	Specific	0.22

The outputs of this experiment show that the dominant experimental entity in this case study is the rule-set. Notice the similarity of the Fuzzy Threat Levels in experiments one and three as well as two and four. This leads to the conclusion that the membership functions are not as important and could possibly be neglected as a primary area of research. However, another case is also possible. These results could be grouped in this particular fashion because the specific rule set has the capability to properly categorize threats that have been considered low.

5.5 Case Study 3 (Medium Threat Dataset)

This case study represents all alerts that are deemed “medium threat” that exist in the dataset collected for case study 1. All medium threat alerts from case study 1 were copied into a local database that represents a sensor with a medium threat level. The output of the experiment on a set of alerts that represents a medium threat is shown below in table 5.4.

Table 5.4: Experimental Results of Case Study 3 (Medium Threat Alerts)

Experiment Number	Membership Function	Rule Set	Fuzzy Threat Level
1	General	General	1
2	General	Specific	0.67
3	Specific	General	0.3
4	Specific	Specific	0.57

The results of the experiments for case study 3 are not as clear as the first two case studies. There is no obvious trend shown in this data and few conclusions can be drawn from this case study. The proximity of the outputs with varying membership functions in experiment two and four could serve to solidify the assumption that a general membership function maybe be utilized to categorize alerts. The amount of difference between the second and fourth experiment (.1) is relatively high to make this assumption

based on this study alone. Based on this case study, it can be inferred that a different approach to medium threat alerts should be investigated in the future.

5.6 Case Study 4 (High Threat Dataset)

This case study represents all alerts that are deemed “high threat” that exist in the dataset collected for case study 1. All high threat alerts from case study 1 were copied into a local database that represents a sensor with a high threat level. The output of the experiment on a set of alerts that represents a high threat is shown below in table 5.5.

Table 5.5: Experimental Results of Case Study 4 (High Threat Alerts)

Experiment Number	Membership Function	Rule Set	Fuzzy Threat Level
1	General	General	0.4
2	General	Specific	0.8
3	Specific	General	0.43
4	Specific	Specific	0.7

Once again, it can be seen from the results that fuzzy threat levels group together based on the rule-set in question. This follows the trend shown in case study 2. These results could be grouped in this particular fashion because the specific rule set has the capability to properly categorize threats that have been considered high. While the difference in output levels for the second and fourth experiments is relatively high, .1, it should be noted that they both reside within the interval defined as high.

5.7 Final Analysis

Boundaries for low threats have been defined as a threat level of $0 < \text{threat} \leq .33$. Medium and high threats follow the same pattern: $.33 < \text{threat} \leq .66$ and $.66 < \text{threat} < .99$, respectively. It could be assumed that the optimal combination of rule-sets and

membership functions would be the combination that, over all experiments, would show an increase in threat level from the low threat dataset to the high threat dataset. There are two separate experiments that follow this pattern. The first of which is experiment 2 that used a general membership function and a specific set of rules. The second is experiment 4 that used a specific set of membership functions and a specific set of rules. The output of these experiments is shown below in figures 5.2 and 5.3.

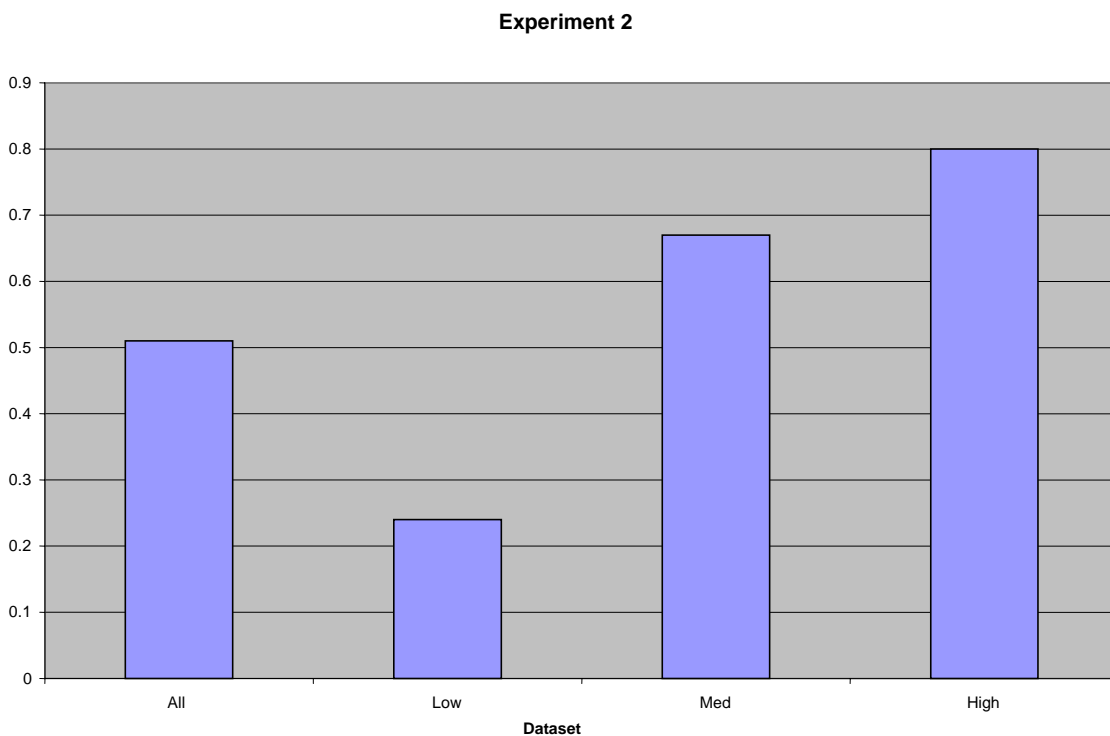


Figure 5.2: Experiment 2 Data (General Membership Functions, Specific Rule-Set)

Figure 5.2 shows a desired trend in data, however the threat levels of the medium dataset do not fall within the range of $(.33,.66]$.

Figure 5.3, shown below, has all the characteristics of a successful test. All expected values meet the actual values and the escalating threat level across datasets

shows that the combination of a specific set of membership functions as well as a specific set of rules is the optimal situation for ARF to function effectively.

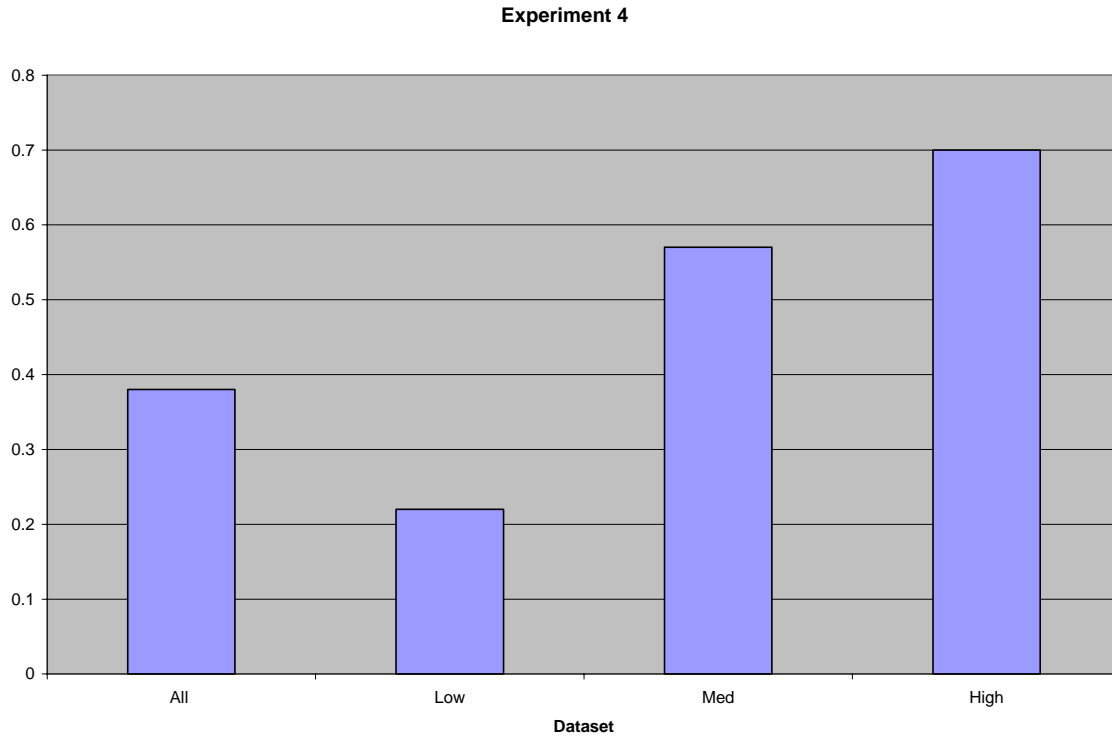


Figure 5.3: Experiment 4 Data (Specific Membership Functions, Specific Rule-Set)

The experimental results depicted in Figure 5.3, above, show a threat level that is slightly lower than in Figure 5.2. The low threats as well as the high threat datasets are roughly in the middle of the range that was expected.

CHAPTER VI – CONCLUSIONS AND FUTURE RESEARCH

The ARF threat evaluation system was developed as an application layer fuzzy logic analysis tool. Specifically, the application examines the alert logs of multiple sensors and uses a rule-based fuzzy logic inference engine to generate a level of threat in a sensor. ARF is designed to be scalable, portable, and customizable. The design called for an application that could be tuned to any environment by someone who has an existing knowledge of Snort signatures. The design also dictated two separate software entities, one for all front end database and GUI functionality and another entity dedicated to the fuzzy logic engine. The brain of the fuzzy logic engine resides in a set of membership functions and a set of rules. Through experimentation, it was found that the more specific the membership functions and rule-sets are, the more accurate the output of the application becomes. This conclusion, while somewhat intuitive, leads to the first area of future research. A non-probabilistic learning system that can change the boundaries set within each membership function could be developed. A similar learning system could be implemented for a set of rules. These learning systems could drastically reduce or eliminate customization in the application. With these systems in place ARF could be a self-contained entity usable to anyone, instead of just knowledgeable security personnel.

Another interesting area of research that would improve the reliability of this application is the use of the Nessus Attack Scripting Language (NASL). Existing scripts, publicly available at www.nessus.org can test the vulnerability of a system for a specific alert. These scripts are used at the kernel level to test the vulnerability of a system and if the vulnerability does not exist, Snort would not log the alert. While this topic is not

directly related to fuzzy logic, the outcome of this Snort enhancement would work to reduce the MySQL storage space necessary and speed up the fuzzy logic process within ARF.

In the current state, ARF calculates a threat level for a single sensor. Further work could be done to correlate alerts across multiple sensors to check for distributed attacks and give a threat level for an entire network. This enhancement would essentially turn Snort into a hybrid intrusion detection system, giving security personnel not only a robust host based system but also a network based system.

With any application, as the amount of data increases the amount of processing time to analyze the data also increases. In the future it will be necessary to adopt a new method of storing and calculating results from the fuzzy logic engine. A dynamic programming approach could possibly be implemented in this application to only analyze new alerts within the MySQL log. This method of storing results for small portions of data to be looked up later, instead of being recalculated, could radically decrease processing time in the application.

REFERENCES

1. Bace, R.; Intrusion Detection, Macmillan Technical Publishing, 2nd Edition, Indiana, 2000
2. Kemmerer, R.A.; Vigna, G.; “Intrusion Detection: A Brief History and Overview”, IEEE Xplore Computer, Volume 35, Issue 4, Part Supplement, April 2002 Page(s):27 – 30
3. Dorothy E. Denning. "An Intrusion-Detection Model," *sp*, p. 118, 1986 IEEE Symposium on Security and Privacy, 1986.
4. Innella, R.; McMillan, O.; “An Introduction to Intrusion Detection Systems”, <http://www.securityfocus.com/infocus/1520>, updated 12-6-2001, accessed 10-5-2005
5. Topallar, M.; Depren, M.O.; Anarim, E.; Ciliz, K. “Host-based intrusion detection by monitoring Windows registry accesses” IEEE Signal Processing and Communications Applications Conference, 2004, Publication Date: 28-30 April 2004, On page(s): 728 - 731
6. Mukherjee, B.; Heberlein, L.T.; Levitt, K.N.; “Network intrusion detection” IEEE Network, Publication Date: May/Jun 1994, Volume: 8, Issue: 3, On page(s): 26-41
7. Herringshaw, C.; “Detecting Attacks on Networks”, IEEE Computer, 1997, Volume 30, Issue 12, Dec. 1997 Page(s):16 – 17
8. Marchette, D.; “Computer Intrusion Detection and Network Monitoring : A Statistical Viewpoint”, Springer Verlag, July 2001

