Fuzzy intrusion detection

by

John Edward Dickerson

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Major Professor: James Davis

Iowa State University

Ames, Iowa

2001

Graduate College

Iowa State University

This is to certify that the Master's thesis of

John Edward Dickerson

has met the thesis requirements of Iowa State University

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Jim Davis, for his broad understanding, positive encouragement, and helpful suggestions during this research. It was his Introduction to Information Security class that inspired me to pursue graduate research in computer security in the first place. I would also like to give thanks to Dr. Doug Jacobson for his vast knowledge of computer security and networking. I greatly valued his technical insight on a wide range of topics . I was also very pleased to have had Dr. Richard Hindman on my committee because he has helped me to see that computer security and productivity should never be in opposition.

Additional thanks must go to Chris McCoy, the manager of Engineering Computing Support Services at Iowa State University, who graciously allowed me time off from work when I needed it and was always willing to listen patiently while I ranted about computer security.

Finally, I must give loving thanks to my wife Julie Dickerson and to Stormie. Julie, for her valuable insight into fuzzy systems and her unfailing support, and Stormie for never letting me forget that home is still the thing that matters most.

ABSTRACT

This thesis examines the application of fuzzy systems to the problem of network intrusion detection. Historically, there have been two primary methods of performing intrusion detection: misuse detection and anomaly detection. In misuse detection, a database of attack signatures is maintained that match known intrusion activity. While misuse detection systems are very effective, they require constant updates to the signature database to remain effective or to detect distinctly new attacks. Anomaly detection systems attempt to discover suspicious behavior by comparing system activity against past usage profiles. In this research, network activity is collected and usage profiles established for a variety of metrics. A network data gathering and data analysis tool was developed to create the metrics from the network stream. Great care is given to identifying the metrics that are most suitable for detecting intrusion activity. Visual data mining techniques are used to assess which metrics are most effective at detecting different types of attacks. The research confirms that data aggregation and data reduction play crucial roles in the formation of the metrics. Once the proper metrics are identified, fuzzy rules are constructed for detecting attacks in several categories. The attack categories are selected to match the different phases that intruders frequently use when attacking a system. A suite of attacks tools is assembled to test the fuzzy rules. The research shows that fuzzy rules applied to good metrics can provide an effective means of detecting a wide variety of network intrusion activity. This research is being used as a proof of concept for the development of system known as the Fuzzy Intrusion Recognition Engine (FIRE).

INTRODUCTION

Security of computer systems connected to the Internet has become an increasingly important subject as more and more aspects of our daily lives involve online computing. The widely publicized distributed denial-of-service attacks against major Internet sites in February 2000 showed that our networks and systems are woefully vulnerable to attacks. In the aftermath of those attacks, several prominent security experts identified intrusion detection as one of several technologies that can contribute to overall improvements in system security [CERT, 2000]. This thesis develops a novel technique for intrusion detection and tests its performance in real-world scenarios.

Intrusion detection is the process of identifying computer and network activity that can lead to the compromise of a security policy [Amoroso, 1999]. Intrusion detection can be performed on a single computer system by inspecting system logs and audit trails, or one can place the intrusion detection system on a telecommunications network where it can observe the messages sent between machines. In general, there are three methodologies for performing intrusion detection: misuse detection, intrusion traps, and anomaly detection.

In misuse detection, comparing input data against known intrusion signatures discovers suspicious activity. The data stream, either system logs or network data, is compared against a database of known attacks. When a pattern in the input stream matches a signature in the database, the system sends an alert to the security administrator. Many commercial intrusion detection systems are misuse detection systems. While misuse detection systems are quite effective, they also require continuous updates to the security databases to remain effective. In general, they are unable to recognize an intrusion unless a pattern exists for it in the signature database.

Intrusion traps are disguised or modified resources that issue an alarm when an intruder uses them. Though highly effective at detecting some types of computer break-ins, they can require a high degree of customization to prepare and are highly system specific [Ranum, 1999]. What security administrators would prefer to have is a detection system that can adapt to new types of attacks without extensive modification to the underlying threat database or operating environment.

Anomaly-based intrusion detection seeks to find suspicious activity by comparing the current system activity against typical patterns of use. The essence of anomaly detection is that an intrusion will leave a trail of unusual activity that, if one knows what to look for, will reveal the malicious activity. The goal of most anomaly detection systems is to be able to detect new attacks, even when that attack has never been observed before [Amoroso, 1999]. Of the three types of intrusion detection, anomaly detection has received the greatest amount of research attention since it has the potential to solve a much larger type of intrusion detection problems and adapts well to new types of attacks. [Crosbie, 1995].

It is important to note that, by itself, intrusion detection cannot make a system more secure [Crosbie and Spafford, 1995]. Rather, intrusion detection can provide notification to a security administrator about the occurrence of an attack, help reveal system vulnerabilities, and identify what areas may need increased security protections

[Denning, 1987]. A good intrusion detection system can also help a system administrator become better informed about the state of the networks and systems they administer.

Prior researchers in the field have explored numerous methods for anomaly detection [Frank, 1994]. One technique that has not been widely explored is the use of fuzzy systems. Fuzzy systems use fuzzy logic to express rules in human terms that are then transformed into machine-executable rules. An expert analyst can use the often-imprecise language of experience and intuition when describing the rules relating system inputs to outputs. Fuzzy systems can be combined with machine learning algorithms to help quantify the rule structures. Fuzzy systems have been applied successfully to a variety of pattern recognition and control system applications including speed regulation of high-speed trains and signal processing applications [Kosko, 1993]. We believe that the inherent flexibility of fuzzy systems make them an attractive technology for the problem of intrusion detection.

One technology that has received some investigation for intrusion detection is the area of data mining [Lee and Wenke, 1997]. Data mining is the process of looking for hidden patterns and relationships in data [Schearing, 1997]. This technology has been applied successfully to a wide variety of data analysis problems, including health-care cost predictions, and automotive sales forecasting. Visual data mining is the technique of displaying multivariate input data sets in order to visualize the relationships that exist in data [Buja and Cook, 1996]. We believe that visual data mining can provide valuable insight into which data parameters are the most effective measures of data anomalies.

This research addressed by this thesis serves as a proof-of-concept for a system being developed at Iowa State University called the Fuzzy Intrusion Recognition Engine (FIRE). This goal of the FIRE project is to develop an easy –to-deploy anomaly detection system that uses fuzzy systems in the intrusion assessment engine.

BACKGROUND AND MOTIVATION

Related Research

The Denning Model

Research into intrusion detection is not new. Investigators have been exploring the field in earnest for several years [Frank, 1994]. In [Denning, 1987], Dorothy Denning made an invaluable contribution to the field of intrusion detection by defining the fundamental framework for describing all intrusion detection systems. Denning developed a model that shows that nearly all anomaly detection systems can be characterized as rule-based pattern matching systems. A depiction of this model is shown in Figure 1. In this model, audit trails of network packet data and host-based system logs produce all of the input data. The audit trail data are compared against usage activity profiles for the data feed. The profiles are derived from statistical metrics or models of the data over time. A security operator develops pattern-matching rules that define which input conditions should be considered anomalies and also how they should be interpreted.

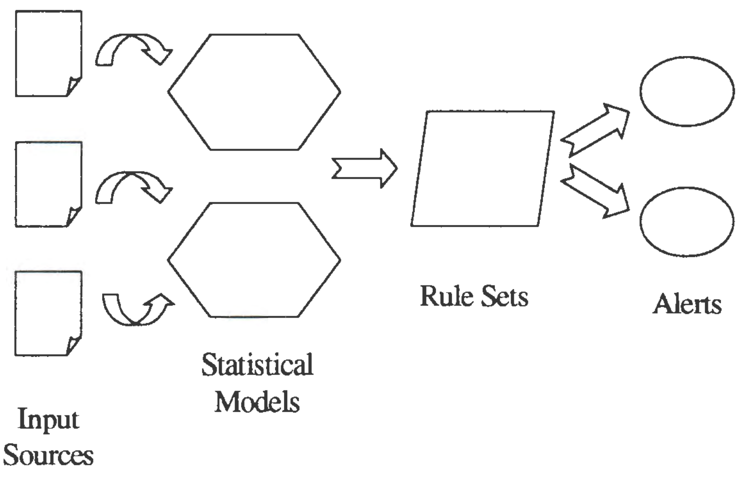

Figure 1. The Denning Model for Intrusion Detection.

This basic model exists in nearly all intrusion detection systems and provides a neat reference point for comparing and contrasting how different intrusion detection systems (IDSs) function.

Activity profiles are particularly important in Denning's model. A profile consists a metric (e.g. number of logins, length of TCP connections) and some type of statistical model of that metric. Denning describes three general metrics: Event Counter, Interval Timer, and Resource Measure. The event counter is simply a count of a particular data element such as the number of network packets received during a time interval. The interval timer is a measure of the time between particular events, such as user login and logout events. Denning describes the resource measure as an element that resides "at a level below the audit records". An example of a resource measure might be that a probability that a particular user logs in on a computer from midnight to 6 AM . This suggests that most resource measures but must be derived from data in the audit logs. Extracting resource measures from the available data is a form of data reduction since resource measures typically summarize other audit events.

Another important component of Denning's model is the statistical models used to construct the activity profiles. Denning identifies several types of statistical models:

- Operation Model – A fixed limit is used as the threshold for determining what abnormal means.
- Mean and Standard Deviation Model – Input metrics are treated as random variables that have statistical mean and standard deviation. The mean and standard deviation are computed from input data over time using the equations:

$$sum = x_1 + \ldots + x_n$$

$$sumsquares = x_1^2 + \ldots + x_n^2$$

$$mean = sum / n$$

$$stdev = \sqrt{\left(\frac{sumsquares}{(n-1)} - mean^2\right)} \quad \textbf{(1)}$$

- Multivariate Model – Similar to the Mean and Standard Deviation Model, except that it is based on correlations of two or more metrics. This method is particularly useful if better discriminating power can be obtained from combinations of related measures.
- Markov Process Model – Regards the input events as state variables and sequences of events as state transition processes. This technique is useful for identifying intrusions characterized by a particular chain of events. An intrusion would be detected when the probability of a certain event occurring in the sequence was below a predefined threshold.
- Time series models – Considers the events in the context of when the event occurred. A new observation is considered to be abnormal if the likelihood of it occurring at a particular time is low.

While other statistical models can be used, they will generally be derived from the above models. A survey of intrusion detection systems indicates that while there are many different ways to implement a detection system, a large majority of the systems share one or more aspects of the Denning model for intrusion detection and vary predominantly only in the techniques used. The Denning model is a surprisingly accurate description of almost all anomaly-based ID systems.

Other Models

There are models other than the Denning Model that have been used to describe frameworks for intrusion detection. One of these models is the computer immunology approach presented by Forrest, et al. [Forrest, 1997]. Their concept draws a strong analogy between the task of detecting hostile intrusions in a computer system and how a complex organism detects and isolates antigens in the bloodstream. After an initial training period, the system determines which system inputs/outputs constitute an image of “self” in the organism/computer system. System inputs that fail to match this concept of self are designated as intrusions. Kim and Bentley extend this research by investigating negative selection and pattern niching to reduce the amount of training necessary to evolve the system [Kim, 1998]. The computer immunology approach is interesting because it implies a built-in response to the intrusion. Though this approach has yet to be implemented in the real world, it presents an intriguing vision for intrusion detection in the future.

Another approach that has attracted interest is the application of genetic algorithms to intrusion detection. Crosbie and Spafford describe an intrusion detection system that uses simple agents that focus on a particular aspect of the system [Crosbie and Spafford, 1995]. The agents learn patterns that indicate an intrusion and adapt their rules to new intrusions over time. Like the computer immunology approach, they can be directed to attack the intruder by shutting it down or limiting its access to system resources. As we shall see, the use of autonomous agents as a fundamental data collection and processing component of an IDS is a popular idea.

IDS Architectures

It is important to note that Denning's model does not define an architecture for how the working components of the model must be implemented. However, Amoroso suggests that all intrusion detection systems are composed of several key elements as shown in Figure 2 [Amoroso, 1999]. Amoroso also defines several terms that are useful in describing the typical IDS system:

- Target System – The system being monitored. This could be a network, a computer, or a service.
- Feed – The source of the data (network, system logs, etc.)
- Processing – The execution of algorithms designed to detect malicious activity.
- Knowledge Base – A store of information about attack signatures or usage profiles that are used to identify malicious activity.
- Storage – The IDS must store short term or long term information in order to identify attacks.
- Alarms/Directives – Responses sent by the IDS to interested parties or another processing component when a security event occurs. Ex: Email, beepers, "raise the drawbridge" messages.
- GUI/Operator Interface – An easy-to-use way for the security administrator to interact with the IDS.

- Communications Infrastructure – A consistent framework for how distributed parts of the IDS communicate.
- Multiple intrusion detection systems – A site may use more than one IDS. Interoperability of the tools may be a consideration.
- Network Management – Some larger networks use network management software for monitoring, reporting, and administration. This can be used in conjunction with the IDS.

Several researchers have designed ID systems as a collection of agents that work independently to gather and analyze input data. The most noted of these approaches is

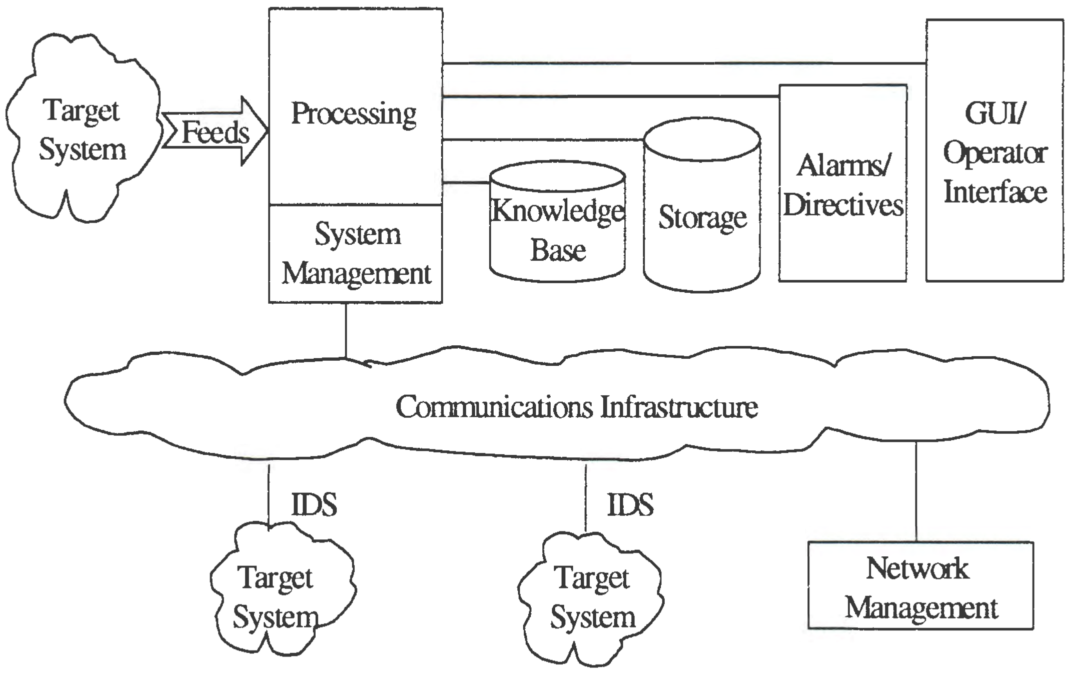

Figure 2. All intrusion detection systems contain these key elements. From [Amoroso, 1999].

the EMERALD system developed at SRI [Porras and Neumann 1997]. EMERALD is a complex misuse and anomaly detection system that uses specialized agents to detect specific intrusions. The system is essentially hierarchical so that lower level monitoring agents can share their results with a management layer that coordinates the actions of the entire system. The agents use a combination of statistical methods and experience databases to identify malicious activity. Also, the agents can monitor network-based and host-based data feeds. The management layer shares the results from the various agents with other agents while simultaneously constructing an overall view of the system state.

