CERIAS Tech Report 2005-55

BEHAVIORAL FEATURE EXTRACTION FOR NETWORK ANOMALY DETECTION

by James P. Early

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

BEHAVIORAL FEATURE EXTRACTION FOR NETWORK ANOMALY DETECTION

A Thesis

Submitted to the Faculty

of

Purdue University

by

James P. Early

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2005

To my mother, Martha Williams, for teaching me to be inquisitive; and to my wife, Catharine, and daughter, Caroline, for their encouragement and unwavering support.

ACKNOWLEDGMENTS

Although the dissertation is considered the work of a solitary author, many others make contributions to the work in various ways. My case is no exception, and I would like to take this opportunity to acknowledge a few of these contributions.

I would first like to thank my co-advisors, Carla Brodley and Eugene Spafford. Professor Brodley introduced me to the machine learning techniques that are at the core of my work. Professor Spafford always displayed a willingness to share his expertise, provide guidance and support. For this, I am extremely grateful to both of them. I would also like to thank the remaining members of my examination committee: Mikhail Attalah and Elisa Bertino. They provided invaluable feedback for the work in this dissertation.

I am grateful for the support provided by the various CERIAS sponsors. I am also grateful to the Intel Corporation for their award of a Ph.D. Fellowship. In particular, the material in this dissertation is based upon work supported by the National Science Foundation under Grant No. 0335574, and the Air Force Research Lab under Grant No. F30602-02-2-0217.

Many of my colleagues at Purdue participated in discussions about the details of the work in this dissertation. These included fellow CERIAS graduate students Florian Buchholz, Benjamin Kuperman, Tom Daniels, Rajeev Gopalakrisna, Paul Williams, and Diego Zamboni. The graduate students of the Machine Learning Lab (Serdar Cabuk, Maja Pusara, and Xiaoli Zhang Fern) also provided valuable commentary. Although less formal, the many discussions held during "Systems Lunch" with Doug Comer, Dennis Brylow, Ethan Blanton, and Dan Ardelean were also very helpful.

The decision to return to graduate school in mid-life required careful consideration of many factors, and I would like to thank my friends Karen Johnson, Larry and

Betsy Toler, Tom and Ann Sloane, Mark and Sheila Peyrot, Whitney Sherman, and Martha Kennedy for their advice and encouragement. Our friends Bill and Shari Gass, and Gary and Cindy Goldberg helped make West Lafayette feel like home.

I would like to express my heartfelt thanks for the incredible amount of support I received from my family. My brothers and sisters-in-law, Stephen and Debra Early, and Timothy and Kristina Early were always there with a pat on the back or word of encouragement. Their senses of humor provided a welcome respite at times. The concern shown by my step-father, Michael Williams, is gratefully acknowledged.

And finally, but most importantly, I am fortunate to have been deeply influenced by three generations of Early women. My mother, Martha Williams, helped instill the importance of education and fostered my sense of curiosity and wonder. Her concern and prayers during challenging times were an abiding comfort. My wife, Catharine Early, encouraged me to return to graduate school. She willingly moved away from her home and friends to allow me to pursue my dream, and for that I am eternally grateful. Her support and understanding were unwavering during times when work seemed all-consuming. No one could ask for a better marriage partner. Finally, my daughter Caroline provided my inspiration. Witnessing her growth provides a daily reminder to strive to reach one's full potential. Words cannot express the depth of my appreciation to each of them.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Early, James P. Ph.D., Purdue University, August, 2005. Behavioral Feature Extraction for Network Anomaly Detection. Major Professors: Carla E. Brodley and Eugene H. Spafford.

This dissertation presents an analysis of the features of network traffic commonly used in network-based anomaly detection systems. It is an examination designed to identify how the selection of a particular protocol attribute affects performance. It presents a guide for making judicious selections of features for building network-based anomaly detection models.

We introduce a protocol analysis methodology called *Inter-flow versus Intra-flow Analysis* (IVIA) for partitioning protocol attributes based on operational behavior. The method aids in the construction of flow models and identifies the protocol attributes that contribute to model accuracy, and those that are likely to generate false positive alerts, when used as features for network anomaly detection models.