9. Fenner F et al. Smallpox and its eradication. Geneva, World Health Organization, 1988
10. Can infectious diseases be eradicated? Symposium papers. Reviews of infectious diseases, 1982, 4: 913-984.
11. The American Heritage® Dictionary of the English Language, Fourth Edition
Copyright © 2000 by Houghton Mifflin Company.
12. WarRoom Research LLC. “1996 Information Systems Security Survey”.
13. Computer Security Institute’s 1998 Computer Crime Survey
14. Forrest S, Hofmeyr S A, Somajayi A. “Computer Immunology” Communications of the ACM, 40(10): 88-96, 1997.
15. Immunity by Design: An Artificial Immune System. S. Hofmeyr and S. Forrest.
16. Zadeh, Lofti A. “Fuzzy Logic”, IEEE Computer April 1988, On page(s): 83-92
17. Zadeh, Lofti A. “Making Computers Think like People”, IEEE Spectrum August 1984, On page(s): 26-32
18. Zadeh, Lofti A. “Fuzzy Logic = Computing with Words”, IEEE Transactions on Fuzzy Systems, Vol 4 No. 2 May 1996, On page(s): 103-111
19. Yan, Jun; Ryan, Michael; Power, James; “Using Fuzzy Logic” Prentice Hall International 1994
20. Yagishita, O., et al., “Application of fuzzy reasoning to the water purification proves” in Industrial Applications of Fuzzy Control, ed. by Sugeno, M., 1985, 19-40
21. Yasunobu, S., et al., “Automatic train operation by predicative fuzzy control”, in Industrial Application of Fuzzy Control, ed. by Sugeno, M., 1985, 1-18

22. Sugeno, M., "Industrial Application of Fuzzy Control", North-Holland, 1985
23. Castro, J. L.; "Fuzzy Logic Controllers Are Universal Approximators", IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL 25, NO. 4, APRIL 1995, On page(s): 629-635
24. W. Frawley and G. Piatetsky-Shapiro and C. Matheus, Knowledge Discovery in Databases: An Overview. AI Magazine, Fall 1992, pp. 213-228.
25. Lee, Wenke; Stolfo Salvatore J., "Data Mining Approaches for Intrusion Detection"; 7th USENIX Security Symposium, 1998 Pp. 79-94 of the Proceedings.
26. Mehmed Kantarzik, Personal Interview, 2-10-2006
27. Ryan, Jake; Meng-Jang Lin; Miikkulainen, Risto; "Intrusion Detection with Neural Networks"
28. http://www.snort.org/about_snort/. Accessed 2/21/2006
29. <http://www.net-security.org/article.php?id=860>. Accessed 2/24/2006
30. Carrettoni, F; Castano, S; Martella, G; Samarati, P; "RETISS: A Real Time Security System for Threat Detection using Fuzzy Logic"
31. Dickerson, John E; Juslin Jukka; Koukousoula, Ourania; Dickerson, Julie; "Fuzzy Intrusion Detection"
32. Luo, Jianxiong; Bridges, Susan M.; "Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection"; International Journal of Intelligent Systems; Volume: 15; No: 8, pages(s): 687-703; copyright 2000 John Wiley & Sons, Inc.

33. <http://afrodita.unicauca.edu.co/~cbedon/snort/snortdevdiagrams.html>. Created: April 14, 2005. Accessed: April 24, 2006
34. <http://ieeexplore.ieee.org/iel2/575/5225/00202210.pdf?isnumber=&arnumber=202210>. Created: October 3, 1991. Accessed: April 24, 2006
35. <http://www.ciscopress.com/articles/article.asp?p=25327&seqNum=4&rl=1>. Created: February 15, 2002. Accessed: April 24, 2006.
36. <http://en.wikipedia.org/wiki/Image:LTI.png>. Created December 27, 2005. Accessed: April 24, 2006.
37. Yu, Yingbing. "Anomaly Intrusion Detection and Threat Evaluation using Artificial Immunity Model and Fuzzy Logic". University of Louisville Graduate School Dissertation. May 2005.

APPENDIX I – TESTING DOCUMENTS

AllAlerts.xls

This document serves as a log for the experimental outputs of case study 1. There are six separate excel worksheets associated with this file to be named and described below.

Totals

This worksheet shows the output of all four experiments for this case study

Experimental Outputs

Generated: 4/18/2006 2:52:44 PM

Experiment Number	Membership Function	Rule Set	Fuzzy Threat Level
1	General	General	0.63
2	General	Specific	0.51
3	Specific	General	0.37
4	Specific	Specific	0.38

Experiment 1 – Application State

Shows the final state of all custom application objects used during the first experiment.

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	3	32	1	1.50	50	2
2	1003	45	1	77.15	42	4
3	262	9	1	13.10	13	15
4	9	42	1	4.50	44	16
5	6	3	1	3.00	10	882
6	1	0	1	1.00	0	1002
7	99	20	1	14.14	8	1070
8	6	35	1	6.00	35	1112
9	18	21	1	6.00	7	1113
10	102	20	1	14.57	36	1248
11	102	20	1	14.57	36	1288
12	2	14	1	2.00	7	1671
13	102	20	1	14.57	36	1807
14	61	6	1	1.27	20	1852
15	187	0	1	6.93	35	2570
16	2	1	1	2.00	0	3463
17	2	1	1	2.00	0	3813

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.31	0.80	1.00	0.28	0.80	2
2	0.80	0.80	1.00	0.80	0.12	4

3	0.80	0.80	1.00	0.80	0.17	15
4	0.73	0.80	1.00	0.63	0.58	16
5	0.52	0.12	1.00	0.45	0.11	882
6	0.17	0.10	1.00	0.22	0.10	1002
7	0.80	0.80	1.00	0.80	0.21	1070
8	0.52	0.80	1.00	0.80	0.80	1112
9	0.80	0.80	1.00	0.80	0.80	1113
10	0.80	0.80	1.00	0.80	0.20	1248
11	0.80	0.80	1.00	0.80	0.20	1288
12	0.24	0.10	1.00	0.33	0.10	1671
13	0.80	0.80	1.00	0.80	0.20	1807
14	0.80	0.80	1.00	0.25	0.40	1852
15	0.80	0.80	1.00	0.80	0.11	2570
16	0.24	0.10	1.00	0.33	0.10	3463
17	0.24	0.10	1.00	0.33	0.10	3813

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0.5	2
2	1	4
3	1	15
4	0.6	16
5	0.1	882
6	0.1	1002
7	1	1070
8	0.7	1112
9	0.8	1113
10	1	1248
11	1	1288
12	0.1	1671
13	1	1807
14	0.6	1852
15	1	2570
16	0.1	3463
17	0.1	3813

Experiment 2 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	3	32	1	1.50	50	2
2	1003	45	1	77.15	42	4
3	262	9	1	13.10	13	15
4	9	42	1	4.50	44	16
5	6	3	1	3.00	10	882
6	1	0	1	1.00	0	1002
7	99	20	1	14.14	8	1070

8	6	35	1	6.00	35	1112
9	18	21	1	6.00	7	1113
10	102	20	1	14.57	36	1248
11	102	20	1	14.57	36	1288
12	2	14	1	2.00	7	1671
13	102	20	1	14.57	36	1807
14	61	6	1	1.27	20	1852
15	187	0	1	6.93	35	2570
16	2	1	1	2.00	0	3463
17	2	1	1	2.00	0	3813

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.31	0.80	1.00	0.28	0.80	2
2	0.80	0.80	1.00	0.80	0.12	4
3	0.80	0.80	1.00	0.80	0.17	15
4	0.73	0.80	1.00	0.63	0.58	16
5	0.52	0.12	1.00	0.45	0.11	882
6	0.17	0.10	1.00	0.22	0.10	1002
7	0.80	0.80	1.00	0.80	0.21	1070
8	0.52	0.80	1.00	0.80	0.80	1112
9	0.80	0.80	1.00	0.80	0.80	1113
10	0.80	0.80	1.00	0.80	0.20	1248
11	0.80	0.80	1.00	0.80	0.20	1288
12	0.24	0.10	1.00	0.33	0.10	1671
13	0.80	0.80	1.00	0.80	0.20	1807
14	0.80	0.80	1.00	0.25	0.40	1852
15	0.80	0.80	1.00	0.80	0.11	2570
16	0.24	0.10	1.00	0.33	0.10	3463
17	0.24	0.10	1.00	0.33	0.10	3813

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0.2	2
2	0.4	4
3	0.6	15
4	1	16
5	0	882
6	0.1	1002
7	0.6	1070
8	0.6	1112
9	0.8	1113
10	0.8	1248
11	0.4	1288
12	0.1	1671
13	0.6	1807

14	0.1	1852
15	1	2570
16	0.7	3463
17	0.7	3813

Experiment 3 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	3	32	1	1.50	50	2
2	1003	45	1	77.15	42	4
3	262	9	1	13.10	13	15
4	9	42	1	4.50	44	16
5	6	3	1	3.00	10	882
6	1	0	1	1.00	0	1002
7	99	20	1	14.14	8	1070
8	6	35	1	6.00	35	1112
9	18	21	1	6.00	7	1113
10	102	20	1	14.57	36	1248
11	102	20	1	14.57	36	1288
12	2	14	1	2.00	7	1671
13	102	20	1	14.57	36	1807
14	61	6	1	1.27	20	1852
15	187	0	1	6.93	35	2570
16	2	1	1	2.00	0	3463
17	2	1	1	2.00	0	3813

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.02	0.20	1.00	0.00	1.00	2
2	1.00	0.20	1.00	0.20	1.00	4
3	1.00	0.20	1.00	0.20	0.95	15
4	0.05	0.20	1.00	0.00	0.46	16
5	0.03	0.99	1.00	0.00	1.00	882
6	0.01	1.00	1.00	0.00	1.00	1002
7	0.97	0.20	1.00	0.20	0.91	1070
8	0.03	0.20	1.00	0.00	0.20	1112
9	0.09	0.20	1.00	0.00	0.20	1113
10	1.00	0.20	1.00	0.20	0.91	1248
11	1.00	0.20	1.00	0.20	0.91	1288
12	0.01	1.00	1.00	0.00	1.00	1671
13	1.00	0.20	1.00	0.20	0.91	1807
14	0.35	0.20	1.00	0.00	0.67	1852
15	1.00	0.54	1.00	0.00	1.00	2570
16	0.01	1.00	1.00	0.00	1.00	3463
17	0.01	1.00	1.00	0.00	1.00	3813

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0.4	2
2	0.2	4
3	0.3	15
4	0.4	16
5	0.4	882
6	0.4	1002
7	0.3	1070
8	0.4	1112
9	0.4	1113
10	0.3	1248
11	0.3	1288
12	0.4	1671
13	0.3	1807
14	0.5	1852
15	0.5	2570
16	0.4	3463
17	0.4	3813

Experiment 4 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	3	32	1	1.50	50	2
2	1003	45	1	77.15	42	4
3	262	9	1	13.10	13	15
4	9	42	1	4.50	44	16
5	6	3	1	3.00	10	882
6	1	0	1	1.00	0	1002
7	99	20	1	14.14	8	1070
8	6	35	1	6.00	35	1112
9	18	21	1	6.00	7	1113
10	102	20	1	14.57	36	1248
11	102	20	1	14.57	36	1288
12	2	14	1	2.00	7	1671
13	102	20	1	14.57	36	1807
14	61	6	1	1.27	20	1852
15	187	0	1	6.93	35	2570
16	2	1	1	2.00	0	3463
17	2	1	1	2.00	0	3813

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.02	0.20	1.00	0.00	1.00	2

2	1.00	0.20	1.00	0.20	1.00	4
3	1.00	0.20	1.00	0.20	0.95	15
4	0.05	0.20	1.00	0.00	0.46	16
5	0.03	0.99	1.00	0.00	1.00	882
6	0.01	1.00	1.00	0.00	1.00	1002
7	0.97	0.20	1.00	0.20	0.91	1070
8	0.03	0.20	1.00	0.00	0.20	1112
9	0.09	0.20	1.00	0.00	0.20	1113
10	1.00	0.20	1.00	0.20	0.91	1248
11	1.00	0.20	1.00	0.20	0.91	1288
12	0.01	1.00	1.00	0.00	1.00	1671
13	1.00	0.20	1.00	0.20	0.91	1807
14	0.35	0.20	1.00	0.00	0.67	1852
15	1.00	0.54	1.00	0.00	1.00	2570
16	0.01	1.00	1.00	0.00	1.00	3463
17	0.01	1.00	1.00	0.00	1.00	3813

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0	2
2	0.1	4
3	0	15
4	1	16
5	0	882
6	0.2	1002
7	0.6	1070
8	0.3	1112
9	0.3	1113
10	0.5	1248
11	0.2	1288
12	0.2	1671
13	0.6	1807
14	0.4	1852
15	0.7	2570
16	0.7	3463
17	0.7	3813

HighThreatExperiment.xls

Totals

Experimental Outputs

Generated: 4/18/2006 3:44:52 PM

Experiment Number	Membership Function	Rule Set	Fuzzy Threat Level
1	General	General	0.4
2	General	Specific	0.8
3	Specific	General	0.43
4	Specific	Specific	0.7

Experiment 1 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	187	0	1	6.93	35	2570
2	2	1	1	2.00	0	3463
3	2	1	1	2.00	0	3813

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.80	0.80	1.00	0.80	0.11	2570
2	0.24	0.10	1.00	0.33	0.10	3463
3	0.24	0.10	1.00	0.33	0.10	3813

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	1	2570
2	0.1	3463
3	0.1	3813

Experiment 2 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	187	0	1	6.93	35	2570
2	2	1	1	2.00	0	3463
3	2	1	1	2.00	0	3813