The Autonomous Agents for Intrusion Detection (AAFID) system is also built from specialized agents that perform intrusion detection while sharing the results with other coordination elements in a hierarchical arrangement [Zamboni, 1998]. The AAFID architecture is designed to be fault tolerant in the presence of failures and to be lightweight. That is, AAFID uses readily available tools for network and host monitoring that can be run concurrently on the existing computer systems with minimal impact.

In the aftermath of the Internet Worm unleashed in 1988 by Robert Morris, Jr. interest in intrusion detection as a practical science has increased. Kumar and Spafford explored using system call trace analysis to reveal pattern signatures [Kumar and Spafford, 1994]. They showed that by examining Unix system calls during execution of the *sendmail* program one could identify suspicious activity. Kumar used execution graphs to represent the path of system call execution and portray typical execution paths in the program. Lane and Brodley expanded this research to explore the use of audit trail analysis as a means to identify typical system behavior [Lane and Brodley, 1999]. Their research showed that typical profiles could be established for a variety of system usages and that anomalies to these profiles could reveal intrusion activity.

Another valuable intrusion detection system is the Graph-based Intrusion Detection System (GrIDS) developed at UC Davis [Bishop et al., 1997]. GrIDS models a network as an activity graph where machines represent the nodes in the graph and connections between systems are represented as edges. Benign activity on a network over time forms an activity graph for the network. Current activity is compared against known signatures for hostile activity to produce security warnings.

Data Reduction

Data reduction is important to anomaly detection for two reasons. First, data reduction can reduce amount of storage required to represent the information in a typical data stream. Second, data reduction can be used to represent the data in a form that is more amenable to performing intrusion detection analysis. Several researchers have noted the importance of data reduction in the anomaly detection process. Lane and Brodley showed that by filtering out key data elements that a profile data set could be greatly reduced while yielding improved anomaly-detection performance. Denning also discusses the necessity for data reduction in [Denning, 1987].

Data Aggregation

Data aggregation is the technique of combining data fields together to form new data objects. Frequently, it is used to group dependent data objects. Denning argues that data aggregation is an effective technique for representing the dependence of two or more data elements.

Data Source Selection

Several researchers have discussed which data elements should be selected when developing anomaly detection schemes. Lee and Wenke discuss how classifiers can be induced from *tcpdump* data to distinguish network attacks from normal traffic [Lee and Wenke, 1997]. Their approach used a *tcpdump* trace as a training input to a data classifier algorithm. They concluded that adding temporal-statistical features to the data can greatly improve the performance of the classifier engine. They acknowledge that choosing the right features to analyze in the input data takes considerable time and a deep understanding of the input domain. They also acknowledge that there is need for tools that can provide insight into the patterns that might be exhibited in the data and for tools that can help the security analyst understand the nature of the observed anomalies.

Network Data Reduction

A number of researchers have explored the problem of reducing data to a form that can be analyzed efficiently. Zhang and Paxson have applied packet signature analysis to network traces to reveal patterns associated with intruder backdoors and the use of stepping stone machines to hide the origination point [Zhang and Paxson, 2000]. Their research showed that packet header data was sufficient to identify potentially malicious activity. They also showed that one could eliminate a large amount of network data, such as outgoing web traffic, with no loss of detection accuracy.

Fuzzy Systems

A fuzzy system is an expert system that uses a collection of fuzzy membership functions and rules, instead of Boolean logic, to reason about data. The rules in a fuzzy system are usually of a form similar to the following:

if A is LOW and B is HIGH then C = MEDIUM

where A and B are input variables, C is an output variable, LOW is a fuzzy membership function defined on A, HIGH is a membership function defined on B, and MEDIUM is a membership function defined on C. The "if" part of the rule describes to what degree the rule applies, while the "then" of the rule assigns a membership function to each of one or more output variables. In fuzzy systems the rules may overlap. When this happens, the outcomes are evaluated together and an approximation of the overall response is determined. The set of rules in a fuzzy expert system is known as the rule base or knowledge base. The general inference process in a fuzzy system occurs in four steps:

1. Fuzzification. The membership functions defined on the input variables are applied to their actual values, to determine the degree of truth for each rule premise.
2. Inference. The truth-value for the premise of each rule is computed, and applied to the conclusion part of each rule. This results in one fuzzy subset to be assigned to each output variable for each rule. Usually only MIN or PRODUCT are used as inference rules. In MIN inferencing, the output membership function is clipped off at a height corresponding to the rule premise's computed degree of truth (fuzzy

logic AND). In PRODUCT inferencing, the output membership function is scaled by the rule premise's computed degree of truth.

3. Composition. All of the fuzzy subsets assigned to each output variable are combined together to form a single fuzzy subset for each output variable. This can be done a variety of ways. The two most common methods are MAX and SUM composition. In MAX composition, the combined output fuzzy subset is constructed by taking the point wise maximum over all of the fuzzy subsets assigned to a variable by the inference rule (fuzzy logic OR). In SUM composition, the combined output fuzzy subset is constructed by taking the point wise sum over all of the fuzzy subsets assigned to the output variable by the inference rule.
4. Defuzzification. This is used when it is useful to convert the fuzzy output set to a crisp number. Quite a few defuzzification algorithms are known. Two of the more common techniques are the CENTROID and MAXIMUM methods. In the CENTROID method, the crisp value of the output variable is computed by finding the variable value of the center of gravity of the membership function for the fuzzy value. In the MAXIMUM method, one of the variable values at which the fuzzy subset has its maximum truth-value is chosen as the crisp value for the output variable.

Clustering Techniques for Identifying Rules

While it is possible to create the fuzzy rule base analytically without relying on input or training data, a more common approach is to use real-world data to help define the fuzzy membership functions. This process of deriving fuzzy rule sets from data usually involves analyzing the input and output spaces to find the natural clusters that relate an input domain to an output response. The clusters define the centroids and ranges of the fuzzy rules. For this research, we use the Fuzzy C-means clustering technique.

Fuzzy C-Means Clustering

The fuzzy *c*-means (FCM) algorithm minimizes the objective function [Bezdek, 1981].

$$J(\mathbf{U},\mathbf{V}) = \sum_{k=1}^{n}\sum_{i=1}^{c}(u_{ik})^m \|x_k - v_i\|^2 \qquad \text{subject to } u_{ij} \in [0,1] \text{ and } \sum_{i=1}^{c} u_{ik} = 1 \ \forall\ k \qquad (2)$$

U is the partition matrix that shows to what degree the *k*-th data point x_k belongs to each cluster as measured by its distance from the prototype of the *i*-th cluster, v_i. *m* is a weighting exponent. *c* is the number of clusters and *n* is the number of data points. This well-known algorithm has the steps listed below [Bezdek, 1981]:

1) Fix the number of clusters *c*, $2 \leq c < n$; choose an inner product metric, such as the Euclidean norm and the weighting exponent *m*, $1 \leq m < \infty$. Initialize the partition *U(0)*.

2) Calculate the *c* fuzzy cluster centers $\{\mathbf{v}_i^{(l)}\}$ using

$$\mathbf{v}_i = \frac{\sum_{k=1}^{n}(u_{ik})^m \mathbf{x}_k}{\sum_{k=1}^{n}(u_{ik})^m} \qquad (3)$$

3) Update the partition matrix *U(l)* and $\{\mathbf{v}_i(l)\}$.

$$u_{ik} = \frac{1}{\sum_{j=1}^{c}\left(\frac{\|\mathbf{x}_k - \mathbf{v}_i\|}{\|\mathbf{x}_k - \mathbf{v}_j\|}\right)^{2/(m-1)}} \tag{4}$$

4) If $\left\|\mathbf{U}^{(l+1)} - \mathbf{U}^{(l)}\right\| \le \varepsilon$ then stop, otherwise repeat steps 2 through 4.

Visual Data Mining

Data mining is the process of finding hidden features and relationships in data [Shearing, 1997]. Visual data mining presents data visually in order to help the data analyst see features that would be hard to describe verbally or understand with numerical statistics alone [Buja and Cook, 1996]. Visual data mining is most useful when characteristics in the data are not obvious until the data are viewed graphically. There are quite a few methods for viewing data visually. Some of the more common methods are:

Scatter plots –	Multi-column data records are plotted in 2-D or 3-D. Scatter plots are especially useful for plotting data elements in which one or more of the axes are not numerical quantities (e.g. words or strings). Scatter plot viewers will attempt to display points with similar features together.
Splat visualizers-	Splat visualizers attempt to create clusters from the input data elements to help visualize general structures in the data.
Data classification –	Data classification performs regression analysis on the data to identify which data elements are the most significant to a collection of data elements.

Design Considerations

Default Criteria

There are a number of design criteria that one must be aware of when implementing an intrusion detection system. Since some of the criteria conflict, it is up to the designer to identify those criteria that are most crucial to the goals of the IDS. We can summarize some of the most desirable design goals as follows:

- Timeliness – The reporting of intrusion events to the security administrator should occur within a short time period of time after their onset. The ideal system should be able to report suspicious activity in real-time.
- Reliability – The IDS should be able to identify all intrusion activity. The system should have a low false negative rate [Amoroso, 1999].
- Accuracy – Ideally, an IDS will generate a low number of false positive detections. If the number of false positives is large, the security administrator may learn to ignore the results of the IDS [Amoroso, 1999].
- Flexibility – The security administrator should be able to adapt the IDS to new types of attacks with a minimal of reconfiguration or training of the system. Similarly, one should be able to turn off detection features when they are no longer needed.
- Adaptability – The IDS should be able to adapt to changes in the system being monitored over time. The system should be able to adjust to new meanings of “normal” or “abnormal”.

- Appropriateness – One should be able to configure the IDS in consistence with the security policy in effect [Zamboni, 1998].
- Unobtrusiveness – The IDS should have a minimal impact on system resources. The configuration of the system should not interfere with the normal operation of the systems or networks that it monitors.
- Fault Tolerance – The IDS should be able to function in the presence of IDS component faults. The system should be able to produce valid responses with a degraded number of inputs, though perhaps with a lower confidence in the result.
- Resistance to Subversion – The IDS itself must be resistant to attacks. An attacker should not be able to disable the IDS nor insert false or misleading information to the IDS.

It should be apparent that meeting all of these goals optimally would be difficult. The IDS designer then must identify the most important characteristics for their application and accept the appropriate trade-offs in design criteria. Some of these trade-offs are being driven by changes in the computer security "landscape"

Changes in the Computer Security Landscape

The computer world has been undergoing a startling amount of change in recent years. These changes are forcing security administrators to modify how they do their jobs. This trend is quite apparent in intrusion detection. The increase in switched networks has made it more difficult for network monitoring tools to observe all traffic on a LAN. Increasingly, a computer running in a promiscuous network-sniffing mode will only be able to see the network packets sent from or to it. Network intrusion detection systems will tend to distribute the role of network data gathering to a number of IDS devices and cooperate on the task of data correlation. Another trend that will affect future IDS designs is that network speeds are increasing faster than the ability of individual computers to analyze the potential network traffic. Today it is nearly impossible for dedicated IDS systems to keep pace with the amount of network data that passes through a typical network router [Ranum, 1999]. This is forcing network IDS systems to become more distributed and detection components to be more specific. Also, more and more traffic is encrypted. Web traffic, virtual private network (VPN) connections, and even backdoor channels set up by intruders are some examples of encrypted data that travel over our computer networks. This is making it very difficult for misuse detection systems that rely on the ability to scan for a particular pattern in the network stream. Instead, monitoring of host-based logs and system activity will play an increasing role in intrusion detection. Finally, there has been a steady increase in mobile computing in recent years leading to huge increase in the number of legitimate users connecting from remote locations or relying on dynamic host address allocation. Network patterns of usage are likely to become more varied, causing network data to become less effective as intrusion detection data source.

We can conclude from these trends that the ID systems of the future will have these characteristics:

- They will consist of multiple monitoring agents that cooperate and share information about the states of the network and computer systems.

- Intrusion detection assessment will increasingly rely on the ability to combine results from network and host-based monitoring processes.

These trends have greatly influenced the design of the FIRE system.

SYSTEM DESIGN

Design Goals

Many of the implementation decisions during this proof-of-concept are influenced by the design of FIRE system. FIRE is designed to be a practical tool used by security administrators for monitoring the security of their networks. The FIRE goals are:

- To demonstrate how fuzzy systems can be used in intrusion detection.
- To identify which data sources provide the best inputs to the fuzzy intrusion detection system.
- To determine the best methods for representing network input data.
- To show how the system can be scaled to distributed intrusion detection involving multiple hosts and/or networks.

Additionally, the system has been designed with two additional goals: to use readily available software and hardware as much as possible, and to be a tool accessible to the system/security administrator.