We introduce a set of data preprocessing operations that transform these previously identified "noisy" attributes into useful features for anomaly detection. We refer to these as *behavioral features*. The derivation of this new class of features from observed measurements is both possible and feasible without undue computational effort, and can therefore keep pace with network traffic.

Empirical results using unsupervised learning show that models based on behavioral features can achieve higher classification accuracies with markedly lower false positive rates than their traditional packet header feature counterparts. Behavioral features are also used in the context of supervised learning to build classifiers of server application flow behavior.

1 INTRODUCTION

1.1 Problem Statement

Computer systems in highly connected networks, such as the Internet, exist in a hostile landscape. The number of annually reported system and application vulnerabilities has grown exponentially since the first year of collection in 1995 [1]. The number of reported vulnerabilities in 2005 is on pace to reach nearly 5,000, representing a 29% increase over 2004 and a nearly 30-fold increase over 1995. Computer crime survey statistics [2] indicate that the leading sources of financial loss (by parties both external and internal to an organization) are computer viruses, denial of service attacks, and theft of proprietary information. This increased vulnerability is accompanied by explosive growth in the number of Internet hosts [3] and users [4]. The combination of increasing numbers of highly connected systems with increasingly more vulnerabilities amplifies the number and impact of individual attacks.

The challenge for users is to keep pace with this constantly growing number of new attacks. Researchers have tried to meet this challenge by developing a variety of techniques, tools, and products to understand attacks and mitigate their effect. However, the majority of these solutions require some *a priori* knowledge of the attack to be effective [5]. This means that the attack must first appear "in the wild" before it can be dealt with on a wide scale. Such attacks have traditionally been referred to as *intrusions*, and systems designed to monitor for their presence are called *intrusion detection systems* [6].

The use of anomaly detection techniques in the context of network intrusion detection has been touted as a promising method of identifying and understanding novel attack behaviors. However, despite many years of research and many proposals, there is no consensus on how anomaly detection should be performed on network

traffic. Further, concerns about the quality of training data [7] and the high false positive rates exhibited by many deployed systems [8] has led some to conclude that network based anomaly detection is impractical.

In many published articles, the focus has been the analysis of values carried in packet headers [9–15]. It has been reasoned that anomalous combinations of these values with respect to a training set can be considered malicious, or at least suspect. However, as will be seen in Section 3.2, this claim cannot be supported because of the vast space of normal values associated with even the simplest of protocols. When anomalous combinations of packet header values occur, it is far more likely to be a normal case excluded from the training data than a malicious event. This observation indicates a need for a more comprehensive understanding of the features used to build anomaly detection models for network intrusion detection.

This dissertation presents an analysis of features that have been used in previous work focusing on network intrusion detection. We discuss the characteristics of useful features, and equally important, those of undesirable features. We then show how these undesirable features can be processed to produce features that yield practical anomaly detection models.

1.2 Basic Concepts

We now introduce some basic concepts that will be referenced throughout the remainder of this dissertation.

1.2.1 Intrusion Detection

Attacks launched against a host or network have traditionally been referred to as *intrusions*. The definition provided in [16] is as follows:

Intrusion – Any set of actions that attempt to compromise the integrity, confidentiality, or availability of a computer resource

Individuals deemed *intruders* include both those without appropriate authorization to use the resource and individuals abusing their authorization (i.e., "insiders") [17]. We will use the terms *intrusion* and *attack* interchangeably throughout this dissertation.

The definitions we will use for intrusion detection and an intrusion detection system are presented in [18]:

Intrusion Detection – The problem of identifying an attempt to compromise the integrity, confidentiality, or availability of a computer resource

Intrusion Detection System – A computer system (possibly a combination of software and hardware) that attempts to perform intrusion detection

It should be noted that intrusion detection is generally decoupled from intrusion *response*. Typically, an intrusion detection system merely alerts an operator of the presence of an attack condition. The operator then determines the appropriate response.