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.80	0.80	1.00	0.80	0.11	2570
2	0.24	0.10	1.00	0.33	0.10	3463
3	0.24	0.10	1.00	0.33	0.10	3813

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	1	2570
2	0.7	3463
3	0.7	3813

Experiment 3 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	187	0	1	6.93	35	2570
2	2	1	1	2.00	0	3463
3	2	1	1	2.00	0	3813

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	1.00	0.54	1.00	0.00	1.00	2570
2	0.01	1.00	1.00	0.00	1.00	3463
3	0.01	1.00	1.00	0.00	1.00	3813

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0.5	2570
2	0.4	3463
3	0.4	3813

Experiment 4 – Application State

Crisp Threat Data

Number	Occurances	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	187	0	1	6.93	35	2570
2	2	1	1	2.00	0	3463
3	2	1	1	2.00	0	3813

Fuzzy Threat Data

Number	Occurances	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	1.00	0.54	1.00	0.00	1.00	2570
2	0.01	1.00	1.00	0.00	1.00	3463
3	0.01	1.00	1.00	0.00	1.00	3813

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0.7	2570
2	0.7	3463
3	0.7	3813

MedThreatExperiment.xls

Totals

Experimental Outputs

Generated: 4/18/2006 3:44:52 PM

Experiment Number	Membership Function	Rule Set	Fuzzy Threat Level
1	General	General	1
2	General	Specific	0.67
3	Specific	General	0.3
4	Specific	Specific	0.57

Experiment 1 – Application State

Crisp Threat Data

Number	Occurances	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	99	20	1	14.14	8	1070
2	102	20	1	14.57	36	1248
3	102	20	1	14.57	36	1807

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.80	0.80	1.00	0.80	0.21	1070
2	0.80	0.80	1.00	0.80	0.20	1248
3	0.80	0.80	1.00	0.80	0.20	1807

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	1	1070
2	1	1248
3	1	1807

Experiment 2 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	99	20	1	14.14	8	1070
2	102	20	1	14.57	36	1248
3	102	20	1	14.57	36	1807

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.80	0.80	1.00	0.80	0.21	1070
2	0.80	0.80	1.00	0.80	0.20	1248
3	0.80	0.80	1.00	0.80	0.20	1807

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0.6	1070
2	0.8	1248
3	0.6	1807

Experiment 3 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	99	20	1	14.14	8	1070
2	102	20	1	14.57	36	1248
3	102	20	1	14.57	36	1807

Fuzzy Threat Data

Number	Occurances	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.97	0.20	1.00	0.20	0.91	1070
2	1.00	0.20	1.00	0.20	0.91	1248
3	1.00	0.20	1.00	0.20	0.91	1807

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0.3	1070
2	0.3	1248
3	0.3	1807

Experiment 4 – Application State

Crisp Threat Data

Number	Occurances	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	99	20	1	14.14	8	1070
2	102	20	1	14.57	36	1248
3	102	20	1	14.57	36	1807

Fuzzy Threat Data

Number	Occurances	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.97	0.20	1.00	0.20	0.91	1070
2	1.00	0.20	1.00	0.20	0.91	1248
3	1.00	0.20	1.00	0.20	0.91	1807

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0.6	1070
2	0.5	1248
3	0.6	1807

LowThreatExperiment.xls

Totals

Experimental Outputs

Generated: 4/18/2006 3:44:52 PM

Experiment Number	Membership Function	Rule Set	Fuzzy Threat Level
1	General	General	0.5

2	General	Specific	0.24
3	Specific	General	0.4
4	Specific	Specific	0.22

Experiment 1 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	6	3	1	3.00	10	882
2	6	35	1	6.00	35	1112
3	102	20	1	14.57	36	1288
4	2	14	1	2.00	7	1671
5	61	6	1	1.27	20	1852

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.52	0.12	1.00	0.45	0.11	882
2	0.52	0.80	1.00	0.80	0.80	1112
3	0.80	0.80	1.00	0.80	0.20	1288
4	0.24	0.10	1.00	0.33	0.10	1671
5	0.80	0.80	1.00	0.25	0.40	1852

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0.1	882
2	0.7	1112
3	1	1288
4	0.1	1671
5	0.6	1852

Experiment 2 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	6	3	1	3.00	10	882
2	6	35	1	6.00	35	1112
3	102	20	1	14.57	36	1288
4	2	14	1	2.00	7	1671
5	61	6	1	1.27	20	1852

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.52	0.12	1.00	0.45	0.11	882
2	0.52	0.80	1.00	0.80	0.80	1112

3	0.80	0.80	1.00	0.80	0.20	1288
4	0.24	0.10	1.00	0.33	0.10	1671
5	0.80	0.80	1.00	0.25	0.40	1852

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0	882
2	0.6	1112
3	0.4	1288
4	0.1	1671
5	0.1	1852

Experiment 3 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	6	3	1	3.00	10	882
2	6	35	1	6.00	35	1112
3	102	20	1	14.57	36	1288
4	2	14	1	2.00	7	1671
5	61	6	1	1.27	20	1852

Fuzzy Threat Data

Number	Occurrences	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.03	0.99	1.00	0.00	1.00	882
2	0.03	0.20	1.00	0.00	0.20	1112
3	1.00	0.20	1.00	0.20	0.91	1288
4	0.01	1.00	1.00	0.00	1.00	1671
5	0.35	0.20	1.00	0.00	0.67	1852

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0.4	882
2	0.4	1112
3	0.3	1288
4	0.4	1671
5	0.5	1852

Experiment 4 – Application State

Crisp Threat Data

Number	Occurrences	Timespan (seconds)	Severity	Src IP frequency	Average Time Between (seconds)	Signature
1	6	3	1	3.00	10	882
2	6	35	1	6.00	35	1112

3	102	20	1	14.57	36	1288
4	2	14	1	2.00	7	1671
5	61	6	1	1.27	20	1852

Fuzzy Threat Data

Number	Occurances	Timespan	Severity	Src IP frequency	Average Time Between	Signature
1	0.03	0.99	1.00	0.00	1.00	882
2	0.03	0.20	1.00	0.00	0.20	1112
3	1.00	0.20	1.00	0.20	0.91	1288
4	0.01	1.00	1.00	0.00	1.00	1671
5	0.35	0.20	1.00	0.00	0.67	1852

Fuzzy Threat Level Data

Number	Threat Level	Signature
1	0	882
2	0.3	1112
3	0.2	1288
4	0.2	1671
5	0.4	1852

APPENDIX II – SOURCE CODE

ARF_Main.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Xml;
using MySQL.Data;
using MySQL.Data.MySqlClient;
using System.Data.SqlClient;
using System.Diagnostics;

using System.Net;
using System.Net.Sockets;
using System.Threading;

namespace arf
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class ARF_Main : System.Windows.Forms.Form
    {
        /// <summary>
        /// Constructor for the form object
        /// </summary>
        public ARF_Main()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();

            //
            //
        }

        /// <summary>
        /// Preliminary Setup for the Form Object
        /// </summary>
        private void Form1_Load(object sender, System.EventArgs e)
        {
            PopulateNetworkListBox();
            treeView1.ExpandAll();
        }

        /// <summary>
        /// Starts the threaded synchronization of all the
        /// sensors listed in hosts.xml
    }
}
```

```

    /// </summary>
    private void button1_Click(object sender, EventArgs e)
    {
Thread startSync = new Thread(new ThreadStart(StartHostConnect));
        startSync.Start();
    }

    /// <summary>
    ///     Starts the threaded synchronization process
    /// </summary>
    public void StartHostConnect()
    {
        //txtSyncStatus.Text = "";
        txtSyncStatus.Text = "Starting Sync " +
            System.DateTime.Now.ToString() + "\r\n";

        MySqlHostConnect();
        txtSyncStatus.Text += "Ending Sync " +
            System.DateTime.Now.ToString() + "\r\n";
    }

    /// <summary>
    ///     Sync a remote snort databases to local MySQL Server
    /// </summary>
    /// <param name="connection">Connection String For Remote
    ///     MySQL Server</param>
    /// <param name="SensorName">Alias of Remote Server to
    ///     reference in localdatabase</param>
    /// <returns>True if the host syncs</returns>
    private bool SsyncSingle(MySqlConnection connection, string
    SensorName)
    {
    // Find Timestamp of Last Alert Logged Locally for Sensor
    // Create transactions of the alerts from Remote Database
    // to be copied locally
        // --> events table
        // --> iphdr table
        // Start the transaction to copy remote records
        // locally
        // If the transaction succeeds, commit the
        // transaction and return true
        // if the transaction fails, rollback the
        // transaction and return false

        // Open remote connection
        // Create insert snapshot sql statement
        // execute statement
        // Save Resultset
        // close connection

        // Open local connection
        // insert saved result into event monitor
        // close Local connection
    }

```

```

// *****
//     FIND timestamp of last alert logged locally
// *****
MySQLConnection LocalConnection =
    this.MySqlConnection();
LocalConnection.Open();
string LastAlertTime = "SELECT MAX(timestamp) from
    event ";
LastAlertTime += "WHERE sensor_alias = '" +
    SensorName + "'";
string LastTime = "";
string Options = "";

MySQLCommand LastTimestampCommand = new
    MySQLCommand(LastAlertTime, LocalConnection);
MySQLDataReader LastTimestampReader;
LastTimestampReader =
    LastTimestampCommand.ExecuteReader();

MySQL.Data.Types.MySQLDateTime tempMySQLDate;
System.DateTime tempDateTime;
while(LastTimestampReader.Read())
{
    tempMySQLDate =
        LastTimestampReader.GetMySQLDateTime(0);

    if(!tempMySQLDate.IsNull)
    {
        tempDateTime =
            tempMySQLDate.GetDateTime();
        tempDateTime = tempDateTime.AddSeconds(-
            1.0);
        LastTime = tempDateTime.ToString("yyyy-
            MM-dd HH:mm:ss");
    }
    else
        LastTime = "";
}

if(LastTime != "")
    Options = " where timestamp >= '" + LastTime + "'";
else
    Options = "";

LocalConnection.Close();
LastTimestampReader.Close();

//*****
//     SELECT ALL REMOTE ALERTS TO BE COPIED
//*****

// -----
//EVENTS table
// -----
string mySelectQuery = "SELECT e.sid, e.cid,
    e.signature, e.timestamp, '";

```

```

mySelectQuery += SensorName + "' FROM event e " +
    Options;

string pkey = " WHERE 0=1 ";
ArrayList InsertStatements = new ArrayList();

MySQLCommand myCommand = new
    MySQLCommand(mySelectQuery,connection);

connection.Open();
MySQLDataReader myReader;

// make sure proper format in remote database
try
{
    myReader = myCommand.ExecuteReader();
}
catch
{
    return false;
}

while (myReader.Read())
{
    MySQL.Data.Types.MySqlDateTime d =
        myReader.GetMySqlDateTime(3);
    System.DateTime dt = d.GetDateTime();

    InsertStatements.Add("INSERT IGNORE INTO
arf_monitor.event (sid, cid, signature,
timestamp, sensor_alias) VALUES (" +
myReader.GetString(0) + ", " +
myReader.GetString(1) + ", " +
myReader.GetString(2) + ", '" +
dt.ToString("yyyy-MM-dd HH:mm:ss") + "', '" +
myReader.GetString(4) + "')");

    pkey += "OR (sid = " + myReader.GetString(0) +
" AND cid= " + myReader.GetString(1) + ")";
}
myReader.Close();
connection.Close();

// -----
//IPHDR table
// -----

string mySelectQuery2 = "SELECT i.sid, i.cid,
i.ip_src, i.ip_dst, i.ip_ver, i.ip_hlen, ";
mySelectQuery2 += i.ip_tos,i.ip_len, i.ip_id,
i.ip_flags,i.ip_off,i.ip_ttl,i.ip_proto, ";
mySelectQuery2 += i.ip_csum FROM home_alerts.iphdr i
" + pkey;

string t = "";

```

```

ArrayList InsertStatementsIP = new ArrayList();

MySQLCommand myCommand2 = new
    MySQLCommand(mySelectQuery2,connection);
connection.Open();
MySQLDataReader myReader2;

// make sure proper format in remote database
try
{
    myReader2 = myCommand2.ExecuteReader();
}
catch
{
    return false;
}
while (myReader2.Read())
{
    t = "INSERT IGNORE INTO arf_monitor.iphdr (sid,
    cid, ip_src, ip_dst, ip_ver, ip_hlen, ip_tos,
    ip_len, ip_id, ip_flags, ip_off, ip_ttl,
    ip_proto, ip_csum, sensor_alias) VALUES (";

    t+= myReader2.GetString(0) + ", " +
    myReader2.GetString(1) + ", " +
    myReader2.GetString(2) + ", " +
    myReader2.GetString(3) + ", " +
    myReader2.GetString(4) + ", " +
    myReader2.GetString(5) + ", " +
    myReader2.GetString(6) + ", ";

    t+= myReader2.GetString(7) + ", " +
    myReader2.GetString(8) + ", " +
    myReader2.GetString(9) + ", " +
    myReader2.GetString(10) + ", " +
    myReader2.GetString(11) + ", " +
    myReader2.GetString(12) + ", " +
    myReader2.GetString(13) + ", '" + SensorName +
    "'";

    t+= ")";
    InsertStatementsIP.Add(t);
}
myReader2.Close();
connection.Close();