System Architecture

FIRE is expected to consist of the four types of components shown in Figure 4. There are two types of data collector components, the network data collector (NDC) and the log data collector (LDC). There are two types of data analysis tools: the network data analyzer (NDA), which process network data, and the log data analyzer (LDA), which processes system logs. However, for this proof-of-concept, only the network data processing portions of the system are being investigated. The data analyzer components

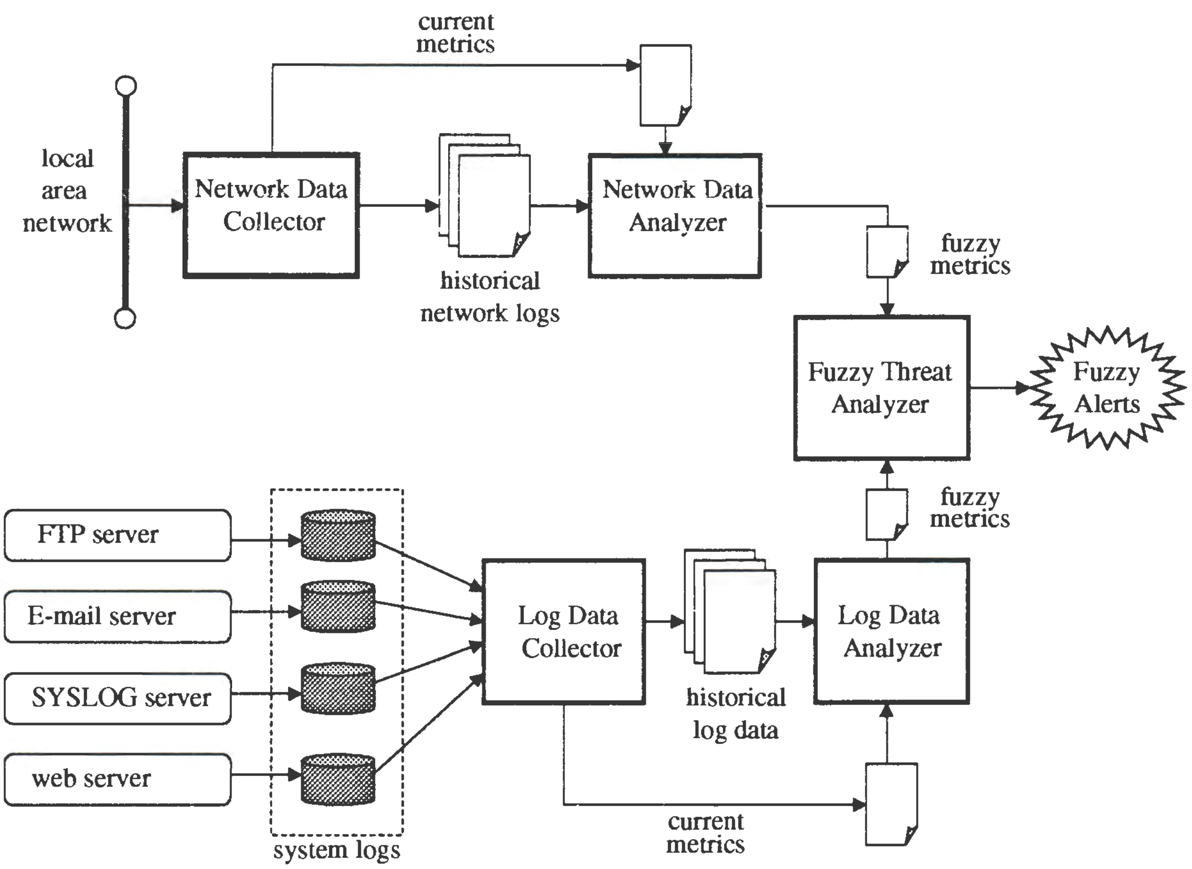

Figure 4. The FIRE architecture. Data is collected from network streams and system logs. Historical data is analyzed against current metrics. Fuzzy metrics are collected from the data analyzers by the fuzzy threat analyzer and used to produce alerts.

produce input values that are read by the fuzzy threat analyzer (FTA). The outputs of the fuzzy threat analyzer are made available to a restricted web server and posted on a web page.

The Network Data Collector

For this proof-of-concept, a Network Data Collector program was developed called *ndump*. This program is run at periodic intervals using the *cron* facility under

Unix. The length of data collection interval is important as it represents the maximum response time between when an attack is started and when the information is reported to the security administrator. Too long a response time may allow too long a period to go by before the administrator is made aware of a serious attack, and too short a time may not allow enough data to be collected for meaningful statistics. Normally, a data collection interval of ten minutes provides a reasonably good trade-off between system response and completeness of data collection. *Ndump* uses the *tcpdump* utility developed at Lawrence Livermore Labs for all raw network data collection. This is similar to the network data collection method used by the Shadow network-monitoring tool [CIDER, 1999]. Since *tcpdump* is only available for variants of the Unix operating system, network data collection units must be Unix-based computers.

Ndump executes in two phases. First, it checks to see if there is a previous *tcpdump* process already running on the computer. If there is, it is stopped and the raw *tcpdump* log file that process was writing is copied to a temporary location on the hard drive. *Ndump* then forks another *tcpdump* process to begin collecting packets to a new *tcpdump* log. During the second phase, *ndump* extracts the data from the *tcpdump* log copied to the temporary location. *Ndump* makes one pass through the log file, sorting information according to TCP, UDP, and ICMP packet types.

As it sorts the *tcpdump* log file, *ndump* performs important filtering, data aggregation, and reformatting of the data to creates a very compact summary of the network data seen during the data collection interval. Outgoing web traffic (i.e. packets with a local source address equal to the local domain and with a destination port of 80) is discarded since there is simply too much data of this type and it is generally uninteresting from an intrusion detection perspective. The data is reformatted and appended to an historical log of packet data. Each packet type (i.e. TCP, UDP, and ICMP) has its own historical log. We will expand on the content of the historical logs when we discuss the data reduction process performed by the network data analyzer program.

It is important to note that *ndump* processes the network data collection in two phases in order to minimize packet loss. While developing *ndump*, it was discovered that attempting to sort or reformat each packet as it was read by *tcpdump* caused an I/O bottleneck leading to dropped or lost packets. This is caused when network packets arrive at the input queue of the network interface card too frequently to allow the packet processing routines to sort, format and log all packets waiting in the read buffer on the interface card. The interface card simply drops the oldest packets in the input queue to make room for newly arriving packets. To prevent this, *ndump* processes the network data in two parallel phases. The raw *tcpdump* data collected for the previous collection interval is processed while the current collection interval is saving data to a new tcpdump log. In our tests, we could avoid packet loss altogether on a moderately busy, unswitched, 10Mb/s Ethernet interface by writing the tcpdump output directly to disk and processing it afterwards.

The Network Data Analyzer

The goal of the network data analyzer (NDA) is to derive network metrics from the historical logs and to update the fuzzy rule constraints immediately afterwards. For this proof-of-concept implementation, the data reduction function and fuzzy rule updates are performed separately. The fuzzy rule constraint updates are performed by hand using *Matlab.* It is expected that in the FIRE system, these two functions will be incorporated into the data processing performed by the *ndump* program. The current network data reduction program, called *plotter*, is written in Perl. The *plotter* program runs on the same computer where *ndump* is run to avoid adding extra network traffic to the data stream.

Plotter reads the historical network logs of network data and creates numerous derived elements. The kinds of metrics produced by the plotter program is crucial to determining what types of intrusions can be detected.

Data Description

Network Data Reduction

Because we must create statistics of the network traffic over time, it is important to save network data for periods of a week or more. Saving a month's worth of data from a typical network would pose a considerable storage problem. We have found that using data aggregation enables us to create relatively compact logs without significant loss of network information. Our basic approach is to store network data under an aggregate data key that represents a network connection or service points. There are three types of aggregate keys: the *sdp*, the *sdt*, the *servid.* The *sdp* is formed by concatenating the source IP address, destination IP address, and destination port into a single key. The *sdt* performs the same function as the *sdp*, however it applies only to ICMP connections. It is composed of the source IP address, the destination IP address, and the ICMP type. The *servid* is an aggregate key composed of the destination IP address and the destination port. In each case, the aggregate key becomes an efficient way to represent the type of service connection between hosts on the network.

TCP State Monitoring

UDP and ICMP data are relatively easy to monitor because they are connectionless. Though we treat each UDP or ICMP packet as a unique connection in this analysis, this is a simplification, since some ICMP and UDP dialogs depend upon primitive state information. For TCP data, however, it is very important to be able to monitor the state of the connection. To do this, we must observe the TCP flags sent between two hosts and record the state using a state machine. For this research, the network data collection program tracks the TCP state information. The TCP state machine works as follows:

1. For each packet observed, a connection ID is constructed that consists of the source IP + source port + destination IP + destination port. If the connection id has not been observed before, then we flip the ID around to see if the id has been observed by the designation destination IP + destination port + source IP + source port. A new connection ID is added to the list of connection IDs. Initially, the connection is given a TCP state of 0.
2. The TCP flags are extracted from each TCP packet used to update the state of that TCP connection.
3. If the TCP flags are consistent with the expected state of the connection, the connection state is updated as necessary. In order to assess the state, it is necessary to record the TCP sequence number and offset, and to calculate the expected sequence number for a future packet.

4. The normal progression of TCP states is from SYN → SYN-ACK→ ACK → FIN → FIN-ACK → FIN-ACK. Alternatively, the FIN shutdown sequence can be skipped when one end of the connection issues a RESET, in which case an acknowledgement is not required.
5. There are two common situations that occur when a connection is not successful. A connection attempt to a closed port would see a dialog of SYN → RESET. A prematurely closed connection would see a sequence of SYN→ SYN-ACK→RESET.

While the behavior described above is certainly not an exhaustive treatment of all TCP states possible or even those commonly observed, it forms the vast majority of all TCP traffic on a network.

Exceptions to TCP State Processing

Intruders frequently make use of undefined behavior in the TCP specification to gather information about open service ports on a host. For this reason, it is not sufficient to monitor only those connections that obey the normal TCP startup and shutdown sequences. For example, if a server host receives a FIN connection on an open port from a client before a SYN->SYN-ACK startup sequence has been properly initiated, the host will normally issue a TCP RESET packet in response. An intruder can use this behavior to help expose those ports that are in use on a system without ever starting a proper connection. This type of port scanning is sometimes called "stealthy" port scanning since these exceptions are dealt with by the operating system kernel and not forwarded to higher level access control tools such as *TCPdump.* Since some of the more interesting network traffic from an intrusion detection point of view lies in these failed connections, the TCP status of every TCP connection is recorded by *ndump*, regardless of its outcome.

Another difficult issue to contend with in TCP data is the problem of long-lived connections. Many types of client/server applications start connections and keep them open for weeks or even months at a time. This makes is very difficult to establish which end of the connection is the server, and which end is the client. Though one might assume that a host receiving connections on a so-called well-known port must be the server, this would cause one to miss a great deal of scanning or backdoor type connections.

Data Selection

The *sdp* is a key composed of the source IP address, destination IP address, and destination port number of a network connection. The *sdp* can used to reference both TCP and UDP connections. The *sdt* is composed from source IP address, destination IP address, and ICMP type of an ICMP connection. As this suggests, the *sdt* is used to represent ICMP data. The served is composed of the destination IP address and destination port of hosts that are providing a service on a port. The servid is only applicable to machines that are observed providing a TCP service. The reason for this is to help detect connections to unusual ports such as one would see in a Trojan horse or backdoor intrusion [Ruiu, 1999]

Data reduction extracts essential information from the raw data and stores it in a form that allows for fast recalculation of statistics. We have found that by pre-processing the network data to extract particular measures, we can greatly reduce the amount of historical data we need to save while simultaneously retaining only the data that enhances our ability to detect anomalies.

The scheme for reducing network data is as follows:

1. For TCP data, ignore outgoing web traffic. This is a reasonably safe assumption since web surfing produces extremely random connection patterns and is generally uninteresting security-wise. However, we do maintain a list of internal web servers. Incoming web traffic destined for hosts not on our list of internal web servers is tracked.
2. Other than above, track all TCP, UDP, and ICMP traffic.
3. Discard all other packets.
4. Monitor the state of TCP connections. Successful connections that followed the normal TCP 3-way handshake to start up, and which followed the normal shutdown sequence are recorded with a state of 6. Successful connections that are reset are given a state of 7. Failed connections are assigned the state of 10. Half-open connections are assigned a state of 9. All of these states are recorded in a log according to according to *sdp*. This allows many connections to be represented with a single log entry.

Even by taking these steps, there are still too many packets to store for historical purposes. We can further reduce the amount of information we need to store by considering the importance of data to the detection of anomalies. For this analysis, we ignore packet payload data. Each a summary representing the outcome of a connection is saved. Assuming that "friendly" machines frequently make the same network connections to the same types of network services (FTP, telnet, etc.), we are generally only interested in identifying unusual host/service combinations.

We apply a novel method to represent a host/service combination. If we form an aggregate field by concatenating the source IP address, destination IP address, and destination IP port of TCP headers together, we create a new field we refer to as the *sdp*. The *sdp* represents a particular service connection between two hosts. We can then use the *sdp* as the basis for our statistical analysis of the network data. The TCP log data can be reduced to a summary of *sdp* observations. In fact, the TCP historical log created by the NDC is simply a list of the number of packets seen for each *sdp* during a collection interval. By pre-processing the network data in this fashion, we can reduce the amount of storage required to less than 1% of that required to store the raw input data.

Network Metrics

There are several groups of metrics gathered for the network data. There are general metrics which summarize TCP, UDP, and ICMP connections for a time interval. For UDP and TCP, detailed metrics are also gathered for each observed service port in the time interval. Additional metrics are gathered for TCP servers. The general metrics gathered for TCP connections are shown in Table 1. The values collected for TCP port statistics are shown in Table 2. Table 3 lists the values that are extracted for each observed TCP server observed. General metrics gathered for UDP connections is shown in Table 2. General metrics for ICMP traffic is shown in Table 3.

Table 1. General TCP Metrics.

Name of Metric	**Description**
NumConnections	Number of connections seen in a time interval.
NumAttempts	Number of connections attempted (SYN observed).
NumFailed	Number of connections failed (RESET in response to SYN).
NumOpened	Number of connections that were successfully opened.
NumCompleted	Number of connections that were properly completed (normal shutdown sequence).
NumReset	Number of proper connections that were ended with a RESET.
NumSDPs	Number of SDPs observed.
NumSrcs	Number of source IP addresses observed.
NumDests	Number of destination IP addresses observed.
NumForeign	Number of Foreign IP addresses observed.
Num Services	Number of service ports observed in use (successful connections only).
NumServers	Number of hosts that accepted a connection on a service port.
NumUniqSDPs	Number of unique SDPs observed during the interval.
NumUniqSrcs	Number of unique source IP addresses observed during the interval.
NumUniqDests	Number of unique destination IP address observed during the interval.
NumUniqForeign	Number of unique foreign SDPs (source IP address from outside the sub network).
NumUniq Services	Number of unique services observed.
NumUniqServers	Number of unique servers observed.