// *****
//          Copy All new data to local machine
// *****

LocalConnection.Open();

MySQLCommand LocalCommand =
    LocalConnection.CreateCommand();
MySQLTransaction myTrans;

// Start a local transaction

```

```

myTrans = LocalConnection.BeginTransaction();
// Must assign both transaction object and connection
// to Command object for a pending local transaction
LocalCommand.Connection = LocalConnection;
LocalCommand.Transaction = myTrans;

try
{
    foreach(string s in InsertStatements)
    {
        LocalCommand.CommandText = s;
        LocalCommand.ExecuteNonQuery();
    }
    foreach(string s in InsertStatementsIP)
    {
        LocalCommand.CommandText = s;
        LocalCommand.ExecuteNonQuery();
    }
    myTrans.Commit();
    return true;
}
catch(Exception e)
{
    try
    {
        myTrans.Rollback();
        txtSyncStatus.Text += "transaction
            failed...rolled back\r\n";
    }
    catch (MySqlException ex)
    {
        if (myTrans.Connection != null)
        {
            txtSyncStatus.Text += "transaction
                failed...could not roll back\r\n";

            Debug.WriteLine("An exception of
                type " + ex.GetType() + " was
                encountered while attempting to
                roll back the transaction.");
        }
    }
}

Debug.WriteLine("An exception of type " +
    e.GetType() + " was encountered while
    inserting the data.");

Debug.WriteLine("No records were written to
    database.");
txtSyncStatus.Text += "An exception of type " +
    e.GetType() + " was encountered\r\n";

return false;
}
finally
{
    LocalConnection.Close();
}

```



```

    }
}

/// <summary>
///     Connects to the specified RevTreeNode and
///     synchronizes data if need be
/// </summary>
/// <param name="m">The RevTreeNode of the sensor to
///     synchronize</param>
private void ConnectSyncSingle(arf.RevTreeNode m)
{
    string conn = "SERVER=" + m.IP + "; DATABASE=" +
m.DatabaseName + "; UID=" + m.DatabaseUser + ";
PASSWORD=" + m.DatabasePassword + ";";

    MySqlConnection SingleConnection =
        ConnectSingleMySQL(conn);

    if(SingleConnection != null)
    {
        //txtSyncStatus.Text += "connected\r\n";
        // Sync the databases
        // change the icon

        if(SnycSingle(SingleConnection, m.Alias))
        {
            txtSyncStatus.Text += "success\r\n";
            m.ImageIndex = 0;
            m.SelectedImageIndex = 0;
        }
        else
        {
            txtSyncStatus.Text += "synchronization
                failure\r\n";
        }
    }
    else
    {
        txtSyncStatus.Text += "connection failure\r\n";
    }
}

/// <summary>
///     Connect and SNYC to all hosts listed for the
///     site in hosts.xml
/// </summary>
private void MySQLHostConnect()
{
    int i = 1;
    int count = 0;
    string message="";
    foreach(TreeNode n in treeView1.Nodes)
    {
        i = 1;
        count = n.Nodes.Count;
        foreach(arf.RevTreeNode m in n.Nodes)
        {

```

```

        message = "connecting to " + m.Alias + "
        @ " + n.Text + " " + i.ToString() + " of
        " + count.ToString() + "...";

        txtSyncStatus.Text += (message);
        ConnectSyncSingle(m);
        i++;
    }
}

/// <summary>
///     Connect to a remote database
/// </summary>
/// <param name="conn">The Connection string</param>
/// <returns>The mysql connection object or
///     null if connection is not available</returns>
private MySqlConnection ConnectSingleMySQL(string conn)
{
    try
    {
        MySqlConnection connection = new
            MySqlConnection(conn);
        connection.Open();
        connection.Close();
        return connection;
    }
    catch
    {
        return null;
    }
}

/// <summary>
///     Connects to the local mysql database
/// </summary>
/// <returns>The mysql connection object</returns>
private MySqlConnection MySQLConnect()
{
    string MyConString = "SERVER=localhost;
    DATABASE=arf_monitor; UID=root; PASSWORD=XXXX;";
    MySqlConnection connection = new
        MySqlConnection(MyConString);
    return connection;
}

/// <summary>
///     Opens a form to add a new host
/// </summary>
private void menuItem4_Click(object sender,
    System.EventArgs e)
{
    AddHost Child = new AddHost();
    Child.ShowDialog();
    PopulateNetworkListBox();
}

```

```

/// <summary>
///     Opens a form to add a new network
/// </summary>
private void menuItem5_Click(object sender,
    System.EventArgs e)
{
    AddNetwork Child = new AddNetwork();
    Child.ShowDialog();
}

/// <summary>
///     Creates the Tree View object from the hosts.xml
file, storing all pertinent
///     information about each host in the RevTreeNode
object
/// </summary>
private void PopulateNetworkListBox()
{
    System.Xml.XmlDocument xmldoc = new XmlDocument();

    xmldoc.Load(xmlfilename);

    treeView1.ImageList = this.NetworkImages;
    XmlNodeList xmlnetworks =
        xmldoc.GetElementsByTagName("network");
    XmlNodeList xmlhosts;

    System.Windows.Forms.TreeNode m;
    System.Windows.Forms.TreeNode [] n;
    System.Windows.Forms.TreeNode o;

    string alias;
    string ip;
    string dbn;
    string dbu;
    string dbp;

    for(int i=0;i<xmlnetworks.Count;i++)
    {
        xmlhosts = xmlnetworks[i].ChildNodes;
        n = new TreeNode[xmlhosts.Count];
        for(int j=0;j<xmlhosts.Count;j++)
        {
            // Each Host within a network

            alias =
            xmlhosts[j].ChildNodes[0].InnerText;
            ip =
            xmlhosts[j].ChildNodes[1].InnerText;
            dbn =
            xmlhosts[j].ChildNodes[2].InnerText;
            dbu =
            xmlhosts[j].ChildNodes[3].InnerText;
            dbp =
            xmlhosts[j].ChildNodes[4].InnerText;

```

```

        o = new
arf.RevTreeNode(alias, ip, dbn, dbu, dbp, 1, 1); // Image Default:Off(Red)
        n[j] = o;

    }
    m = new
TreeNode(xmlnetworks[i].Attributes[0].Value, 2, 2, n);
    treeView1.Nodes.Add(m);
    }
}

private void ChangeRawHostAlertTable(object sender,
System.Windows.Forms.TreeViewEventArgs e)
{
    string sql = "";

    sql += "SELECT e.sid, e.cid, e.timestamp, s.sig_sid
as signature, s.sig_name, inet_ntoa(i.ip_src) as
ip_src, inet_ntoa(i.ip_dst) as ip_dest ";

    sql += "FROM event e ";
    sql += "INNER JOIN signature s ";
    sql += "ON e.signature = s.sig_id inner join iphdr i
using (sid,cid) ";
    sql += "where e.sensor_alias = '" +
treeView1.SelectedNode.Text + "' ";

RawAlertDataset = new DataSet("event");
if(e.Node.GetType().ToString() == "arf.RevTreeNode")
{

    MySqlConnection conn = this.MySqlConnection();
    MySqlDataAdapter adapter = new
        MySqlDataAdapter();
    adapter.SelectCommand = new MySqlCommand(sql,
        conn);
    adapter.Fill(RawAlertDataset);
    //dataGridView1.DataSource = RawAlertDataset;

dataGridView1.SetDataBinding(RawAlertDataset, "Table");

    // Fuzzy Alert For Host
    lblSensor_Network.Text =
treeView1.SelectedNode.FullPath;
    double d =
arf.fuzzy.FuzzyEngine.HostThreatLevel(Raw
AlertDataset, lstXMLOutput);
    lblSensorThreat.Text = d.ToString("f");
}
}

/// <summary>
/// Gets the string representation of the signature
/// from the local MySQL Database
/// </summary>
/// <param name="sigId">integer ID (in string

```

```

format) of the signature in question</param>
/// <returns>string name of the signature</returns>
public static string GetSnortSignatureName(string sigId)
{
    string MyConString = "SERVER=localhost;
    DATABASE=arf_monitor; UID=root; PASSWORD=XXXX;";
    MySqlConnection LocalConnection = new
    MySqlConnection(MyConString);
    LocalConnection.Open();

    string GetSignatureName = "SELECT s.sig_name FROM
    signature s where sig_sid = " + sigId.ToString();

    string SigName = "";
    MySqlCommand GetSignatureCommand = new
    MySqlCommand(GetSignatureName, LocalConnection);
    MySqlDataReader GetSignatureReader;
    GetSignatureReader =
    GetSignatureCommand.ExecuteReader();

    while(GetSignatureReader.Read())
    {
        SigName = GetSignatureReader.GetString(0);
    }
    LocalConnection.Close();
    GetSignatureReader.Close();

    return SigName;
}

/// <summary>
/// Wrapper Function to call the Hex Encoding Functions
/// </summary>
/// <param name="hexString">Hexidecimal String holding a
    packet payload</param>
/// <returns>a byte array represneting the Hexidecimal
    string in bytes</returns>
public byte[] HexEncoder(string hexString)
{
    int discarded;

    byte[] byteArray =
        HexEncoding.GetBytes(hexString, out discarded);
    return byteArray;
}

/// <summary>
/// Get the IP address of this computer in a.b.c.d format
/// </summary>
/// <returns>This Computers IP Address</returns>
private string GetMyIp()
{
    string hostName = Dns.GetHostName();
    Console.WriteLine("Host Name = " + hostName);
    IPHostEntry local = Dns.GetHostByName(hostName);
    string ip = local.AddressList[0].ToString();
}

```

```

        return ip;
    }

    /// <summary>
    ///     Synchronizes a Single, selected sensor
    /// </summary>
    private void button6_Click(object sender, EventArgs e)
    {
        Thread startSync = new Thread(new
            ThreadStart(StartSingleHostConnect));
        startSync.Start();
    }

    /// <summary>
    ///     Start the thread to sync a single, selected host
    /// </summary>
    public void StartSingleHostConnect()
    {
        try
        {
            if(this.treeView1.SelectedNode.GetType().ToString()
                == "arf.RevTreeNode")
            {
                this.txtSyncStatus.Text = "Connecting to " +
                    this.treeView1.SelectedNode.Text + "...";
                ConnectSyncSingle((arf.RevTreeNode)this.treeView1.SelectedNode);
            }
            catch
            {
                MessageBox.Show("Select a sensor");
            }
        }

        /// <summary>
        ///     Runs the experiment as described in the testing section
        /// </summary>
        private void menuItem7_Click(object sender, EventArgs e)
        {
            arf.fuzzy.FuzzyExperiment.Experiment();
        }

        /// <summary>
        ///     Opens an external link to www.snort.org to view more
        ///     details of an alert
        /// </summary>
        private void ViewSignatureDetails(object sender,
            System.Windows.Forms.KeyPressEventArgs e)
        {
            string full =
                this.lstXMLOutput.SelectedItem.ToString();

```

```

        int startnum = full.IndexOf("|");

        string num = full.Substring(++startnum, (full.Length-
            1)-startnum);

        try
        {
            int i = int.Parse(num);
            System.Diagnostics.Process.Start(http://www.sno
            rt.org/pub-bin/signs.cgi?sid= + i.ToString() );
        }
        catch
        {
            MessageBox.Show("Signature not found, is it in
                this format |10|?");
        }
    }
}
}
}

```

AddHost.cs

```

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Xml;
using MySQL.Data;
using MySQL.Data.MySqlClient;
using System.IO;

namespace arf
{
    /// <summary>
    /// Summary description for AddHost.
    /// </summary>
    public class AddHost : System.Windows.Forms.Form
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;

        /// <summary>
        /// Constructor for the form object
        /// </summary>
        public AddHost()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
        }
    }
}

```

```

}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if(components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

/// <summary>
/// WRAPPER: Tests a connection to a remote database
/// </summary>
private void btnTest_Click(object sender, EventArgs e)
{
    TestDBConnection();
}

/// <summary>
/// Tests a connection to a remote database
/// </summary>
/// <returns>True if connection is valid, false
/// otherwise</returns>
private bool TestDBConnection()
{
    lblSuccess.BackColor = System.Drawing.Color.Gray;
    try
    {
        string MyConString = "SERVER=" +
            this.txtIP.Text + "; DATABASE=" +
            this.txtDBName.Text + "; UID=" +
            this.txtDBUser.Text + "; PASSWORD=" +
            this.txtDBPass.Text + ";";

        MySqlConnection connection = new
            MySqlConnection(MyConString);
        connection.Open();
        connection.Close();
        lblSuccess.BackColor =
            System.Drawing.Color.Green;
    }
    catch
    {
        lblSuccess.BackColor =
            System.Drawing.Color.Red;
        return false;
    }
    return true;
}

```



```

/// <summary>
///Adds some default values to the form for debugging
/// purposes
/// </summary>
private void btnDebug_Click(object sender, EventArgs e)
{
    this.txtAlias.Text = "Thesis Box";
    this.txtIP.Text = "localhost";
    this.txtDBName.Text = "arf_monitor";
    this.txtDBUser.Text = "root";
    this.txtDBPass.Text = "donkeylips";
}

/// <summary>
///      Sets up the form for data input
/// </summary>
private void AddHost_Load(object sender, EventArgs e)
{
    // Find Host ID's
    // Bind Names to DropDown cboNetwork

    System.Xml.XmlDocument hosts = new XmlDocument();
    hosts.Load(xmlfilename);

    System.Xml.XmlNodeList nl =
        hosts.GetElementsByTagName("network");

    cboNetwork.Items.Clear();
    foreach(System.Xml.XmlNode n in nl)
        cboNetwork.Items.Add(n.Attributes["name"].Value);

    cboNetwork.SelectedIndex = 0;
}

/// <summary>
///      Inserts entered data into hosts.xml
/// </summary>
private void InsertHostIntoXML()
{
    System.Xml.XmlDocument xmldoc = new XmlDocument();

    xmldoc.Load(xmlfilename);

    XmlElement newcatalogentry =
        xmldoc.CreateElement("host");

    XmlElement firstelement =
        xmldoc.CreateElement("alias");
    firstelement.InnerText = this.txtAlias.Text;
    newcatalogentry.AppendChild(firstelement);
    XmlElement secondelement =
        xmldoc.CreateElement("ip");
    secondelement.InnerText = this.txtIP.Text;
    newcatalogentry.AppendChild(secondelement);
    XmlElement thirdelement =
        xmldoc.CreateElement("dbname");

```

```

        thirdelement.InnerText = this.txtDBName.Text;
        newcatalogentry.AppendChild(thirdelement);
        XmlElement fourthelement =
            xmldoc.CreateElement("dbuser");
        fourthelement.InnerText = this.txtDBUser.Text;
        newcatalogentry.AppendChild(fourthelement);
        XmlElement fifthelement =
            xmldoc.CreateElement("dbpass");
        fifthelement.InnerText = this.txtDBPass.Text;
        newcatalogentry.AppendChild(fifthelement);

        System.Xml.XmlNode refNode = null;
        System.Xml.XmlNodeList nl =
            xmldoc.GetElementsByTagName("network");
        foreach(System.Xml.XmlNode n in nl)
        {
            if(n.Attributes["name"].Value ==
                (string)this.cboNetwork.SelectedItem)
            {
                refNode = n;
                break;
            }
        }

        refNode.AppendChild(newcatalogentry);
        FileStream fsxml = new
            FileStream(this.xmlfilename, FileMode.Truncate,
            FileAccess.Write, FileShare.ReadWrite);

        xmldoc.Save(fsxml);
        fsxml.Close();
    }

    /// <summary>
    ///     Accepts the data entered into the form
    /// </summary>
    private void btnAccept_Click(object sender, EventArgs e)
    {
        DialogResult result;
        if(!TestDBConnection())
            result = MessageBox.Show("This Connection is
            not working, Insert Anyway?", "Database
            connection unavailable",
            System.Windows.Forms.MessageBoxButtons.YesNo,
            System.Windows.Forms.MessageBoxIcon.Warning);
        else
            result = DialogResult.Yes;

        if(result == DialogResult.Yes)
            InsertHostIntoXML();

        this.Close();
    }
}
}
}

```

AddNetwork.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Xml;
using System.IO;

namespace arf
{
    /// <summary>
    /// Summary description for AddNetwork.
    /// </summary>
    public class AddNetwork : System.Windows.Forms.Form
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;

        /// <summary>
        /// Constructor for the form object
        /// </summary>
        public AddNetwork()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if(components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        /// <summary>
        /// WRAPPER: Adds the network to hosts.xml
        /// </summary>
        private void btnAddNetwork_Click(object sender,EventArgs e)
        {
            AddNetworkToXML();
            // Close the form
            this.Close();
        }
    }
}
```

```

    /// <summary>
    ///     Actually adds the name to the hosts.xml file
    /// </summary>
    public void AddNetworkToXML()
    {
        if(this.txtNetworkName.Text.Trim() != "" &&
            this.txtNetworkName.Text != null)
        {
            // Load Existing hosts.xml file
            System.Xml.XmlDocument networks = new
                XmlDocument();
            networks.Load(xmlfilename);

            //Create list of nodes with tag name "network"
            System.Xml.XmlNodeList nl =
                networks.GetElementsByTagName("network");

            // Create New XML Element
            XmlElement newcatalogentry =
                networks.CreateElement("network");
            XmlAttribute newcatalogattr =
                networks.CreateAttribute("name");
            newcatalogattr.Value =
                this.txtNetworkName.Text;

            newcatalogentry.SetAttributeNode(newcatalogattr);

            // Add XML element at end of node list
            networks.DocumentElement.InsertAfter(newcatalog
                entry, networks.DocumentElement.LastChild);

            // Save the new XML file to disk
            FileStream fsxml = new
                FileStream(this.xmlfilename, FileMode.Truncate,
                    FileAccess.Write, FileShare.ReadWrite);

            networks.Save(fsxml);
            fsxml.Close();
        }
    }
}

```

RevTreeNode.cs

```

using System;
using System.Windows.Forms;

namespace arf
{
    /// <summary>
    ///     A custom TreeNode object that holds all applicable
    ///     information about intrusion detection // sensors
    /// </summary>

```

```

public class RevTreeNode : TreeNode
{
    private string alias;
    private string ip;
    private string dbname;
    private string dbuser;
    private string dbpass;

    /// <summary>
    ///     A name to call the sensor
    /// </summary>
    public string Alias
    {
        get{return alias;}
    }

    /// <summary>
    ///     The IP address of the sensor in a.b.c.d format
    /// </summary>
    public string IP
    {
        get{return ip;}
    }

    /// <summary>
    ///Name of the MySQL database sensor information resides
    /// </summary>
    public string DatabaseName
    {
        get{return dbname;}
    }

    /// <summary>
    ///The restricted privelage user used to copy sensor data
    /// </summary>
    public string DatabaseUser
    {
        get{return dbuser;}
    }

    /// <summary>
    ///     Cleartext password for the user listed above
    /// </summary>
    public string DatabasePassword
    {
        get{return dbpass;}
    }

    /// <summary>
    ///     Creates a Tree Node Object
    /// </summary>
    /// <param name="a">Alias of the Node (shown in
    ///     tree)</param>
    /// <param name="ipaddr">IP address of the node</param>
    /// <param name="dn">MySQL database name</param>

```

```

    /// <param name="du">MySQL database user</param>
    /// <param name="dp">MySQL database password</param>
    /// <param name="imageIndex">Index of the Image to use to
        represent the node</param>
    /// <param name="SelectedImageIndex">Which Image is
        selected in the list</param>
    public RevTreeNode(string a, string ipaddr, string dn,
        string du, string dp, int imageIndex, int
        selectedImageIndex)
    {
        base.Text = a;
        alias = a;
        ip = ipaddr;
        dbname = dn;
        dbuser = du;
        dbpass = dp;
        base.ImageIndex = imageIndex;
        base.SelectedImageIndex = selectedImageIndex;
    }
}
}

```

Interfaces

```

namespace arf.fuzzy.interfaces
{
    public interface IFuzzification
    {
        FuzzyThreatCollection MakeFuzzy(CrispThreatCollection i);
        FuzzyThreat MakeFuzzy(CrispThreat c);
        FuzzyThreatCollection GetFuzzyThreats();
    }

    public interface IIInferenceEngine
    {
        FuzzyThreatLevelCollection
            FuzzyInference(FuzzyThreatCollection i);
        FuzzyThreatLevel FuzzyInference(FuzzyThreat f);
        FuzzyThreatLevelCollection GetFuzzyThreatLevel();
    }

    public interface IDefuzzifier
    {
        double Defuzzify(FuzzyThreatLevelCollection i);
        double GetCrispOutput();
    }

    public interface IRule
    {
        double FindFuzzyThreatLevel(FuzzyThreat f);
    }

    public interface IMath
    {

```

```

        double solve (double d);
    }
}

```

Objects

```

namespace arf.fuzzy.objects
{
    #region Crisp Threat Objects (CrispThreat, CrispThreatCollection)

    public struct CrispThreat
    {
        #region Properties

        private int occurances;
        private System.TimeSpan timespan;
        private int severity_multiplier;
        private double src_ip_frequency;
        private System.TimeSpan avg_time_between;
        private int signature;

        #endregion

        #region Property Accessors

        /// <summary>
        ///     Number of times an alert has been generated
        /// </summary>
        public int Occurances
        {
            get{return occurances;}
            set{occurances = value;}
        }

        /// <summary>
        ///     The Timespan between the first and last alert
        /// </summary>
        public System.TimeSpan Timespan
        {
            get{return timespan;}
            set{ timespan=value;}
        }

        /// <summary>
        ///     A future-use severity multiplier
        /// </summary>
        public int Severity_Multiplier
        {
            get{return severity_multiplier;}
            set{ severity_multiplier=value;}
        }

        /// <summary>
        ///     The frequency of IPs that are generating this alert
        /// </summary>
        public double Src_Ip_Frequency
        {

```

```

        get{return src_ip_frequency;}
        set{src_ip_frequency=value;}
    }

    /// <summary>
    ///     The Average timespan between alerts
    /// </summary>
    public System.TimeSpan Avg_Time_Between
    {
        get{return avg_time_between;}
        set{avg_time_between=value;}
    }

    /// <summary>
    ///     The SNORT signature of the alert
    /// </summary>
    public int Signature
    {
        get{return signature;}
        set{signature=value;}
    }
#endregion

#region Constructors

    /// <summary>
    ///     Creates a CrispThreat Object from scratch
    /// </summary>
    public CrispThreat(int num_occurrences, System.TimeSpan
    alert_avg_timespan, int severity_mult, System.TimeSpan
    average_time_between_attacks, int alert_signature, double
    source_ip_frequency)
    {
        occurrences = num_occurrences ;
        timespan = alert_avg_timespan;
        severity_multiplier = severity_mult;
        src_ip_frequency = source_ip_frequency;
        avg_time_between = average_time_between_attacks;
        signature = alert_signature;
    }

    /// <summary>
    ///     Creates a copy of a crisp threat object
    /// </summary>
    /// <param name="t">The CrispThreat Object to Copy</param>
    public CrispThreat(CrispThreat t)
    {
        occurrences = t.occurrences;
        timespan = t.timespan;
        severity_multiplier = t.severity_multiplier;
        src_ip_frequency = t.src_ip_frequency;
        avg_time_between = t.avg_time_between;
        signature = t.signature;
    }
#endregion
}

```



```

public class CrispThreatCollection :
    System.Collections.CollectionBase
{
    /// <summary>
    ///     Add a CrispThreat to a Collection
    /// </summary>
    /// <param name="c"></param>
    public virtual void Add(CrispThreat c)
    {
        this.List.Add(c);
    }

    /// <summary>
    ///     Index a CrispThreat within a Collection
    /// </summary>
    public virtual CrispThreat this[int Index]
    {
        get
        {return (CrispThreat)this.List[Index];}
    }
}

#endregion

#region Fuzzy Threat Objects (Fuzzy Threat,
    FuzzyThreatCollection)

    /// <summary>
    /// Same fields as CrispThreat Object, except values are doubles
    /// </summary>
    public struct FuzzyThreat
    {
        #region Properties

        // *****
        // HOLDS Y-VALUE OF MEMBERSHIP FUNCTION
        // *****
        private double fuzz_occurrences;
        private double fuzz_timespan;
        private double fuzz_severity_multiplier;
        private double fuzz_src_ip_frequency;
        private double fuzz_avg_time_between;

        // *****
        // HOLDS X-VALUE OF MEMBERSHIP FUNCTION
        // *****
        private CrispThreat crisp;

        private int signature;

        #endregion

        #region Property Accessors

```

```

public CrispThreat crispThreat
{
    get{return crisp;}
}

public double Occurances
{
    get{return fuzz_occurrences;}
    set{fuzz_occurrences = value;}
}
public double Timespan
{
    get{return fuzz_timespan;}
    set{ fuzz_timespan=value;}
}
public double Severity_Multiplier
{
    get{return fuzz_severity_multiplier;}
    set{ fuzz_severity_multiplier=value;}
}

public double Src_Ip_Frequency
{
    get{return fuzz_src_ip_frequency;}
    set{fuzz_src_ip_frequency=value;}
}
public double Avg_Time_Between
{
    get{return fuzz_avg_time_between;}
    set{fuzz_avg_time_between=value;}
}
public int Signature
{
    get{return signature;}
    set{signature=value;}
}
#endregion

#region Constructors
public FuzzyThreat(CrispThreat c,double
fuzz_num_occurrences, double fuzz_alert_avg_timespan, double
fuzz_severity_mult, double
fuzz_average_time_between_attacks, int alert_signature,
double fuzz_source_ip_frequency)
{
    fuzz_occurrences = fuzz_num_occurrences ;
    fuzz_timespan = fuzz_alert_avg_timespan;
    fuzz_severity_multiplier = fuzz_severity_mult;
    fuzz_src_ip_frequency = fuzz_source_ip_frequency;
    fuzz_avg_time_between =
        fuzz_average_time_between_attacks;
    signature = alert_signature;

    crisp = c;
}
public FuzzyThreat(FuzzyThreat f)
{