Table 2. TCP service port metrics. These metrics are gathered for each service port observed during each time interval.

Name of Metric	**Description**
NumSrcs	Number of source IP addresses that connected to this port.
NumUniqSrcs	Number of unique source IP addresses that connected to this port. (successful or not)
NumDests	Number of destination IP addresses for this port (successful or not).
NumNewDests	Number of unique destination IP addresses (servers) for this port.
NumSDPs	Number of SDPs observed connecting to this port (successful or not).
NumUniqueSDPs	Number of unique SDPs observed connecting to this port (successful or not).
ForeignSDPs	Number of foreign SDPs observed connecting to this port (successful or not).
UniqForeignSDPs	Number of unique foreign SDPs observed connecting to this port (successful or not).
NumServers	Number of hosts observed accepting connecting connections on this port.
NumUniqServers	Number of unique hosts observed accepting connections on this port.
Connections	Number of connections observed on this port (successful or not).
Attempts	Number of attempted connections on this port (SYN sent).
Successes	Number of successful connections on this port.
Failed	Number of failed connections on this port.
Unclaimed	Number of unclaimed connections on this port.

Table 3. TCP server metrics. These metrics are gathered for each observed server at each time interval. A server is identified as any host that receives a properly initiated TCP connection from another host.

Name of Metric	Description
NumClients	Number of clients that successfully connected to this server.
NumUniqClients	Number of unique clients that successfully connected to this server.
ForeignClients	Number of foreign clients that connected to this server.
UniqForeignClients	Number of unique foreign clients that connected to this server.
Connections	Number of successful connections to this server.

Table 4. General UDP Metrics. These metrics are gathered for each collection interval.

Name of Metric	Description
NumConnections	Number of connections seen in a time interval.
NumFailed	Number of connections failed (An ICMP port unreachable error was recorded).
NumSDPs	Number of SDPs observed.
NumSrcs	Number of source IP addresses observed.
NumDests	Number of destination IP addresses observed.
NumForeign	Number of Foreign IP addresses observed.
Num Services	Number of service ports observed in use (successful connections only).
NumServers	Number of hosts that accepted a connection on a service port.
NumUniqSDPs	Number of unique SDPs observed during the interval.
NumUniqSrcs	Number of unique source IP addresses observed during the interval.
NumUniqDests	Number of unique destination IP address observed during the interval.
NumUniqForeign	Number of unique foreign SDPs (source IP address from outside the sub network).
NumUniq Services	Number of unique services observed.
NumUniqServers	Number of unique servers observed.

Table 5. General ICMP Metrics. These metrics are gathered for each collection interval.

Name of Metric	**Description**
NumConnections	Number of connections seen in a time interval
NumFailed	Number of connections failed (An ICMP port unreachable error was recorded)
NumSDTs	Number of SDTs observed
NumSrcs	Number of source IP addresses observed
NumDests	Number of destination IP addresses observed
NumForeign	Number of Foreign IP addresses observed
Num Types	Number of ICMP types observed in use
NumUniqSDTs	Number of unique SDPs observed during the interval.
NumUniqSrcs	Number of unique source IP addresses observed during the interval
NumUniqDests	Number of unique destination IP address observed during the interval
NumUniqForeign	Number of unique foreign SDPs (source IP address from outside the sub network)
NumUniq Types	Number of unique ICMP types observed.

Fuzzy Systems Development

Fuzzy Input Value Generation

The system uses two techniques to create fuzzy sets: Fuzzy C-means clustering, and statistical distributions. The choice of which fuzzy set generation technique to use is determined solely by the expected profile of the observed data. Data that reveals a good distribution of values over time appears to provide the best results when Fuzzy C-means clustering is utilized. Statistical methods appear to be best when applied to data that has a sparse distribution in observed values.

To understand how different input data can have such vastly different input sets, it is helpful to look at a couple of examples. One of the values that the system measures is the number of new servers observed on the network. We would expect that in a stable network, weeks might transpire before a new legitimate server is added to the network. This means that most of the time the number of new servers observed will be zero. There would be no discernible clusters from which to construct fuzzy rules. In this case, we simply revert to a conventional statistical model where the fuzzy term MEDIUM indicates those values that lay within one standard deviation of the mean, MEDIUM-HIGH represents values between 1 and 2 standard deviations above the mean, and HIGH is applied to values greater than two standard deviations above the mean.

Fuzzy Output Values

The output of the fuzzy system corresponds to the degree to which the fuzzy system detects the intrusion condition it is designed to detect. For consistency, the same output membership function is used for all of the fuzzy systems. There are five membership functions in each output set: LOW, MED-LOW, MEDIUM, MED-HIGH, and HIGH. The membership functions are uniformly distributed in the range from 0.0 to 1.0. Figure 5 shows a typical set of output membership functions for an alert.

Creating Fuzzy Rules for Intrusion Detection

With the fuzzy input sets defined, the next step is to write the rules for detecting each type of attack. A collection of fuzzy rules with the same input and output variables is called a fuzzy system. We assume that the security administrator can use their expert knowledge to help create a set of rules for each attack. The rules are created using the fuzzy system editor contained in the Matlab Fuzzy Toolbox. This tool contains a graphical user interface that allows the rule designer to create the member functions for each input or output variable, create the inference relationships between the various member functions, and to examine the control surface for the resulting fuzzy system. But it is not expected that the rule designer will rely solely on intuition to create the rules. We feel that visual data mining can help the rule designer understand data features are most relevant to detecting different kinds of attacks.

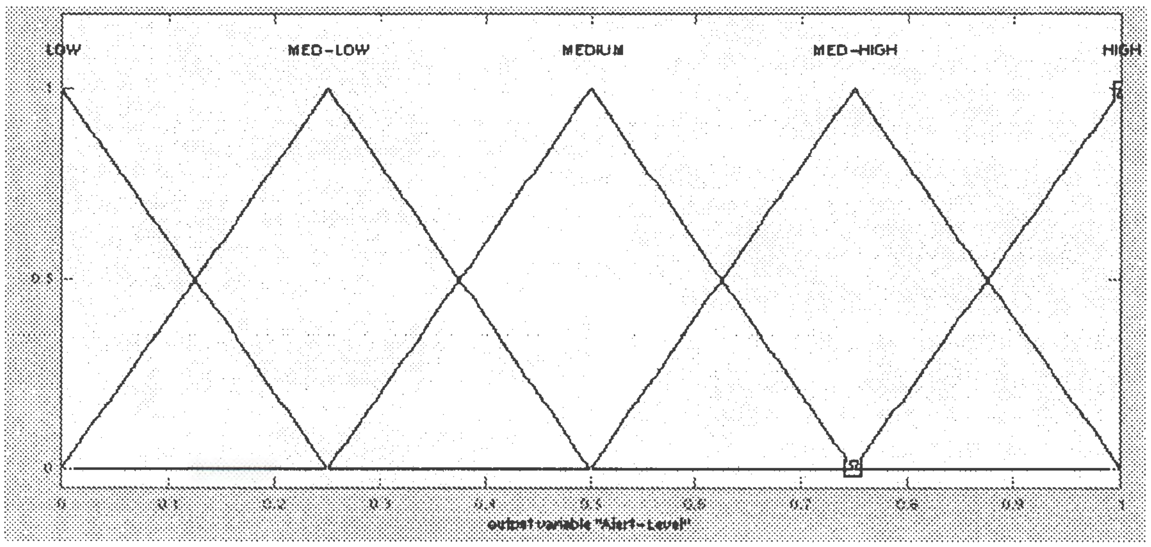

Figure 5. Typical fuzzy output membership function used by each detection fuzzy system. The membership functions correspond to the degree in which an alert condition is true.

Data Mining Tools

Visual data mining plays an important role in this analysis. A good data mining tool must provide several key capabilities:

Ad Hoc Queries	The analyst should be able to pose "what if" kinds of questions against the data.
Filtering	The tool should be able to select data based upon certain filtering criteria.
Data aggregation	One should have the ability to join fields together and perform queries, etc. against the aggregated data.
Binning	The system should be able to create histograms for so-called "parameter-less" data such as strings and other non-numeric data.
Clustering	Ideally, the system should be able to recognize similar data records and group them.

The Mineset program from Silicon Graphics Incorporated is used for all data mining analysis. This tool provides all the features listed above. However, one of Mineset's strengths is in the 3-D visualization of data. The data-mining analyst is expected to use it interactively at a graphics monitor. As a result, its printed output is not as user friendly as a conventional plotting program.

EXPERIMENTS

Methodology

The following approach was used to test the fuzzy intrusion detection system. The *ndump* program was installed on a machine in the Information Systems Security Laboratory that could "see" other traffic on the network. The system gathered data for an extended period of time (usually 7 to 14 days) to develop "normal" network profiles. After this period, attacks were launched against systems on the ISSL network to simulate intruder behavior. The normal and attack network data were processed by the *fire* program to derive the set of metrics. Visual data mining software was used to plot various combinations of metrics to see what features were useful for observing the attacks. Next, fuzzy rules were written that used the metrics that seemed most useful for visualizing the attacks.

The normal profile data was used to create the values for the fuzzy member functions of fuzzy rules. Fuzzy c-means and conventional statistical means were used to create the centers and limits for the fuzzy sets. We then ran the attack data against the fuzzy systems created from the normal profiles to see if the fuzzy system for each kind of attack would produce the desired alert values. .

The Test Environment

For the test environment, we used computers and networks available to us in the Informations Systems Security Laboratory (ISSL) at Iowa State University. The lab consists of approximately thirty computers of various operating systems and hardware configurations. In general, all systems are directly connected to the Internet. Since the machines were directly connected to the Internet, we frequently observed suspicious activity occurring “in the wild”. When unusual data was observed, it was often necessary to analyze the source of the suspicious activity to confirm the nature of the traffic. Sometimes, this revealed rather surprising results.

Attack Phases

The technical approach was tested using different types of intrusion activity that reflected the typical phases an attacker would use against a target. A determined intruder will often follow four phases to gain access to a system [McClure, 1999]:

1. Surveillance. The attacker seeks to learn what the target consists of with regards to hosts on a network, operating systems, and kinds of services available.
2. Information Gathering: Attempting to identify host names, account names, or shared resources on a network.
3. Penetration. In this phase, the attacker tries to gain access to a resource of the target. This could involve taking administrative or root privileges on the target, or installing a program that performs on behalf of the attacker.
4. Exploitation. In this phase, the attacker uses a target resource in a way that violates the security policy of the target. This could mean extracting data, installing a backdoor, or even denying a service on the system. (Unauthorized Servers).

An intruder does not necessarily follow these phases in sequence. Some attacks may attempt the penetration and exploitation phases first and skip the surveillance and information gathering phases.

Attack Scenarios

A different attack was selected to represent each phase. The attack used and the phases they represent are shown in Table 6.

Table 6. The attacks used for each intrusion phase.

Phase	Attack
Surveillance/Target Acquisition	Nmap scans
Information gathering	Portmapper probe
Penetration	BIND buffer overflow
Exploitation	Denial of Service (Pingflood)
Exploitation	Installation of unauthorized server.

Scenario 1: Host or Port Scanning

In a common intrusion scenario, an attacker conducts a remote scan of a network or hosts, trying to identify which hosts exist and which service ports are in use. Typically, this type of information gathering is performed as a precursor to a more specific attack. There are three general forms of scanning: network scans, service scans, and host scans. A network scan seeks to find which hosts on a network are up. A service scan is usually scans many machines to see if specific services are installed, such as FTP or SMTP. A host scan targets a single machine to find out which TCP or UDP ports are in use, though a particularly aggressive attacker may launch a host scan against many hosts. There are several reasons why an attacker may perform host or post scans on a network:

1. The attacker may be simply trying to find out which hosts are up on a network or find out which machines are associated with a given domain. Machines within the same name service sub-domain (e.g. .iastate.edu) may be sharing resources or account information. This fact may be useful to the attacker.
2. An attacker may be trying to find out if there are specific servers (e.g. FTP) in use on a network in order to exploit a known vulnerability in the server.
3. The attacker may be trying to identify the operating system on the target machine using OS fingerprinting. Knowledge of the services running and the host operating system is extremely valuable to the attacker because it helps to narrow the types of vulnerabilities the attacker can exploit on the systems.

To test this scenario, network and service scans were run on local subnets in the ISSL, and host scans against several test systems in the ISSL. All scans were performed using the *nmap* tool because it implements several types of scanning techniques and since it is probably the most common scanning tool used by attackers. The network scans were launched both from a machine on the ISSL local area network, and from a "foreign" machine outside the ISSL LAN. The scans were run with at least 30 minutes between scans so that the plots wouldn't appear cluttered. Prior to running the attacks, data was collected by the system for approximately two weeks to produce a good sense of "normal" behavior.

After the attacks were completed, the *plotter* program was run to produce the set of metrics for each time ten minute time interval of the preceding fourteen days. The unreduced data for the network scan is shown in Figure 6. In this figure, time is shown along the right axis increasing toward the viewer, and individual SDPs are shown along the back axis. The number of packets sent for each SDP is shown in the vertical axis. From this plot, the port scan are easy to observe as horizontal structures along the base of the graph. This is to be expected since a port scan is likely to generate a large set of unique SDPs in a short amount of time. Also, a port scan frequently sends just one or two packets on each port, so we would expect the data points to lie along the base of the plot. These plots suggest that the number of unique SDPs in a period of time of time is strong indicator for a scan. To tell whether the scan is a network, service, or host scan, we will need to combine this with other metrics.

With each type of scan, there are telltale signs that we can use when devising the detection rules. If the attackers use stealth TCP scanning techniques, we would expect to see a large number of unusual TCP SDPs, and a large number of TCP reset packets. However, if an attacker used a stealth TCP scan for a single service, he would be able to simultaneously perform a network scan and a service scan, so it would be extremely difficult to identify the intent of the scan. However, if more than one service port on a network received an unusual number of SDPs then we could conclude primarily that a service scan was taking place. Naturally, if a large number of service ports were observed in an interval and the number of destinations hosts was small, we could conclude that a host scan was taking place.