```

```

        fuzz_occurrences = f.fuzz_occurrences;
        fuzz_timespan = f.fuzz_timespan;
        fuzz_severity_multiplier =
            f.fuzz_severity_multiplier;
        fuzz_avg_time_between = f.fuzz_avg_time_between;
        signature = f.signature;
        fuzz_src_ip_frequency = f.fuzz_src_ip_frequency;
        crisp = f.crisp;
    }
#endregion

}

public class FuzzyThreatCollection :
    System.Collections.CollectionBase
{
    public virtual void Add(FuzzyThreat c)
    {
        this.List.Add(c);
    }

    public virtual FuzzyThreat this[int Index]
    {
        get
        {return (FuzzyThreat)this.List[Index];}
    }
}

#endregion

#region Fuzzy Threat Level Objects (FuzzyThreatLevel
    FuzzyThreatLevelCollection)

/// <summary>
/// All fuzzy metrics combined with a defuzzification method
/// form a single value
/// </summary>
public struct FuzzyThreatLevel
{
    #region Properties

    double fuzzy_threat_level;
    int signature;

    #endregion

    #region Property Accessors
    public double Fuzzy_Threat_Level
    {
        get{return fuzzy_threat_level;}
        set{fuzzy_threat_level = value;}
    }
    public int Signature
    {
        get{return signature;}
    }
}

```

```

        set{signature = value;}
    }
#endregion

#region Constructors

public FuzzyThreatLevel(double threat_level, int sig)
{
    fuzzy_threat_level = threat_level;
    signature = sig;
}
public FuzzyThreatLevel(FuzzyThreatLevel f)
{
    fuzzy_threat_level = f.fuzzy_threat_level;
    signature = f.signature;
}

#endregion

}

public class FuzzyThreatLevelCollection :
    System.Collections.CollectionBase
{
    public virtual void Add(FuzzyThreatLevel c)
    {
        this.List.Add(c);
    }

    public virtual FuzzyThreatLevel this[int Index]
    {
        get
        {return (FuzzyThreatLevel)this.List[Index];}
    }
#region Methods

public System.Xml.XmlDocument CreateXML()
{
    XmlElement signatureNode;
    XmlElement fuzzyNode;
    XmlText fuzzyText;

    XmlDocument xmlDoc = new XmlDocument();
    XmlDeclaration xmlDeclaration =
        xmlDoc.CreateXmlDeclaration("1.0", "utf-8", null);

    // Create the root element
    XmlElement rootNode = xmlDoc.CreateElement("host");
    rootNode.SetAttribute("name", "Insert Host Name
Here");

rootNode.SetAttribute("time", System.DateTime.Now.ToString());
    xmlDoc.InsertBefore(xmlDeclaration,
        xmlDoc.DocumentElement);
    xmlDoc.AppendChild(rootNode);
}
}

```

```

        foreach(FuzzyThreatLevel ftl in this.InnerList)
        {
            // Create the <signature> element
            signatureNode =
                xmlDoc.CreateElement("signature");

            // Set attribute name and value
            signatureNode.SetAttribute("sig",
                ftl.Signature.ToString());

            xmlDoc.DocumentElement.AppendChild(signatureNode);

            // Create the Fuzzy Value Node
            fuzzyNode =
                xmlDoc.CreateElement("fuzzy_value");
            fuzzyText =
                xmlDoc.CreateTextNode(ftl.Fuzzy_Threat_Level.ToString());
            signatureNode.AppendChild(fuzzyNode);
            fuzzyNode.AppendChild(fuzzyText);
        }

        // Save to the XML file
        xmlDoc.Save( "a.xml" );
        return xmlDoc;
    }

    #endregion
}

#endregion
}

```

Membership Functions

Specific Membership Functions

```

using System;
using arf.fuzzy.objects;
using arf.fuzzy.interfaces;
using arf.math;

namespace arf.fuzzy.membershipfunctions
{
    /// <summary>
    /// Specific Set of Membership Functions for the application
    /// </summary>
    public class FuzzificationMembershipFunction : IFuzzification
    {
        #region Properties

        public FuzzyThreatCollection fuzzyThreats = new
            FuzzyThreatCollection();

        #endregion
    }
}

```

```

#region Constructors

public
FuzzificationMembershipFunction(CrispThreatCollection i)
{
    fuzzyThreats = MakeFuzzy(i);
}
#endregion

#region IFuzzification Members

public FuzzyThreatCollection GetFuzzyThreats()
{
    return fuzzyThreats;
}

public FuzzyThreatCollection
MakeFuzzy(CrispThreatCollection i)
{
    FuzzyThreat f;
    foreach(CrispThreat t in i)
    {
        f = new FuzzyThreat(MakeFuzzy(t));
        fuzzyThreats.Add(f);
    }
    return fuzzyThreats;
}

public FuzzyThreat MakeFuzzy(CrispThreat c)
{
    double fuzz_occurrences =
        this.MembershipOccurrences(c.Occurrences);
    double fuzz_timespan =
        this.MembershipTimespan(c.Timespan);
    double fuzz_severity_multiplier =
        this.MembershipSeverity(c.Severity_Multiplier);
    double fuzz_avg_time_between =
        this.MembershipTimeBetween(c.Avg_Time_Between);
    double fuzz_source_ip_frequency =
        this.MembershipSrcIPFreq(c.Src_Ip_Frequency);
    int signature = c.Signature;

    return new
    FuzzyThreat(c, fuzz_occurrences, fuzz_timespan, fuzz_seve
    rity_multiplier, fuzz_avg_time_between, signature, fuzz_
    source_ip_frequency);
}
#endregion

#region Helper (Private) Methods
private double MembershipOccurrences(int occurrences)
{
    double [,] points = new double[11,2];

    points[0,0] = 0;    points[0,1] = .0;
}

```

```

    points[1,0] = 10;  points[1,1] = .05;
    points[2,0] = 20;  points[2,1] = .1;
    points[3,0] = 25;  points[3,1] = .14;
    points[4,0] = 30;  points[4,1] = .2;
    points[5,0] = 38;  points[5,1] = .3;
    points[6,0] = 43;  points[6,1] = .32;
    points[7,0] = 50;  points[7,1] = .35;
    points[8,0] = 80;  points[8,1] = .35;
    points[9,0] = 100; points[9,1] = 1;
    points[10,0] = double.PositiveInfinity ;
    points[10,1] = 1;

    IMath f = new SetOfPointsFunction(points);
    double fuzz = f.solve(occurances);
    return fuzz;
}
private double MembershipTimespan(TimeSpan t)
{
    // OLD: [[0,1],[172800,.2],[172800*2,1],[infinity,1]]
    // NEW: [[0,1],[172800*2,.2],[infinity,.2]]

    double [,] points = new double[3,2];
    points[0,0] = 0;  points[0,1] = 1;
    points[1,0] = 172800*2;  points[1,1] = .2;
    points[2,0] = double.PositiveInfinity;  points[2,1] =
        .2;

    IMath f = new SetOfPointsFunction(points);
    double fuzz = f.solve(t.TotalSeconds);
    return fuzz;
}
private double MembershipSeverity(int s)
{
    return 1;
}
private double MembershipTimeBetween(TimeSpan t)
{
    double [,] points = new double[6,2];
    points[0,0] = 0;  points[0,1] = 1;
    points[1,0] = 3600;  points[1,1] = 1;
    points[2,0] = 86400;  points[2,1] = .2;
    points[3,0] = 172800;  points[3,1] = .2;
    points[4,0] = 432000;  points[4,1] = 1;
    points[5,0] = double.PositiveInfinity;
    points[5,1] = 1;

    IMath f = new SetOfPointsFunction(points);
    double fuzz = f.solve(t.TotalSeconds);
    return fuzz;
}
private double MembershipSrcIPFreq(double ip_fre)
{
    double [,] points = new double[11,2];
    // MAPLE - set of points

```

```

// [0,0],[10,0],[10,.2],[25,.2],[25,.5],[80,.5],
// [80,.8],[100,.8],[100,1],[150,1],[infinity,1]

points[0,0] = 0;   points[0,1] = 0;
points[1,0] = 10;  points[1,1] = 0;
points[2,0] = 10;  points[2,1] = .2;
points[3,0] = 25;  points[3,1] = .2;
points[4,0] = 25;  points[4,1] = .5;
points[5,0] = 80;  points[5,1] = .5;
points[6,0] = 80;  points[6,1] = .8;
points[7,0] = 100; points[7,1] = .8;
points[8,0] = 100; points[8,1] = 1;
points[9,0] = 150; points[9,1] = 1;
points[10,0] = double.PositiveInfinity ;
points[10,1] = 1;

IMath f = new SetOfPointsFunction(points);
double fuzz = f.solve(ip_fre);
return fuzz;
}

/// <summary>
///     For testing purposes only
/// </summary>
/// <returns>a random number in the set [0,.25,.5,.75,1]</returns>
private double R()
{
    System.Random r = new Random();
    int temp = r.Next(1,5);
    if(temp == 1)
        return 0.0;
    else if(temp == 2)
        return 0.25;
    else if(temp == 3)
        return 0.5;
    else if(temp == 4)
        return 0.75;
    return 1.0;
}
#endregion
}
}
}

```

General Membership Functions

```

using System;
using arf.fuzzy.objects;
using arf.fuzzy.interfaces;
using arf.math;

namespace arf.fuzzy.membershipfunctions
{
    /// <summary>
    ///     General Set of Membership Functions for the
    Application

```



```

/// </summary>
public class GeneralMembership : IFuzzification
{
    #region Properties

    public FuzzyThreatCollection fuzzyThreats = new
        FuzzyThreatCollection();

    #endregion

    #region Constructors

    public GeneralMembership(CrispThreatCollection i)
    {
        fuzzyThreats = MakeFuzzy(i);
    }

    #endregion

    #region IFuzzification Members

    public FuzzyThreatCollection
        MakeFuzzy(CrispThreatCollection i)
    {
        FuzzyThreat f;
        foreach(CrispThreat t in i)
        {
            f = new FuzzyThreat(MakeFuzzy(t));
            fuzzyThreats.Add(f);
        }
        return fuzzyThreats;
    }

    public FuzzyThreat MakeFuzzy(CrispThreat c)
    {
        double fuzz_occurrences =
            this.MembershipOccurrences(c.Occurrences);
        double fuzz_timespan =
            this.MembershipTimespan(c.Timespan);
        double fuzz_severity_multiplier =
            this.MembershipSeverity(c.Severity_Multiplier);
        double fuzz_avg_time_between =
            this.MembershipTimeBetween(c.Avg_Time_Between);
        double fuzz_source_ip_frequency =
            this.MembershipSrcIPFreq(c.Src_Ip_Frequency);
        int signature = c.Signature;

        return new
            FuzzyThreat(c, fuzz_occurrences, fuzz_timespan, fuzz_seve
            rity_multiplier, fuzz_avg_time_between, signature, fuzz_
            source_ip_frequency);
    }

    public FuzzyThreatCollection GetFuzzyThreats()
    {
        return fuzzyThreats;
    }
}

```

```

}

#endregion

#region Helper (Private) Methods

private double MembershipOccurrences(int occurrences)
{
    double [,] points = new double[3,2];

    points[0,0] = 0;    points[0,1] = .1;
    points[1,0] = 10;  points[1,1] = .8;
    points[2,0] = double.PositiveInfinity;
    points[2,1] = .8;

    IMath f = new SetOfPointsFunction(points);
    double fuzz = f.solve(occurrences);
    return fuzz;
}

private double MembershipTimespan(TimeSpan t)
{
    double [,] points = new double[3,2];

    points[0,0] = 0;    points[0,1] = .1;
    points[1,0] = 172800; points[1,1] = .8;
    points[2,0] = double.PositiveInfinity;
    points[2,1] = .8;

    IMath f = new SetOfPointsFunction(points);
    double fuzz = f.solve(t.TotalSeconds);
    return fuzz;
}

private double MembershipSeverity(int s)
{
    return 1;
}

private double MembershipTimeBetween(TimeSpan t)
{
    double [,] points = new double[3,2];

    points[0,0] = 0;    points[0,1] = .1;
    points[1,0] = 172800/2; points[1,1] = .8;
    points[2,0] = double.PositiveInfinity;
    points[2,1] = .8;

    IMath f = new SetOfPointsFunction(points);
    double fuzz = f.solve(t.TotalSeconds);
    return fuzz;
}

private double MembershipSrcIPFreq(double ip_fre)
{
    double [,] points = new double[3,2];

    points[0,0] = 0;    points[0,1] = .1;
    points[1,0] = 6;    points[1,1] = .8;
    points[2,0] = double.PositiveInfinity;
    points[2,1] = .8;
}

```

```

        IMath f = new SetOfPointsFunction(points);
        double fuzz = f.solve(ip_fre);
        return fuzz;
    }

    #endregion
}
}

```

Rules

```

public class RuleBase
{
    protected const double startlow = .2;
    protected const double startmed = .5;
    protected const double starthigh = .7;

    protected double CheckBounds(double a)
    {
        if( a > 1)
            return 1;
        else if( a < 0)
            return 0;
        else
            return a;
    }

    protected bool IsNone(double FuzzyValue)
    {
        if(FuzzyValue == 0)
            return true;
        return false;
    }

    protected bool IsLow(double FuzzyValue)
    {
        if(FuzzyValue <= .33 && FuzzyValue > 0)
            return true;
        return false;
    }

    protected bool IsMed(double FuzzyValue)
    {
        if(FuzzyValue <= .66 && FuzzyValue > .33)
            return true;
        return false;
    }

    protected bool IsHigh(double FuzzyValue)
    {
        if(FuzzyValue <= .99 && FuzzyValue > .66)
            return true;
        return false;
    }

    protected bool IsDefinate(double FuzzyValue)
    {
        if(FuzzyValue == 1.0)
            return true;
    }
}

```

```

        return false;
    }
    protected bool IsMedHigh(double FuzzyValue)
    {
        if(this.IsHigh(FuzzyValue) || this.IsMed(FuzzyValue))
            return true;
        return false;
    }
    protected bool IsLowMed(double FuzzyValue)
    {
        if(this.IsLow(FuzzyValue) || this.IsMed(FuzzyValue))
            return true;
        return false;
    }
}

```

Specific Rules

```

using System;
using arf.fuzzy.interfaces;
using arf.fuzzy.objects;

namespace arf.fuzzy.rules
{
    public class Rules : RuleBase, IRule
    {
        public Rules()
        {
        }

        #region IRule Members

        public double FindFuzzyThreatLevel(FuzzyThreat f)
        {
            double ftl = 0.0;
            #region Signature Switch
            switch(f.Signature)
            {
                case 1671: // 'WEB-MISC /home/www access'
                    ftl = this.WEB_MISC_ACCESS(f);
                    break;
                case 15:
                    ftl =
                        this.HTTP_INSPECT_OVERSIZE_REQUEST(f);
                    break;
                case 4:
                    ftl =
                        this.HTTP_INPECT_BB_UNICODE_ENCODING(f);
                    break;
                case 1112:
                    ftl =
                        this.WEB_MISC_DIRECTORY_TRAVERSAL(f);
                    break;
                case 1113:
                    ftl =
                        this.WEB_MISC_DIRECTORY_TRAVERSAL(f);
            }
            #endregion
        }
    }
}

```

```

        break;
    case 882:
        ft1 = this.WEB_CGI_CALENDAR_ACCESS(f);
        break;
    case 16:
        ft1 =
this.HTTP_INSPECT_OVERSIZE_CHUNK_ENCODING(f);
        break;
    case 2:
        ft1 =
this.HTTP_INSPECT_DOUBLE_DECODING_ATTACK(f);
        break;
    case 1852:
        ft1 = this.WEB_MISC_ROBOTS_ACCESS(f);
        break;
    case 3813:
        ft1 = this.WEB_CGI_COMMAND_EXECUTION(f);
        break;
    case 3463:
        ft1 = this.WEB_CGI_AWSTATS_ACCESS(f);
        break;
    case 1070:
        ft1 = this.WEB_MISC_WEBDAV_SEARCH(f);
        break;
    case 1807:
        ft1 =
this.WEB_MISC_CHUNKED_ENCODING_TRANSFER(f);
        break;
    case 1248:
        ft1 = this.WEB_FRONTPAGE_DLL_ACCESS(f);
        break;
    case 1288:
        ft1 =
this.WEB_FRONTPAGE_VTI_BIN_ACCESS(f);
        break;
    case 1002:
        ft1 = this.WEB_IIS_CMD_ACCESS(f);
        break;
    case 2570:
        ft1 =
this.WEB_MISC_INVALID_HTTP_VERSION_STRING(f);
        break;
    default:
        ft1 = -1.0;
        break;
}
#endregion
return ft1;
}
#endregion

#region RulesBySignature
/*
    f.Occurances;
    f.Timespan;
    f.Avg_Time_Between; (timespan / occurances)
    f.Src_Ip_Frequency; (# IPs / occurances)

```

```

        */

//'WEB-MISC /home/www access'
// http://www.snort.org/pub-bin/signs.cgi?sid=1671
// Low threat
private double WEB_MISC_ACCESS(FuzzyThreat f)
{
    double threat = startlow;

    if(this.IsHigh(f.Src_Ip_Frequency) &&
        this.IsLow(f.Avg_Time_Between))
        threat += .5;
    if(this.IsHigh(f.Src_Ip_Frequency) &&
        !this.IsLow(f.Avg_Time_Between))
        threat += .2;

    if(this.IsHigh(f.Timespan))
        threat -= .2;

    if(this.IsLow(f.Timespan))
        threat -= .1;

    threat = CheckBounds(threat);

    return threat;
}

//'(http_inspect) OVERSIZE REQUEST-URI DIRECTORY' (15)
private double HTTP_INSPECT_OVERSIZE_REQUEST(FuzzyThreat f)
{
    double threat = startlow;

    if(this.IsHigh(f.Occurances))
        threat += threat*2;
    if(this.IsLow(f.Timespan))
        threat -= .2;
    threat = this.CheckBounds(threat);
    return threat;
}

//'(http_inspect) BARE BYTE UNICODE ENCODING' (4)
private double HTTP_INPECT_BB_UNICODE_ENCODING(FuzzyThreat
f)
{
    double threat = startlow;

    if(this.IsHigh(f.Occurances))
        threat += .2;
    if(this.IsLowMed(f.Src_Ip_Frequency))
        threat -= .1;

    threat = this.CheckBounds(threat);
    return threat;
}

//'WEB-MISC http directory traversal' (1112)

```

```

// http://www.snort.org/pub-bin/sigs.cgi?sid=1112
// Low Threat
private double WEB_MISC_DIRECTORY_TRAVERSAL(FuzzyThreat f)
{
    double threat = startlow;

    if(this.IsHigh(f.Occurances))
        threat += .3;
    if(this.IsLowMed(f.Occurances))
        threat += .1;

    if(this.IsLow(f.Src_Ip_Frequency))
        threat -= .2;
    if(this.IsHigh(f.Src_Ip_Frequency) &&
        this.IsHigh(f.Avg_Time_Between))
        threat += .3;

    threat = CheckBounds(threat);
    return threat;
}

//'WEB-CGI calendar access'
// http://www.snort.org/pub-bin/sigs.cgi?sid=882
// Low / No Threat
private double WEB_CGI_CALENDAR_ACCESS(FuzzyThreat f)
{
    // This application does not exist on our system
    return 0.0;
}

//'(http_inspect) OVERSIZE_CHUNK_ENCODING'
// Unknown Threat
private double
HTTP_INSPECT_OVERSIZE_CHUNK_ENCODING(FuzzyThreat f)
{
    return 1.0;
}

//'(http_inspect) DOUBLE_DECODING_ATTACK'
// Unknown Threat
private double
HTTP_INSPECT_DOUBLE_DECODING_ATTACK(FuzzyThreat f)
{
    double threat = startlow;
    if(this.IsHigh(f.Occurances))
        threat += .2;
    if(this.IsLow(f.Timespan))
        threat -= .2;
    threat = this.CheckBounds(threat);
    return threat;
}

//'WEB-MISC robots.txt access'
// http://www.snort.org/pub-bin/sigs.cgi?sid=1852
// Low Threat

```

```

private double WEB_MISC_ROBOTS_ACCESS(FuzzyThreat f)
{
    double threat = startlow;
    if(this.IsHigh(f.Avg_Time_Between))
        threat += .2;
    if(this.IsLowMed(f.Src_Ip_Frequency))
        threat -= .1;
    return threat;
}

//'WE-CGI awstats.pl configdir command execution attempt' (3813)
// http://www.nessus.org/plugins/index.php?view=single&id=16189
// High Threat
private double WEB_CGI_COMMAND_EXECUTION(FuzzyThreat f)
{
    double threat = starthigh;
    if(this.IsHigh(f.Avg_Time_Between))
        threat += .1;
    if(this.IsMedHigh(f.Timespan))
        threat += .2;
    if(this.IsMedHigh(f.Occurances))
        threat += .3;

    threat = CheckBounds(threat);
    return threat;
}

//'WEB-CGI awstats access' (3463)
//
http://www.nessus.org/plugins/index.php?view=single&id=16456
// High Threat
private double WEB_CGI_AWSTATS_ACCESS(FuzzyThreat f)
{
    double threat = starthigh;
    if(this.IsHigh(f.Avg_Time_Between))
        threat += .1;
    if(this.IsMedHigh(f.Timespan))
        threat += .2;
    if(this.IsMedHigh(f.Occurances))
        threat += .3;

    threat = CheckBounds(threat);
    return threat;
}

//'WEB-MISC WebDAV search access' (1070)
// http://www.snort.org/pub-bin/sigs.cgi?sid=1070
// Medium Threat
private double WEB_MISC_WEBDAV_SEARCH(FuzzyThreat f)
{
    double threat = startmed;
    // Possible DOS for unpatched (out of the box) WebDAV

```

Service


```

        if(this.IsHigh(f.Avg_Time_Between))
            threat += .1;
        if(this.IsHigh(f.Src_Ip_Frequency))
            threat += .1;
        if(this.IsLowMed(f.Occurances))
            threat -= .2;

        threat = this.CheckBounds(threat);
        return threat;
    }

    //'WEB-MISC Chunked-Encoding transfer attempt' (1807)
    // http://www.snort.org/pub-bin/sigs.cgi?sid=1807
    // Medium Threat
    private double
WEB_MISC_CHUNKED_ENCODING_TRANSFER(FuzzyThreat f)
    {
        double threat = startmed;
        if(this.IsMedHigh(f.Avg_Time_Between))
            threat += .1;
        if(this.IsHigh(f.Src_Ip_Frequency))
            threat += .1;

        threat = this.CheckBounds(threat);
        return threat;
    }

    //'WEB-FRONTPAGE rad fp30reg.dll access' (1248)
    // http://www.snort.org/pub-bin/sigs.cgi?sid=1248
    // Medium Threat
    private double WEB_FRONTPAGE_DLL_ACCESS(FuzzyThreat f)
    {
        double threat = startmed;

        if(this.IsHigh(f.Occurances))
            threat += .1;
        if(this.IsLowMed(f.Occurances))
            threat -= .1;
        if(this.IsHigh(f.Src_Ip_Frequency))
            threat += .2;

        threat = this.CheckBounds(threat);
        return threat;
    }

    //'WEB-FRONTPAGE /_vti_bin/ access' (1288)
    //
    http://www.nessus.org/plugins/index.php?view=single&id=11032
    // Low threat
    private double WEB_FRONTPAGE_VTI_BIN_ACCESS(FuzzyThreat f)
    {
        double threat = startlow;
        if(this.IsHigh(f.Src_Ip_Frequency))
            threat += .2;
    }

```

```

        threat = this.CheckBounds(threat);
        return threat;
    }

    //'WEB-IIS cmd.exe access' (1002)
    // http://www.snort.org/pub-bin/sigs.cgi?sid=1002
    private double WEB_IIS_CMD_ACCESS(FuzzyThreat f)
    {
        double threat = startlow;
        if(this.IsHigh(f.Src_Ip_Frequency))
            threat += .2;
        if(this.IsLowMed(f.Timespan))
            threat -= .1;
        threat = this.CheckBounds(threat);
        return threat;
    }

    //'WEB-MISC Invalid HTTP Version String' (2570)
    // http://www.snort.org/pub-bin/sigs.cgi?sid=2570
    // High Threat
    private double
WEB_MISC_INVALID_HTTP_VERSION_STRING(FuzzyThreat f)
    {
        double threat = starthigh;
        if(this.IsMedHigh(f.Src_Ip_Frequency))
            threat += .2;
        if(this.IsHigh(f.Timespan))
            threat += .1;
        threat = this.CheckBounds(threat);
        return threat;
    }

    #endregion
}
}

```

General Rules

```

using System;
using arf.fuzzy.interfaces;
using arf.fuzzy.objects;

namespace arf.fuzzy.rules
{
    /// <summary>
    ///     General Set of Rules for the application
    /// </summary>
    public class GeneralRules : RuleBase, IRule
    {
        public GeneralRules()
        {
            //
            // TODO: Add constructor logic here
            //
        }
    }
}

```

```

#region IRule Members

public double FindFuzzyThreatLevel(FuzzyThreat f)
{
    double ftl = 0.0;
    ftl = GRule(f);
    return ftl;
}

#endregion

private double GRule(FuzzyThreat f)
{
    double threat = startmed;

    // If anything is a high threat, elevate the threat level
    if(this.IsHigh(f.Occurances) ||
        this.IsHigh(f.Src_Ip_Frequency) ||
        this.IsHigh(f.Timespan) ||
        this.IsHigh(f.Avg_Time_Between))

        threat += .1;

    // If a lot of different IPs are doing it in a low
    amount of time...its probably new and // needs to be
    looked at
    if(this.IsHigh(f.Src_Ip_Frequency) &&
        this.IsLow(f.Avg_Time_Between))
        threat += .5;

    // If a lot of different IPs are doing over a large
    timespan the threat should be
    // elevated
    if(this.IsHigh(f.Src_Ip_Frequency) &&
        this.IsMedHigh(f.Avg_Time_Between))
        threat += .2;

    // If not a lot of people are doing it over not a lot
    // of time, decrement the threat level
    if(this.IsLowMed(f.Src_Ip_Frequency) &&
        this.IsLow(f.Timespan))
        threat -= .3;

    // If not happening a lot, decrement the threat level
    if(this.IsLowMed(f.Occurances))
        threat -= .1;

    threat = CheckBounds(threat);
    return threat;
}
}
}