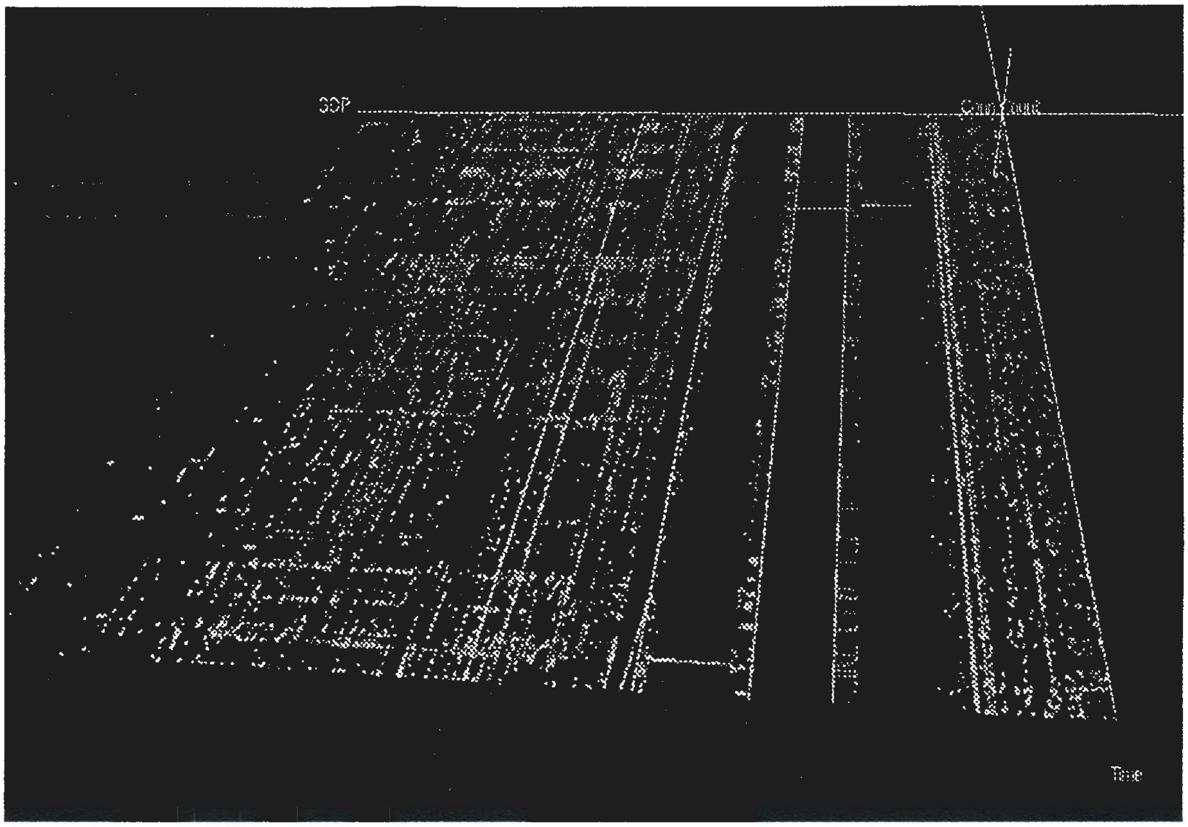

Figure 6. 3-D Plot of unreduced TCP data for a two-week period that shows patterns of TCP connections. Each point represents a potential connection between two hosts on specific service port. Three types of scans appear as short horizontal bands of points. Long vertical bands are regularly occurring connections. Time is shown on the right axis with most recent data nearest the viewer. The back axis is histogram of SDPs observed during the collection period.

Figure 7 attempts to illustrate this argument. The figure shows a 3-D plot of three metrics gathered over the two-week data collection period both prior to the scans being run and including the scans. The plot shows the three metrics that we are interested in: number of unusual SDPs, number of destination hosts observed, and number of service ports observed. The host scan stands out as a lone point at the upper right of the graph, the service scan as the point in the middle on the bottom of the graph, and the network scan as a point nearest the user. This graph serves to suggest that these three metrics alone may be sufficient indicators for identifying TCP scanning activity.

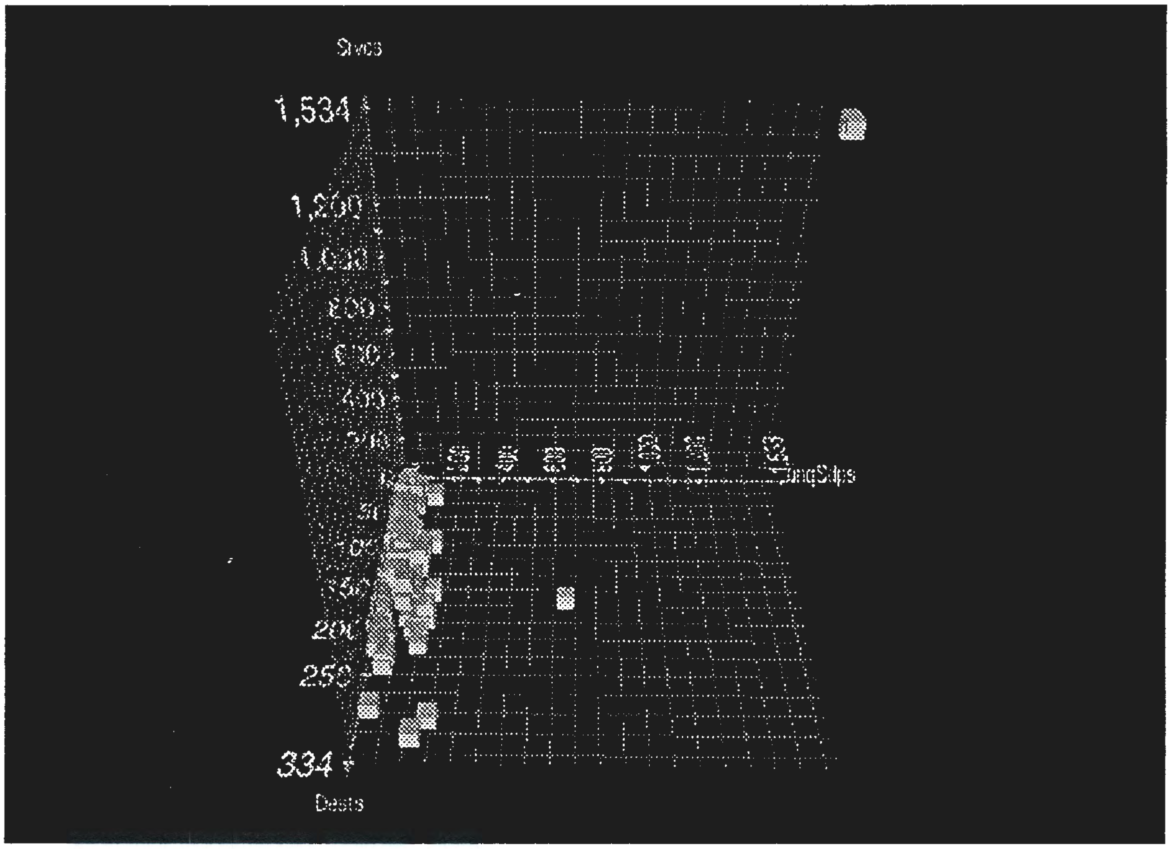

Figure 7. A 3-D plot of the count of unique SDPs, destination hosts, and service ports for each data collection interval over a two-week period. The host scan appears as the lone point at the upper right, the service scan as the lone point out in the middle, and the network scan as the point nearest the viewer at the bottom.

Using these arguments as a basis, we can devise some fuzzy rules that would be most representative of a particular type of scan. The metrics used are the count of unusual SDPs, the count of destination hosts, and the count of observed service ports.

A TCP Service Scan
If the COUNT of UNUSUAL SDPs is HIGH
And the COUNT of DESTINATION HOSTS is HIGH
And the COUNT of SERVICE Ports observed is MEDIUM-LOW
Then Service Scan of Port N is HIGH

A TCP Network/Service Scan
If the COUNT of UNUSUAL SDPs is HIGH
And the COUNT of DESTINATION HOSTS is HIGH
And the COUNT of SERVICE Ports observed is LOW
Then Service Scan is HIGH and Network Scan is HIGH

A Host Scan
If the COUNT of UNUSUAL SDPs is HIGH
And the COUNT of DESTINATION HOSTS is LOW
And the COUNT of SERVICE Ports is HIGH
Then Host Scan is HIGH.

Figures 8, 9, and 10 show the representation of fuzzy rules for detecting TCP host, network, and service scans respectively. Each fuzzy system is assumed to have five member functions, each with a triangular distribution. For simplicity, we assume that the extents of each rule lie at the center point of the adjacent rule. This assures of complete coverage in the input domain. The input domain is clipped at the minimum of the leftmost and maximum of the rightmost rules so extremely high or low values will still lie within the rule domain.

The next step is to enumerate the fuzzy sets for each type of scan. The fuzzy c-means algorithm was applied to the data gathered during the two-week observation period prior to running the attacks. Table 7 shows the values for rule center and extents obtained using the fuzzy c-means algorithm. These values are used to quantify the rules in figures 8, 9, and 10. The resulting membership functions for number of destination hosts, number of service ports observed, and number of unique SDPs are shown in Figures 11, 12, and 13, respectively.

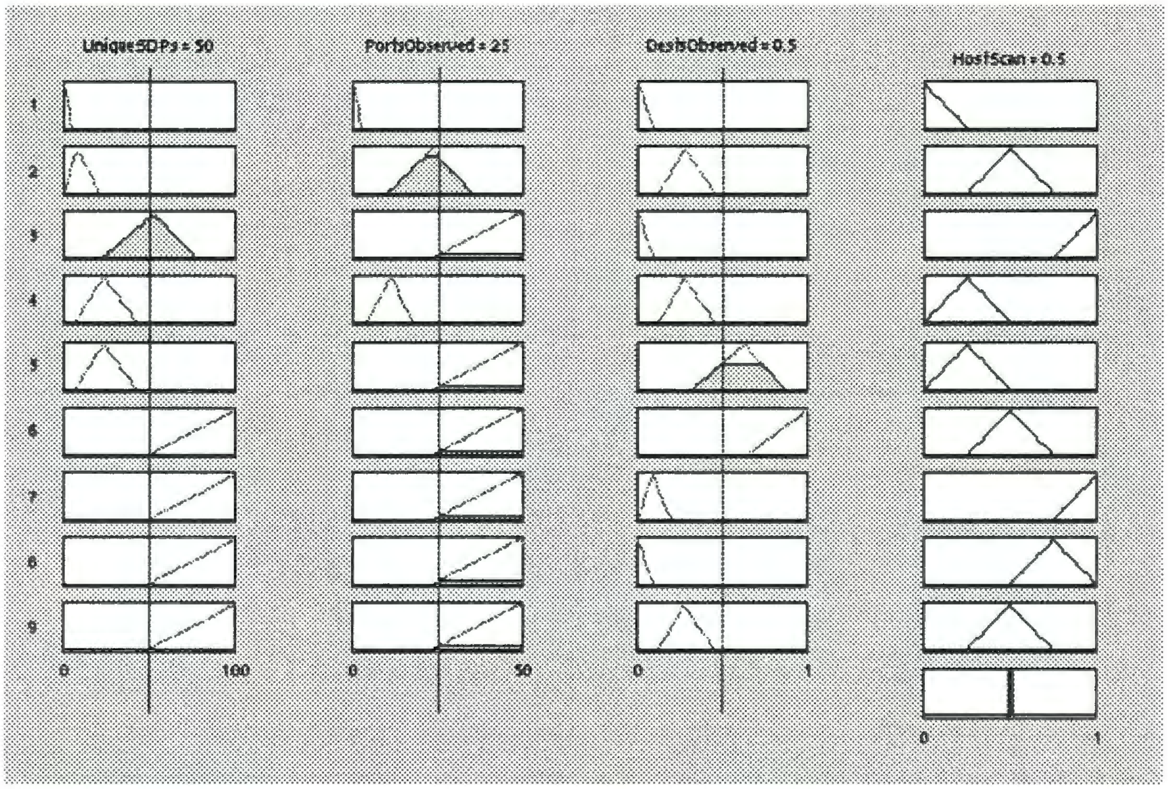

Figure 8. Fuzzy system for detecting a host scan. The inputs are the number of unique SDPs observed, the number of ports observed, and the number of destinations observed. The output is the host scan alert level.

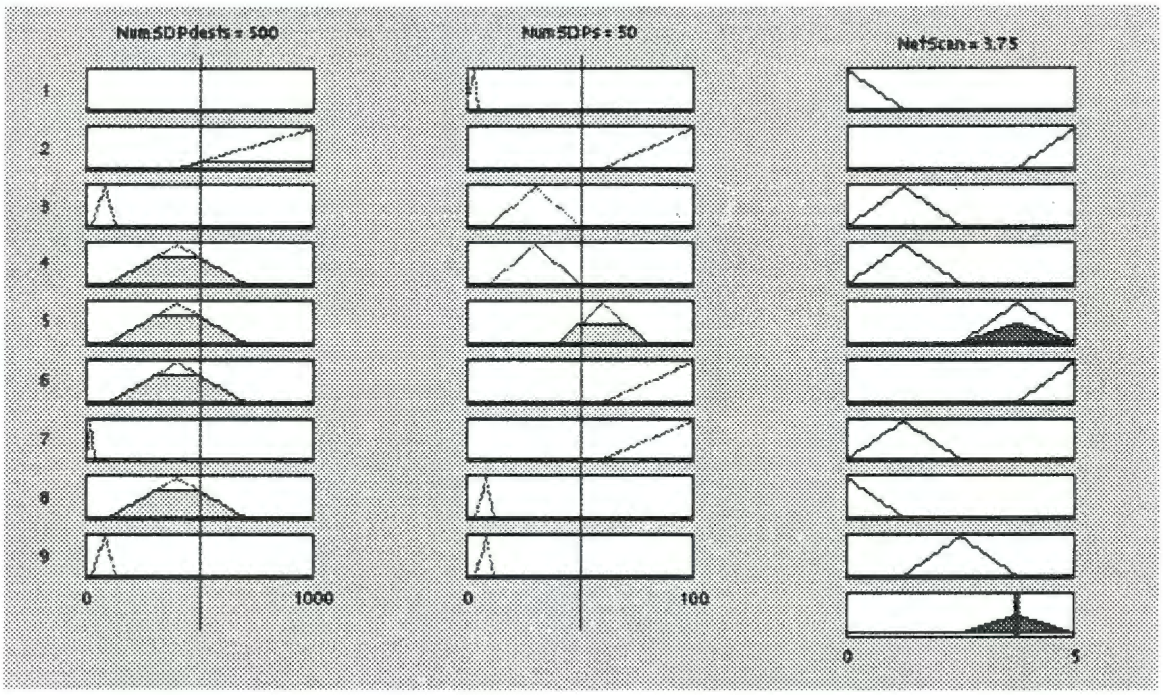

Figure 9. Fuzzy system for detecting a network scan. The inputs are the number of destination IP addresses observed and the number of SDPs observed. The output is the network scan alert level.