```

FuzzyEngine.cs

```
using System;
using System.Data;
using System.Collections;
using arf.fuzzy.objects;
using arf.fuzzy.interfaces;
using arf.fuzzy.rules;
using arf.fuzzy.membershipfunctions;
using System.Xml;

using MySQL.Data;
using MySQL.Data.MySqlClient;

namespace arf.fuzzy
{
    /// <summary>
    ///The main Wrapper for fuzzy Logic Operations...a starting point
    /// </summary>
    public class FuzzyEngine
    {
        /// <summary>
        /// Finds a Threat Level for a host (sensor)
        /// </summary>
        /// <returns>The threat level for the sensor</returns>
        public static double HostThreatLevel(DataSet ds,
            System.Windows.Forms.ListBox l)
        {
            XmlDocument xmlDoc = new XmlDocument();

            SnortAdapter s = new SnortAdapter(ds);
            CrispThreatCollection c1 = s.CrispThreats;

            IFuzzification f = new
            FuzzificationMembershipFunction(c1);
            FuzzyThreatCollection f1 = f.GetFuzzyThreats();

            IRule r = new Rules();
            IIInferenceEngine i = new RuleBasedInference(f1,r);
            FuzzyThreatLevelCollection f2 =
                i.GetFuzzyThreatLevel();

            xmlDoc = f2.CreateXML();
            FormatLog(xmlDoc, l);

            IDefuzzifier MeanOfMaximum = new MOMDefuzzify(f2);
            double o = MeanOfMaximum.GetCrispOutput();
            return o;
        }

        /// <summary>
        /// Basic Logging Function to show specific steps of the
        /// application to the GUI
        /// </summary>
        /// <param name="xmlDoc"></param>
        /// <param name="l"></param>
    }
}
```

```

        /// <returns></returns>
        private static string FormatLog(XmlDocument xmldoc,
System.Windows.Forms.ListBox l)
        {

            l.Items.Clear();
            XmlNodeList xmlnode =
                xmldoc.GetElementsByTagName("signature");
            string s = "";

            for(int i=0;i<xmlnode.Count;i++)
            {
                XmlAttributeCollection xmlattrc =
                    xmlnode[i].Attributes;
                l.Items.Add(xmlnode[i].FirstChild.Name + ": "+
                    xmlnode[i].FirstChild.InnerText + " -> " +
                    arf.ARF_Main.GetSnortSignatureName(xmlattrc[0].
                    Value) + " |" + xmlattrc[0].Value + "|");
            }

            return s;
        }
    }
}

```

SnortAdapter.cs

```

using System;
using System.Data;
using System.Collections;
using arf;
using arf.fuzzy.objects;

namespace arf.fuzzy
{
    /// <summary>
    /// Creates a CrispThreat object based on a common snort event
    /// log dataset
    /// </summary>
    public class SnortAdapter
    {
        private ArrayList dataRows = new ArrayList();
        // An arraylist of (DataRow []) objects
        private DataSet hostAlerts;
        private CrispThreatCollection crispThreats = new
            CrispThreatCollection();

        public CrispThreatCollection CrispThreats
        {
            get{return crispThreats;}
        }

        public SnortAdapter(DataSet HostAlerts)
        {
            ArrayList SortedDataRows = new ArrayList();

```

```

hostAlerts = HostAlerts;
DataRow [] AllDataRows;
DataRow [] SortedSignatureRows;
int [] signatures;
int UniqueSignatures = 0;

DataTable d = CreateEventTable();
DataTable temp;
d = HostAlerts.Tables["Table"];

AllDataRows = d.Select("0=0","signature desc");
temp = SelectDistinct("event",d,"signature");
UniqueSignatures = temp.Rows.Count;
signatures = new int[UniqueSignatures];

for(int i = 0; i < temp.Rows.Count; i++)
{
    signatures[i] =
        Parse(temp.Rows[i].ItemArray[0].ToString());
}

for(int i = 0; i < UniqueSignatures; i++)
{
    SortedSignatureRows = d.Select("signature = " +
        signatures[i].ToString());
    SortedDataRows.Add(SortedSignatureRows);
}

CrispThreat c;
for(int i = 0; i < SortedDataRows.Count; i++)
{
    c = new CrispThreat(CalculateThreat((DataRow
        [])SortedDataRows[i]));
    crispThreats.Add(c);
}
}

```

```

private CrispThreat CalculateThreat(DataRow [] d)

```

```

    int occurrences = d.Length;
    System.TimeSpan t;
    System.TimeSpan a;

    System.DateTime first = new DateTime(1);
    System.DateTime last = new DateTime(1);
    bool IsFirstIteration = true;
    double avg = 0.0;
    string signature = "";
    ArrayList IPs = new ArrayList();
    int UniqueIP = 0;
    double ip_per_alert = 0.0;

    System.DateTime current;
    foreach(DataRow r in d)

```

```

    {
        current =
            DateTime.Parse(r.ItemArray[2].ToString());
        if(IsFirstIteration)
        {
            first = current;
            last = current;
            signature = r.ItemArray[3].ToString();
        }
        if(current < first && !IsFirstIteration)
            first = current;
        if(current > last && !IsFirstIteration)
            last = current;

        IsFirstIteration = false;

        // Find unique IP's
        if(!IPs.Contains(r.ItemArray[5]))
            IPs.Add(r.ItemArray[5]);
    }
    t = last.Subtract(first);
    avg = t.TotalSeconds/(double)occurrences;
    a = new TimeSpan(0,0,(int)avg);
    UniqueIP = IPs.Count;
    ip_per_alert = (double)occurrences / (double)UniqueIP;

    CrispThreat c = new
    CrispThreat(occurrences,t,SeverityFinder(signature),a,
    int.Parse(signature),ip_per_alert);
    return c;
}

#region Private (Helper Functions)
//copied from
http://support.microsoft.com/default.aspx?scid=kb;en-us;326176
public DataTable SelectDistinct(string TableName, DataTable
SourceTable, string FieldName)
{
    DataTable dt = new DataTable(TableName);
    dt.Columns.Add(FieldName,
SourceTable.Columns[FieldName].DataType);

    object LastValue = null;
    foreach (DataRow dr in SourceTable.Select("",
FieldName))
    {
        if ( LastValue == null ||
!(ColumnEqual(LastValue, dr[FieldName])) )
        {
            LastValue = dr[FieldName];
            dt.Rows.Add(new object[] {LastValue});
        }
    }
    if (hostAlerts != null)
        hostAlerts.Tables.Add(dt);
}

```

```

        return dt;
    }

    //copied from
http://support.microsoft.com/default.aspx?scid=kb;en-us;326176
    private bool ColumnEqual(object A, object B)
    {
        // Compares two values to see if they are equal. Also
        compares DBNull.Value.
        // Note: If your DataTable contains object fields,
        then you must extend this
        // function to handle them in a meaningful way if you
        intend to group on them.

        if ( A == DBNull.Value && B == DBNull.Value ) //
        both are DBNull.Value
            return true;
        if ( A == DBNull.Value || B == DBNull.Value ) //
        only one is DBNull.Value
            return false;
        return ( A.Equals(B) ); // value type standard
        comparison
    }

    private DataTable CreateEventTable()
    {
        DataTable dt = new DataTable("event");

        DataColumn dc = new
        DataColumn("sid", System.Type.GetType("System.Int32"));
        dc.AllowDBNull = false;
        dt.Columns.Add(dc);

        dc = new
        DataColumn("cid", System.Type.GetType("System.Int32"));
        dc.AllowDBNull = false;
        dt.Columns.Add(dc);

        dc = new
        DataColumn("signature", System.Type.GetType("System.Int32"));
        dc.AllowDBNull = false;
        dt.Columns.Add(dc);

        dc = new
        DataColumn("timestamp", System.Type.GetType("System.DateTime"));
        dc.AllowDBNull = false;
        dt.Columns.Add(dc);

        dc = new
        DataColumn("sensor_alias", System.Type.GetType("System.String"));
        dc.AllowDBNull = false;
        dt.Columns.Add(dc);

        dc = new
        DataColumn("ip_src", System.Type.GetType("System.String"));
        dc.AllowDBNull = false;
    }

```



```

        dt.Columns.Add(dc);

        dc = new
DataColumn("ip_dest",System.Type.GetType("System.String"));
        dc.AllowDBNull = false;
        dt.Columns.Add(dc);

        return dt;
    }
    private int SeverityFinder(string signature)
    {
        return 1;
    }
    #endregion
}
}

```

RulesBasedInference.cs

```

using System;
using arf.fuzzy.objects;
using arf.fuzzy.interfaces;
using arf.fuzzy.rules;

namespace arf.fuzzy
{
    /// <summary>
    ///     Maps the ruleset to a signature
    /// </summary>
    public class RuleBasedInference : IInferenceEngine
    {
        #region Properties

        private FuzzyThreatLevelCollection fuzzyThreatLevels = new
            FuzzyThreatLevelCollection();
        private IRule r;
        #endregion

        public RuleBasedInference(FuzzyThreatCollection f, IRule
            RuleSet)
        {
            r = RuleSet;
            fuzzyThreatLevels = FuzzyInference(f);
        }

        #region IInferenceEngine Members

        public FuzzyThreatLevelCollection
            FuzzyInference(FuzzyThreatCollection i)
        {
            FuzzyThreatLevel ftl;
            foreach(FuzzyThreat t in i)
            {
                ftl = new FuzzyThreatLevel(FuzzyInference(t));
                fuzzyThreatLevels.Add(ftl);
            }
        }
    }
}

```

```

        }
        return fuzzyThreatLevels;
    }

    public FuzzyThreatLevel FuzzyInference(FuzzyThreat f)
    {
        FuzzyThreatLevel ftl = new
        FuzzyThreatLevel(r.FindFuzzyThreatLevel(f),f.Signature);
        return ftl;
    }

    public FuzzyThreatLevelCollection GetFuzzyThreatLevel()
    {
        return fuzzyThreatLevels;
    }

    #endregion
}
}

```

MOMDefuzzify.cs

```

using System;
using arf.fuzzy.objects;
using arf.fuzzy.interfaces;

namespace arf.fuzzy
{
    /// <summary>
    ///     Mean of Maximum Defuzzifier
    /// </summary>
    public class MOMDefuzzify : IDefuzzifier
    {
        double output;

        public MOMDefuzzify(FuzzyThreatLevelCollection i)
        {
            output = Defuzzify(i);
        }

        #region IDefuzzifier Members

        public double GetCrispOutput()
        {
            return output;
        }

        public double Defuzzify(FuzzyThreatLevelCollection i)
        {
            int count = 0;
            double dValue= 0.0;
            foreach(FuzzyThreatLevel t in i)
            {
                count++;
                dValue += t.Fuzzy_Threat_Level;
            }
        }
    }
}

```

```

        }
        return dValue/(double)count;
    }

    #endregion
}

```

MaxDefuzzify.cs

```

using System;
using arf.fuzzy;
using arf.fuzzy.objects;
using arf.fuzzy.interfaces;

namespace arf.fuzzy
{
    /// <summary>
    ///     Returns the Maximum Fuzzy value
    /// </summary>
    public class MaxDefuzzify : IDefuzzifier
    {
        double output;
        public MaxDefuzzify(FuzzyThreatLevelCollection i)
        {
            output = Defuzzify(i);
        }
        #region IDefuzzifier Members

        public double Defuzzify(FuzzyThreatLevelCollection i)
        {
            double largest = 0.0;

            foreach(FuzzyThreatLevel t in i)
            {
                if(t.Fuzzy_Threat_Level > largest)
                    largest = t.Fuzzy_Threat_Level;
            }
            return largest;
        }

        public double GetCrispOutput()
        {
            return output;
        }

        #endregion
    }
}

```

Vita

Jeremy Gray

419 Winkler Ave, Louisville, KY 40208

Education

University of Louisville

Louisville, Kentucky

- B.S. Computer Engineering and Computer Science, May 2005
-

Experience

Aircraft Maintenance and Engineering Applications Support

2002-2003

UPS

- Primary responsibilities included design and development of enterprise web applications
- Enhanced and tested airline applications using Visual Basic
- Emphasis on software lifecycle documentation and capability maturity model (CMM)

Teaching Assistant – Design of Computer Algorithms

May 2005 – December 2005

University of Louisville

- A graduate level course covering the engineering design of efficient computer algorithms
- Preparing course materials and examples covering a range of topics in computer algorithms

Deputy Sheriff

October 2004 – June 2006 (present)

Jefferson County Sheriff's Office

- Perform security assignments in the Jefferson County Hall of Justice and at various community functions, such as church picnics, Kentucky Derby events, and charity fundraisers
- Exposure to a wide array of training courses, including communication skills and cultural diversity
- Deployed for seven days to Louisiana as part of Hurricane Katrina relief effort in September 2005

Casino Games Dealer

September 2003 – June 2006 (present)

Casino Entertainment

- Dealing craps, blackjack, and poker games, in a Monte Carlo style
- Providing an entertaining environment while teaching gaming fundamentals