Table 7. Rule centers for fuzzy sets obtained by fuzzy c-means clustering for three network metrics

Cluster Number	Number of destination hosts	Number of service ports observed	Number of unique SDPs
1	19.6	15.8	6.6
2	41.9	23.3	12.3
3	99.1	26.4	17.0
4	156.9	68.1	60.7
5	193.0	90.1	155.8

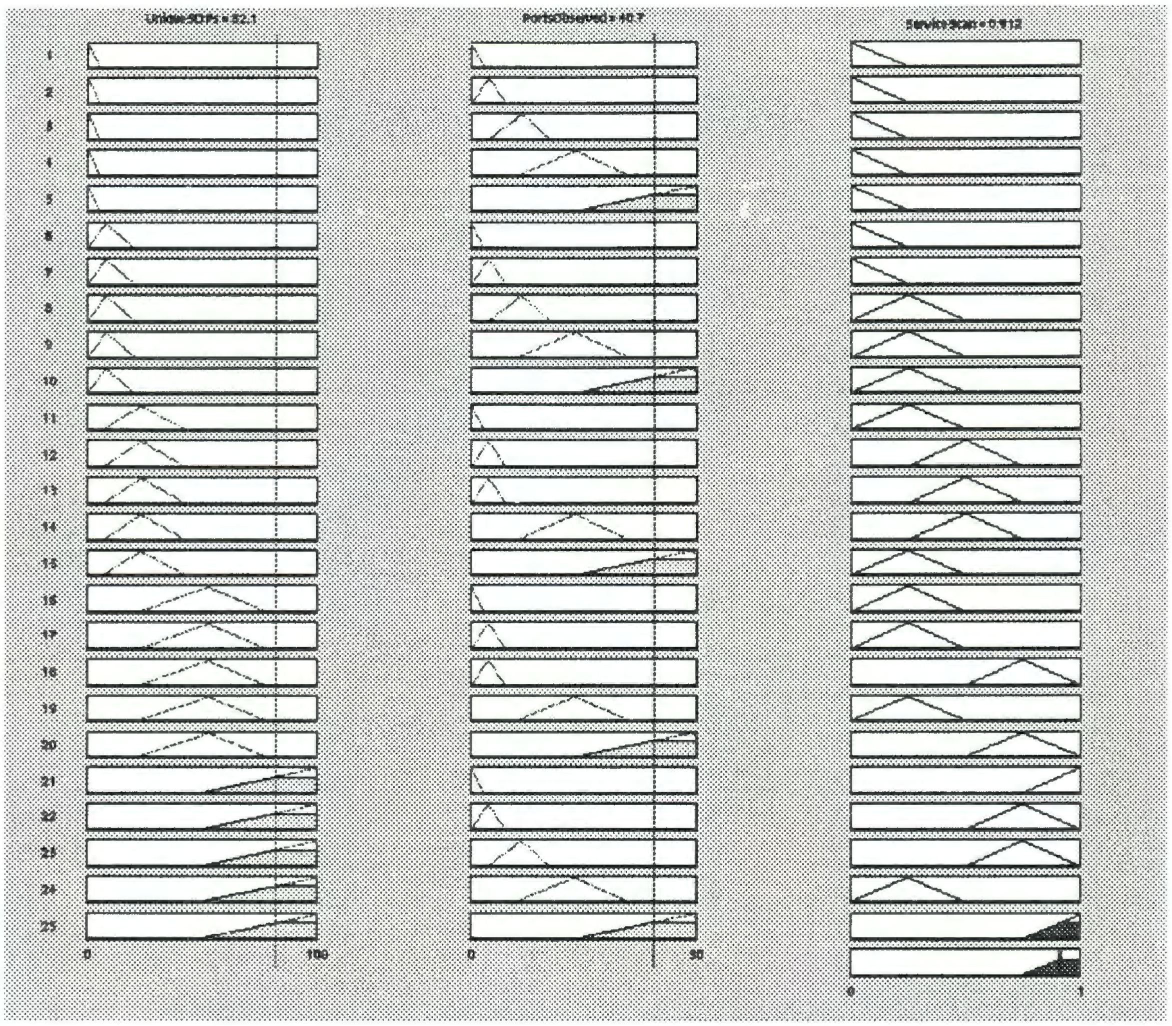

Figure 10. Fuzzy system for detecting a service scan. The inputs are the number of unique SDPs observed, and the number of ports observed. The output is the service scan alert level.

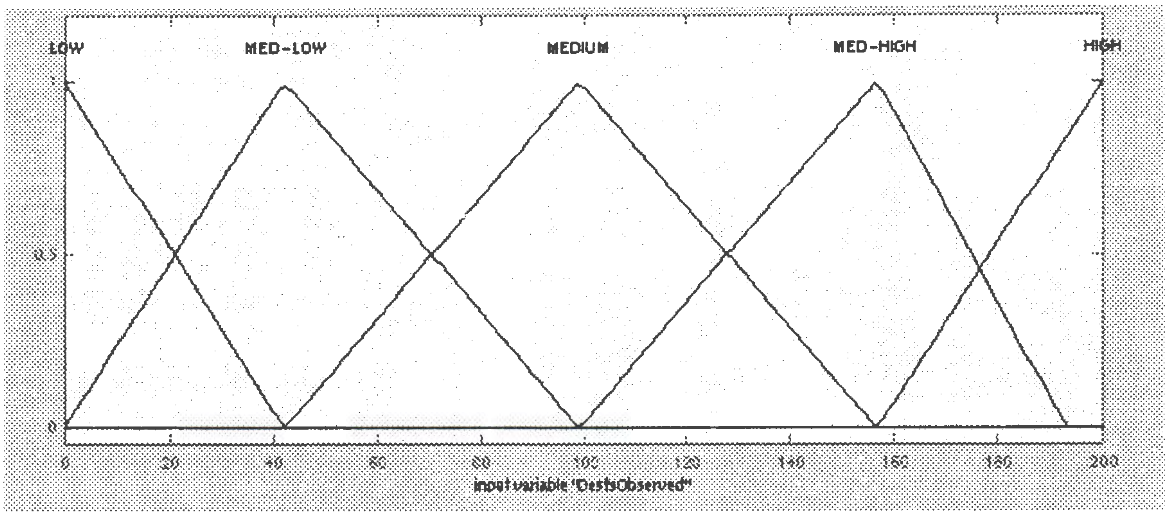

Figure 11. Membership function for number of destinations observed after applying fuzzy C-means clustering to observed data.

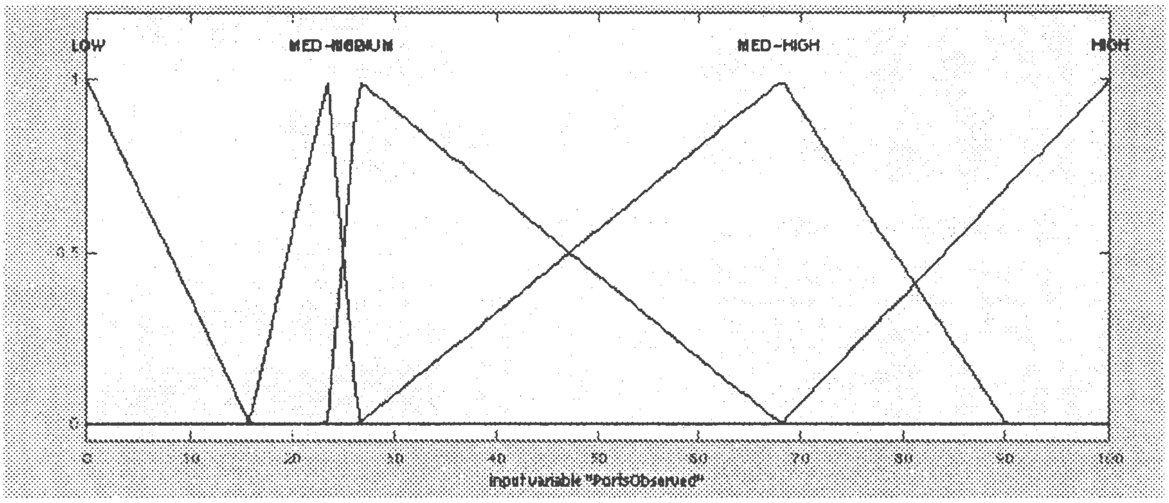

Figure 12. Membership function for number of service ports observed after applying fuzzy C-means clustering to observed data.

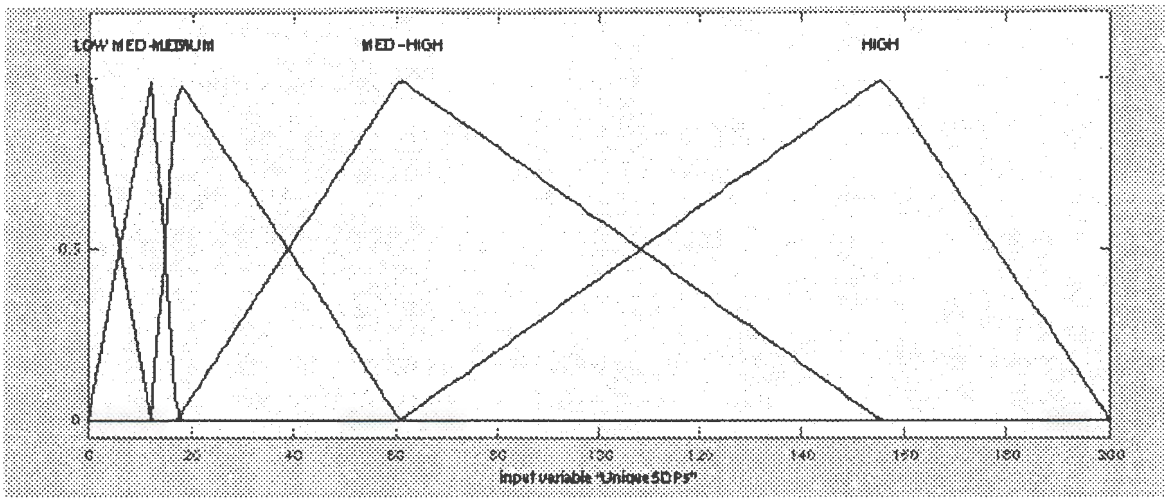

Figure 13. Membership functions for number of unique SDPs observed after applying fuzzy C-means clustering to observed data.

Results

The data collected during the attacks were input to the fuzzy systems for detecting host, network, and service scans. In each case, the fuzzy system issued a HIGH alert value between 0.9 and 1.0. This shows that typical intruder scanning activity can be recognized by these fuzzy detection systems. However, when the data collected in the two weeks prior to running the attacks was entered into the fuzzy systems, quite a few non-attack collection periods also triggered high alert levels. There were 2012 ten-minute collection periods recorded in two weeks of data gathering. Of these, 16 periods would have issued an alert level between 0.85 and 0.92, indicating moderately high to high levels of alert for these scans. Upon further investigation, it was determined that 7 of these data collection periods appeared to have some type of "in the wild" scanning occurring. The remaining 9 data collection periods had high enough SDP counts to trigger the detection scenarios. The primary cause of these false alarms is that the membership functions at the edges of the fuzzy sets need to perform better discrimination between a HIGH and VERY-HIGH input values, though it is also possible to manually adjust the fuzzy sets to that high normal activity does not trigger a HIGH system alert.

Scenario 2: An Portmapper Scan

TCP port 111 is well known to network intruders. It is the service port used by the Sun RPC port mapper program. A number of attacks are designed to be run against services that register RPC programs with the port mapper. Consequently, when an intruder is searching for a vulnerable system, they will frequently seek out hosts running a port mapper program, then query the program to see what RPC services are installed. To detect a port mapper probe, we need to look for unusual connections to port 111. If multiple connections from the same intruder host were made within a short period of time, this would indicate that the intruder was attempting to gather RPC service information. A typical rule for such a probe would be written as:

```
If count of UNIQUE SDPs on Port 111 is HIGH
And COUNT of Foreign SDPs on Port 111 is HIGH
Then PortMapper Probe is HIGH
```

To test the complete set of rules, a portmap scan against port 111 was launched against a portion of the systems on the test network.

Scenario 3: A UDP Buffer Overflow Attack

Buffer overflow exploits against UDP-based services are highly prized by attackers for two main reasons: 1) UDP traffic can be easily spoofed. The attacker can easily hide the origination point of the attack packet and it is not necessary for the attacker to ever receive anything back directly from the target. 2) A single, well-crafted packet that exploits a buffer overflow is frequently all that is needed in order to penetrate a system.

Unlike TCP-based port scanning, it is not always possible for an intruder to tell when a potential target is listening on a given UDP port. The only method an attacker has of telling whether a UDP port is open is by checking for ICMP "port unreachable" responses. However, these responses are unreliable and the UDP scanning process considerable time to scan a whole network. Rather than waste time on a UDP port scan, an attacker will often skip the scanning phase and simply bombard a network with UDP packets that contain a buffer overflow payload against a particular service. We surmise that it is possible to detect a "shotgun" UDP buffer overflow attack by measuring the number of unique SDPs seen against a particular UDP service port.

To test this hypothesis, we conducted a buffer overflow attack against domain name servers on the 129.186.5 subnet using the ADM Worm attack. A plot of UDP SDPs for the two-week collection period that includes this attack is shown in Figure 14. While we can plainly see existence of this attack in the plot, we can also see many other types of UDP scans occurring on the network. Ideally, we would like to have additional discriminators that would help isolate the buffer overflow attacks.

If we consider that we are only concerned about attacks on services that are actually in use on the network, then we can exclude UDP traffic against services that are not in use. Likewise, if we assume that only one service port is being attacked, then the number of observed ports in use during an interval would not be unusual.

Therefore, we can write the essential firing rule for identifying a UDP buffer overflow attack as:

```
IF (COUNT of UNIQUE SDPs is HIGH) and
( COUNT of UDP Ports observed is MEDIUM) then
(UDP ATTACK alert is HIGH)
```

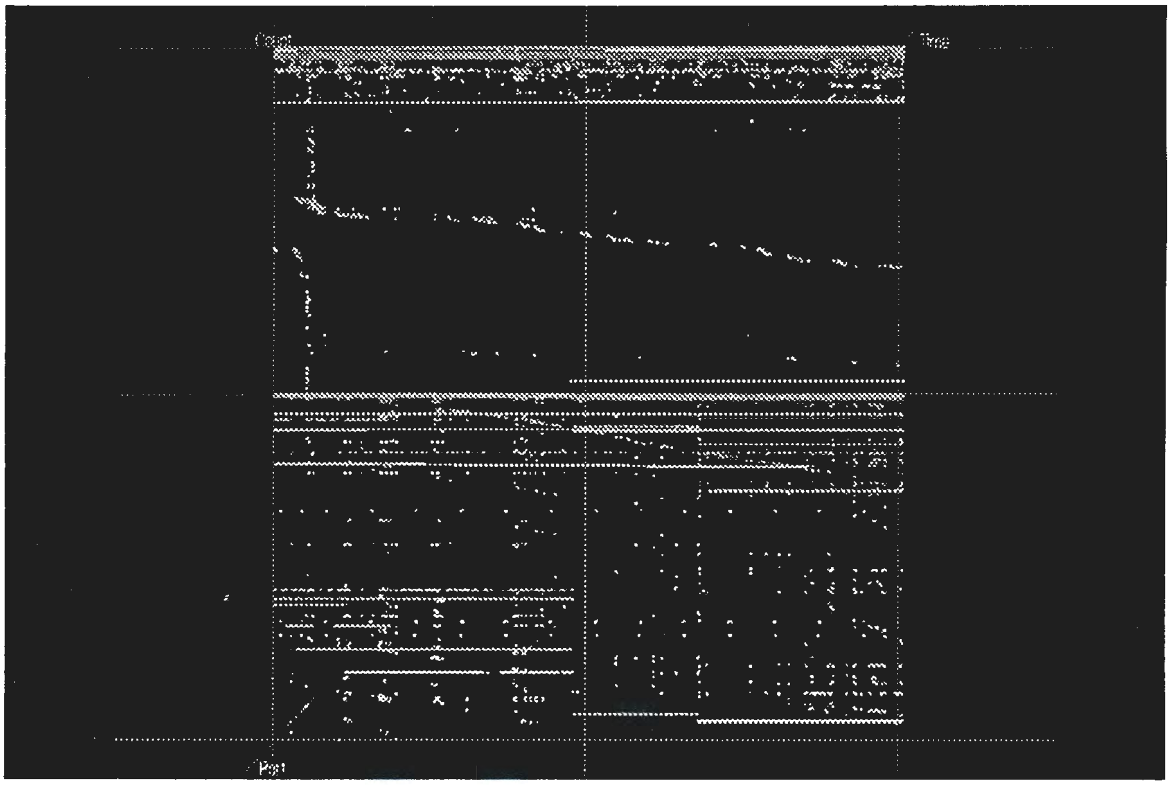

Figure 14. A two week summary of UDP data. Time progresses from left to right along the graph. The vertical column contains a histogram of UDP service ports. Each point represents a UDP connection between two hosts on particular service port. Regularly occurring connections appear as horizontal bands of connections.

The plot in figure 14 shows some interesting features. We see that regularly occurring connections appear as solid bands in the horizontal direction. The ADM Worm attack shows up as a faint band in the vertical direction in the middle of the plot. This suggests that a large number of UDP SDPs against a specific port in a short period of time may indicate some sort of buffer overflow attack. Since we are not recording packet contents, we don't actually know whether the attack is specifically a buffer overflow attack. Nonetheless, we can assume that a such an attack is likely and issue an alert accordingly.

It is also interesting to note the roughly diagonal series of points along the middle of the graph. This indicates a type of UDP port hopping between two hosts on the network, which is a very surprising phenomenon to observe. Looking at this visually reveals features that would be difficult to identify using a non-visual approach.

Scenario 4: A Ping Flood Attack

A ping flood is a common denial of service attack in which an attacking system sends spurious echo reply requests to many hosts on a network causing them to emit a large number of echo replies. This attack causes a network to slow down because of network congestion. We can observe this type of attack quite easily if we simply keep track of the source and destination of echo reply requests. In general, ICMP traffic on a network is very predictable as machines will often use ICMP mechanisms to ping servers or time synchronization services during the course of normal operations. If we plot ICMP traffic over a two week period we see the results in Figure 15. In the scatterplot, time is shown on the right axis, packet count is in the vertical and ICMP SDTs are shown along the left axis. Additionally, we can use a different color or symbol for different

ICMP packet types. The scatterplot shows that there are several regularly occurring ICMP communications between regular hosts on the network.

To see whether we could detect a ping flood, we launched ping flood attack against a range of IP addresses from 129.186.5.1 to 129.186.5.253. We used the *pingflood* tool obtained from the Rootshell site (www.rootshell.com). The results of the pingflood show up on a plot as a band of points running parallel to the SDT axis. We can also see that echo reply requests as a whole are somewhat uncommon.

This plot suggests that it should be relatively easy to construct fuzzy rules for detecting a ping flood attack. Essentially, if the number of ICMP echo reply requests is large and there are also a large number of unusual SDTs observed, then a ping flood must be occurring. This assumption is reasonable since ICMP packets are generally observed

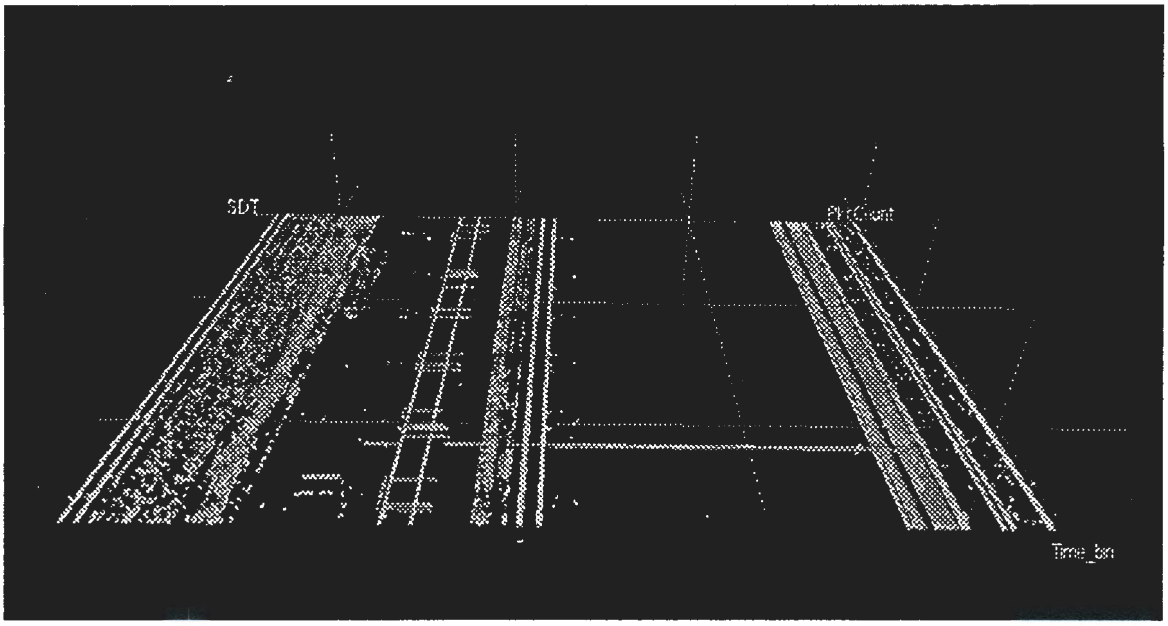

Figure 15. A 3-D plot of two weeks of ICMP traffic. Each point represents an ICMP connection between two hosts during an observation period. The solid vertical bands are regularly recurring ICMP connections on the network. A pingflood attack shows up as a horizontal band of points along the bottom of the graph.

between very specific combinations of machines. A sudden, large increase in the number of unique source/destination host combinations would be highly suspicious. It is important to observe in Figure 15 that simply observing a large number of ICMP echo reply requests was not particularly unusual, as there are several protocols that employ a large number of ICMP echo replies to verify that other allied hosts are still reachable on the network. Therefore it is necessary to qualify the rule with simultaneous occurrence of a larger number of unusual SDTs. A critical rule for detecting a ping flood, then, would be:

```
IF COUNT of UNUSUAL SDTs is HIGH
AND COUNT of ICMP ERQs is HIGH
THEN pingflood ALERT is HIGH
```

Scenario 5: Detecting an Unauthorized Server

In the final stage of an attack, an intruder seeks to take control of a system resource or to disclose private information. A common break-in scenario involves an intruder installing a rogue server, such as FTP or HTTP, through which other intruders may gain access. Frequently, these rogue servers are installed on uncommon ports, or perhaps on a port used for another service. In either case, intruders will often use encrypted channels to help avoid detection.

In this detection scenario, a server is considered to be any host that successfully completes a TCP 3-way handshake from another host. The *servid* aggregate key, composed of the server IP and the service port, represents the server. The *servid* comes into existence whenever a connection is made to the server host.

Since most machines on a network are usually client computers of some sort, we would expect that TCP connections to servers on the network would constitute a small portion of the overall traffic. Likewise, we would expect that new legitimate servers would be added to a network infrequently. Figure 16 shows a scatterplot of TCP activity over a two-week window where time is depicted on the lower right axis, a histogram of server IDs occupies the lower left axis, and the number of connections is on the axis perpendicular to the other two. From this plot, we see that the network is a pretty complicated mixture of data streams. In figure 17, we strip out all data connections except for those in which a proper TCP connection was observed. The result is somewhat startling. Now, we can plainly which servers are actually being connected to on the network. Also, it is fairly easy to discern regularly occurring servers as bands of points that run parallel to the time axis. However, it is not readily apparent on this graph when a new or unique server appears. But if the data points are further color coded according to a measure of the uniqueness of that server ID, then it should become apparent when a new server appears.

From the graphs, we can conclude that the metrics for number of unique servers observed during an interval, and the uniqueness of those servers would both be good metrics from which to develop rules for detecting unauthorized servers on a network. If we assume that we are more sensitive to connections from outside the network, then we can modify the rule so that connections from outside the domain cause the severity of the alert to increase. We can write a representative rule for this scenario as:

IF COUNT of UNIQUE SERVERS is MEDIUM HIGH
And UNIQUENESS of SERVERS is HIGH
And UNIQUE FOREIGN SDPs is MEDIUM HIGH
THEN Unauthorized Server Alert is HIGH

The full set of fuzzy rules for this detection scenario is shown in Figure 18. We use the two-week observation period to create the rule centers. We found that two of the metrics used in the rule set, count of unique servers and uniqueness of servers, were essentially zero for most of the observation period, so we applied a statistical mean to determine the member function quantities.

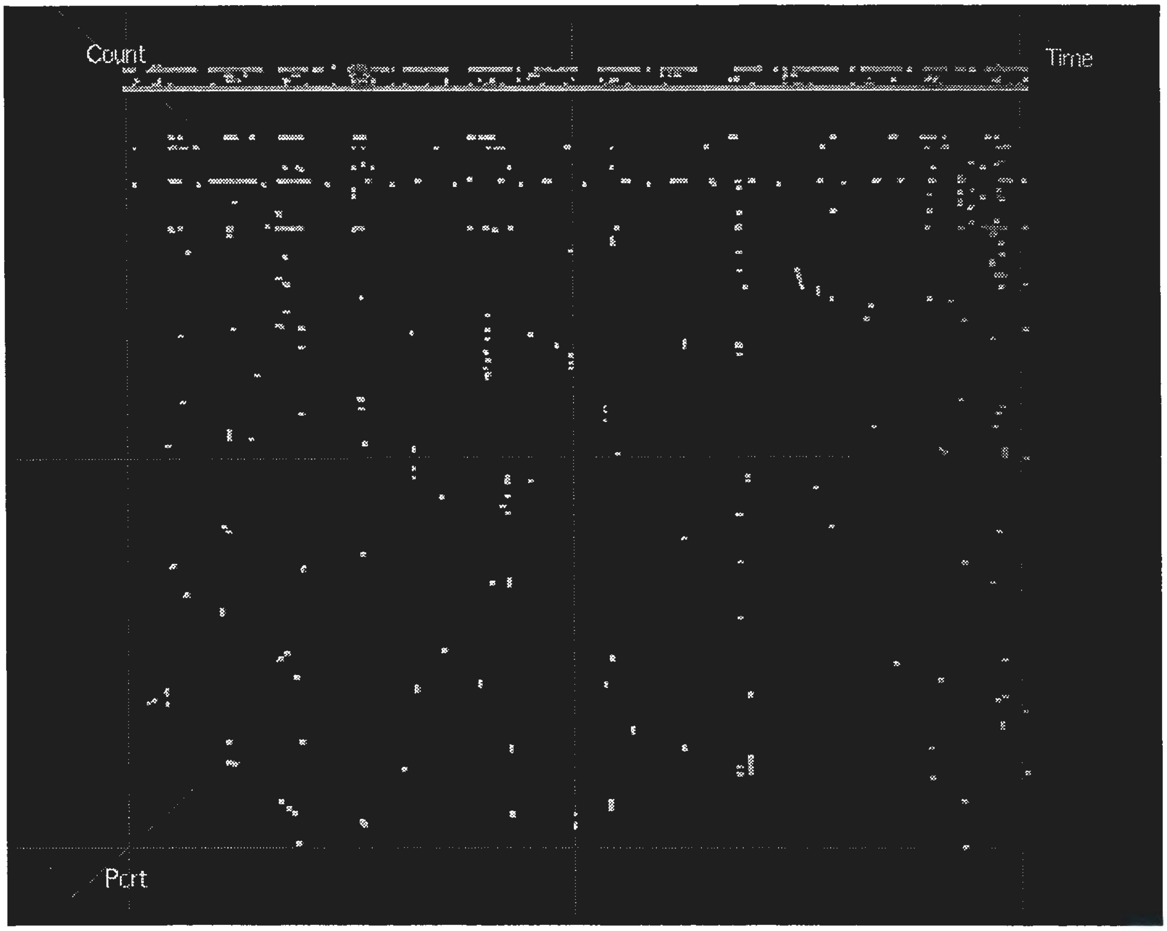

Figure 17. A two-week summary of TCP port activity showing only those connections in which an actual TCP 3-way handshake was observed. Time progresses from left to right. The vertical axis is a histogram of service ports. Only ports below 5000 are shown.

To test this hypothesis, we installed an SSH server on an uncommon port (port 789) then successfully connected to the server from several hosts both from within the local subnet of the host and from an external network. We plotted the results as shown in figure 17. This time, we applied a color scheme to each data point according the frequency of appearance of that server id. The rogue server that we installed clearly shows up in the plot. However, the plot also shows a high incidence of unique servers that were not part of the test. Out of 2012 collection intervals, 11 separate intervals contained MEDIUM-HIGH to HIGH values for the number of unique servers (2 or 3 unique server connections). Although it was difficult to verify the circumstance of every one of these higher values, it was determined through further inspection that at least 4 connections in addition to the staged attack were the result of unauthorzed connections. However, this also meant that there were potentially up to 6 false alarms, or approximately an 55 percent false alarm rate. This can be attributed to the fuzzy C-means clustering of the input data with sparsely distributed values for the number of unique servers. In this case, it is clear that adjusting the fuzzy membership functions manually or using a statistical mean-based fuzzy set quantifier would reduce the number of false positives in this result.

When we applied this fuzzy system to the data gathered when we installed the rogue server, we found that the rogue test server triggered the HIGH alert for Unauthorized Servers. To facilitate detection, it may be a good idea for the system to maintain a list of known legitimate known servers and exclude them from detection processing. However, the unauthorized server detection algorithm should not be impaired by not having such a list. Rather, the list would be helpful at focusing the detection on unknown servers, possibly reducing the number of false alarms.

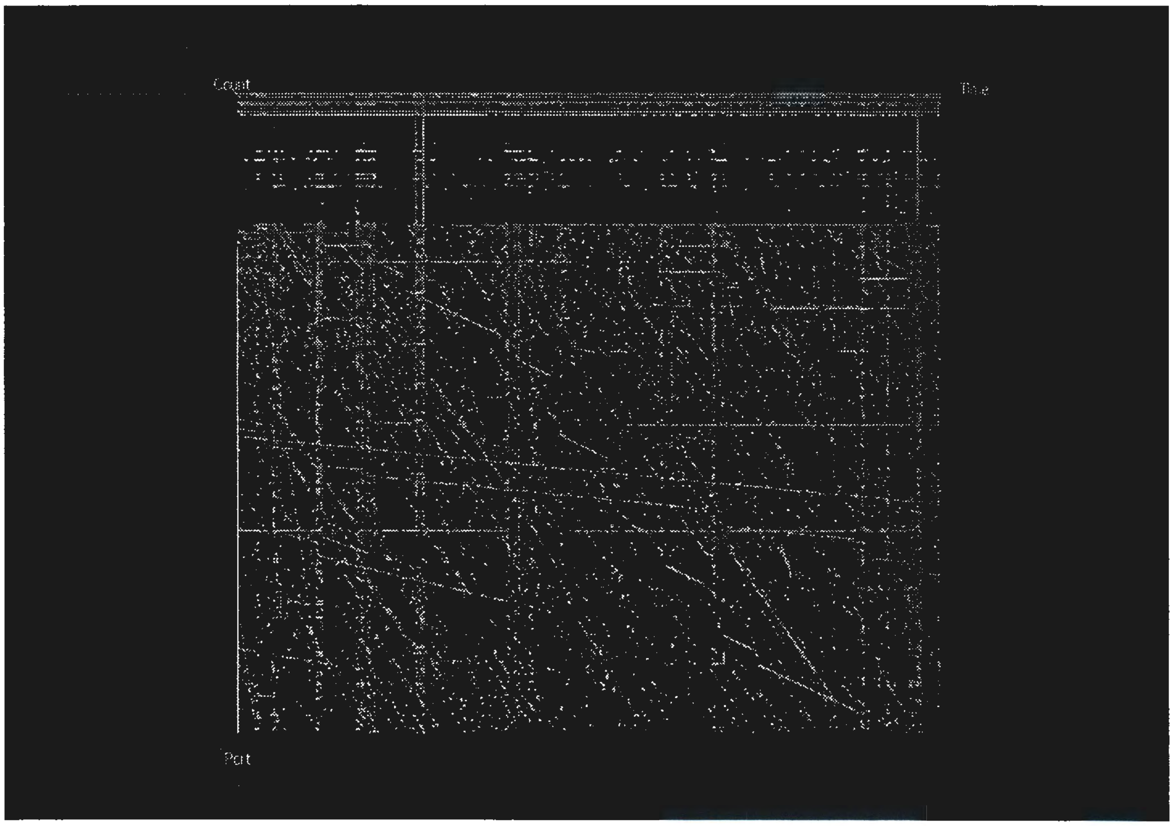

Figure 16. A two-week summary of on all TCP service ports. The time axis increases from left to right. The vertical axis is a histogram of TCP service ports. For clarity, only ports below 5000 are shown.

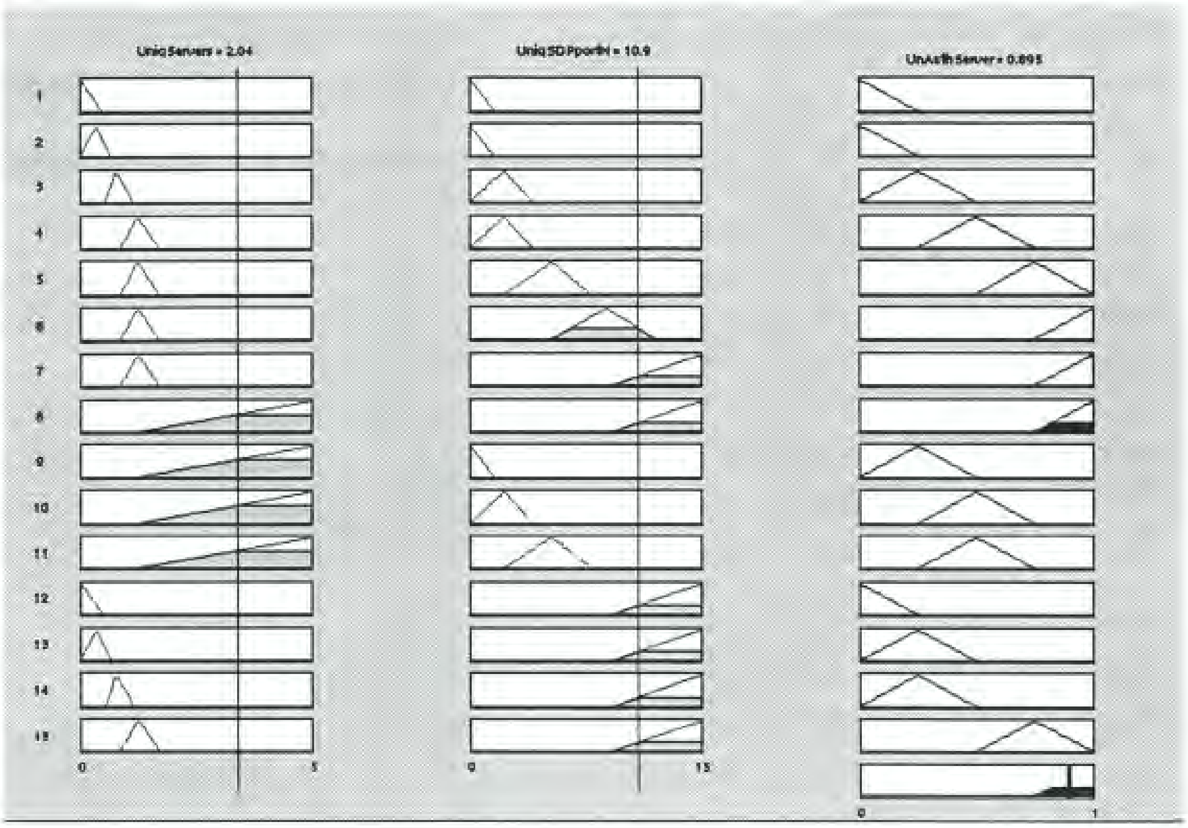

Figure 18. Fuzzy system for detecting unauthorized server use. The inputs are the number of unique servers observed and the number of unique SDPs for a given port. The output is the degree of alert for an unauthorized server.

CONCLUSIONS

It is clear that anomaly-based network intrusion detection can be a complex process. The shear variety in the network data stream, the amount of data to be processed, and the subtle and ever-changing ways that intruders penetrate systems all conspire to complicate the process. Nonetheless, this research has shown that there are several broad classes of intrusion behavior that can be detected with a general anomaly based approach. The metrics we have selected for this phase of the proof of concept will easily enable detection of various scan and flood activity. More specialized metrics, such as server connection activity, and incidence of various service ports on the network make it possible to observe more specific attacks. While this research does not solve the problem of finding all network-based invasions, we feel that the generality of fuzzy intrusion detection may hold great promise as a high-level intrusion detection scheme that works in conjunction with detailed misuse detection systems. Most importantly, this research has laid a solid groundwork for fuzzy intrusion detection and revealed promising areas of continued exploration.

The Denning intrusion detection model provides a sound theoretical basis for the discussion of the science. We have found that it is possible to produce a practical intrusion detection system that remains true to the overall framework of the Denning model. As Denning makes clear, the ultimate performance of the system depends greatly on the choice of data elements. Great care has been given to the selection of metrics and for techniques to store the data efficiently. Visual data mining has proven to be a valuable tool in helping understand the data being collected and what metrics are the most important to the intrusion detection scenarios at hand. Data aggregation and data reduction techniques have proved particularly important in exposing the most sensitive features in the input stream.

The fuzzy systems approach to the pattern matching aspect of the system has shown itself to be versatile and flexible. Fuzzy rules can be written for very general conditions in the data, or for very specific items. Also, the fuzzy systems can be made adaptive to the point that the security administrator does not need to set arbitrary thresholds for alert conditions. Rather, the geometry of the fuzzy member functions can obtained directly from the data.

While this research has proven very enlightening about the difficulties and challenges of anomaly-based intrusion detection, there are several areas that we will continue to explore as we endeavor to create a deployable system for FIRE. A short list of planned enhancements are listed below:

1. Continue to investigate different detection scenarios and the metrics necessary for detecting them.
2. Integrate the method for updating fuzzy membership function constraints directly into the *plotter* program instead of performing the analysis with Matlab.
3. Devise metrics for time signature characteristics in network packet data. Currently the *ndump* program ignores the length (in time) of a connection or the time between messages.

4. Develop host-based processing capabilities including login behavior and email activity.
5. Create a stateful processing capability for ICMP packets. Currently, *ndump* does not logically connect echo reply requests with echo replies.
6. Improve the handling of long-lived connections. There are several improvements planned to the strategy for identifying long-lived connections and properly tracking the TCP state of those connections.

In summary, we feel that this research shows that fuzzy intrusion detection can be deployed in a practical sense to help make the security operator more aware of the activity on a network. Visual data mining for intrusion detection is a new area that promises exciting new opportunities for increasing the security of computer systems. Together, fuzzy intrusion detection and visual data mining can provide powerful analysis capabilities to the security administrator.

REFERENCES

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

Bace, R. "An Introduction to Intrusion Detection & Assessment". ICSA, Inc.

Bezdek, J. C. *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum Press, 1981.

Buja, A. and Cook D. "Interactive High-Dimensional Data Visualization", Journal of Computational and Graphical Statistics. Pp78-99, 1996

Computer Emergency Response Team (CERT), "Results of the Distributed-Systems Intruder Tools Workshop", Pittsburgh, Pennsylvania, USA, November 2-4, 1999

Crosbie, M. and Spafford, E. "Active Defense of a Computer System using Autonomous Agents", Technical Report No. 95-008. Dept. of Computer Sciences, Purdue University, February 15, 1995

Crosbie, M. and Spafford, E. "Applying Genetic Programming to Intrusion Detection", COAST Laboratory. Department of Computer Sciences. Purdue University, 1995

Denning, D. "An Intrusion Detection Model", *IEEE Transactions on Software Engineering, Vol. SE-13. No. 2*, February 1987

Forrest, S., Hofmeyr, S. and Somayaji, A. "Computer Immunology". Communications of the ACM. Vol. 40, No. 10. October 1997

Frank, J. "Artificial Intelligence and Intrusion Detection: Current and Future Directions". Division of Computer Science, University of California, Davis. NSA URP MCA904-93-C-4085, June 9 1994

Internet Security Systems. "Network- vs. Host-based Intrusion Detection". Whitepaper. Internet Security Systems. Electronic source: http://solutions.iss.net/products/whitepapers/nvh_ids.pdf. Downloaded February, 2000

Kim, J. and Bently, P. "Negative Selection and Niching by an Artificial Immune System for Network Intrusion Detection". Electronic source: http://www.cs.ucl.ac.uk/staff/j.kim Downloaded February 2000

Kosko, B. *Neural Networks and Fuzzy Systems*. Englewood Cliffs: Prentice Hall, 1992

Kosko, B. *Fuzzy Engineering*. Upper Saddle River, NJ: Prentice Hall, 1996

Kumar, S. and Spafford, E. "An Application of Pattern Matching in Intrusion Detection". Technical Report CSD-TR-94-013. Purdue University, Department of Computer Sciences, June 17, 1994

Kumar, S. and Spafford, E. "A Software Architecture to support Misuse Intrusion Detection". Technical Report CSD-TR-95-009. Department of Computer Sciences, Purdue Univesity. March 17, 1995

Lane, T. and Brodley, C. "Temporal Sequence Learning and Data Reduction for Anomaly Detection", ACM Transactions on Information and System Security, Vol. 2, No. 3, August 1999, pages 295-331

Lee, W. and Stolfo, S. "Data Mining Approaches for Intrusion Detection". Electronic source: http://www.cs.columbia.edu/~sal/hpapers/USENIX/usenix.html. Downloaded January, 2000.

Levitt, K. et al. "Analysis of an Algorithm for Distributed Recognition and Accountability". Department of Computer Science. Univ. of California, Davis.

Lindqvist, U. and Jonsson, E. "How to Systematically Classify Computer Security Intrusions", Proceedings of the 1997 IEEE Symposium on Security and Privacy, pages 154-163, Oakland, California, USA, May 4-7, 1997. IEEE Computer Society Press.

Northcutt, S. Network Intrusion Detection – An Analyst's Handbook. Indianapolis IN: New Riders, 1999

Pao, Y.-H. Adaptive Pattern Recognition and Neural Networks. Reading MA: Addison-Wesley, 1989

Paxson, V. "Bro: A System for Detecting Network Intruders in Real-Time". Network Research Group, Lawrence Berkeley National Laboratory. LBNL-41197, January 14, 1998

Ptacek, T. and Newsham, T. "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection". Secure Networks , Inc., January 1998

Ranum, M. "Intrusion Detection and Network Forensics". Large Installation and System Administration '99, tutorial handbook, Seattle, Washington, November 7-12, 1999

Ruiu, D. "Cautionary Tales: Stealth Coordinated Attack HOWTO". Electronic source: http://www.nswc.navy.mil/ISSEC/CID/Stealth_Coordinated_Attack.html. Downloaded March, 2000

Zamboni, D. et al. "An Architecture for Intrusion Detection using Autonomous Agents". COAST Technical Report 98/05. COAST Laboratory. Purdue University. June 11, 1998