1.2.2 Desirable Characteristics of an Intrusion Detection System

The following characteristics of an ideal intrusion detection system are presented by Zamboni [18] and represent refinements of an earlier list provided by Crosbie and Spafford [19]:

1. It must run continually with minimal human supervision.
2. It must be fault tolerant:
   (a) The intrusion detection system must be able to recover from system crashes, either accidental or caused by malicious activity.
   (b) After a crash, the intrusion detection system must be able to recover its previous state and resume its operation unaffected.

3. It must resist subversion:

   (a) There must be a significant difficulty for an attacker to disable or modify the intrusion detection system.

   (b) The intrusion detection system must be able to monitor itself and detect if it has been modified by an attacker.

4. It must impose a minimal overhead on the systems where it runs to avoid interfering with their normal operation.

5. It must be configurable to accurately implement the security policies of the systems that are being monitored.

6. It must be easy to deploy. This can be achieved through portability to different architectures and operating systems, through simple installation mechanisms, and by being easy to use and understand by the operator.

7. It must be adaptable to changes in system and user behavior over time. For example, new applications being installed, users changing from one activity to another, or new resources being available can cause changes in system use patterns.

8. It must be able to detect attacks:

   (a) The intrusion detection system must not flag any legitimate activity as an attack (false positives).

   (b) The intrusion detection system must not fail to flag any real attacks as such (false negatives). It must be difficult for an attacker to mask his actions to avoid detection.

   (c) The intrusion detection system must report intrusions as soon as possible after they occur.

   (d) The intrusion detection system must be general enough to detect different types of attacks.

These characteristics will be used throughout this dissertation as a basis for comparison among various approaches to intrusion detection.

1.2.3 Intrusion Detection Methodologies

Intrusion detection systems are often classified based on the source of data used for analysis and the techniques used in building behavior models [20, 21].

Data Sources

A common distinction is made between systems that employ host data (e.g. log files, process metrics, and file system activity) and data obtained by observing network traffic (e.g., packet characteristics and bandwidth allocation) [17]. The selection of a particular method is guided by the needs of the operator. Host-based systems have the advantage of being able to interpret information relevant to the attack directly on the intended victim [22]. Additionally, host-based systems permit monitoring for local attacks. Network-based systems have the advantage of visibility of potentially many hosts and events. The distinction can be blurred by systems that employ *agents* or *detectors* that collect and analyze both types of data. Indeed, a number of systems have been developed employing both techniques [23, 24].

Model Types

Another common distinction is made between systems that employ *misuse detection* versus *anomaly detection* analysis components. This distinction stems from the approaches used to build each component and the resulting alerts they generate. Misuse detection systems are built on knowledge of known attack behaviors. Alerts are generated when these attack behaviors (i.e., signatures) are identified within current behavior. A misuse detection system must be periodically updated with new signatures to identify new attack types. In contrast, anomaly detection systems are

built on knowledge of normal behavior, and alerts are generated when deviations from normal behavior occur [25]. The principle use of anomaly detection methods in intrusion detection is to identify novel attacks with no existing attack signature.

This dissertation is focused on network-based anomaly detection techniques for intrusion detection.

1.2.4 Network-Based Anomaly Detection Design Challenges

Anomaly detection systems are initially *trained* with a collection of examples of normal data. This data is referred to as the *training set*. Once trained, the system examines current behavior to identify instances that deviate from its model of normal behavior. An important design criteria is the choice of *features* used to model normal behavior. Features provide semantics for the values in the data. For example, a feature can be the number of host connections or a value in a packet header. Feature extraction is important because it directly effects the accuracy and utility of the anomaly detection system. Good features provide strong correlation between anomalous behavior and malicious activity, whereas poorly selected features can result in irrelevant anomalies and high false positive rates.

In addition to good feature selection, a designer of a network-based anomaly detection system must also be concerned with the amount and quality of training data. Data used in training should be representative of the target being modeled. In addition, compiling training data is both a time consuming and human intensive (and therefore costly) task. Thus, a design tradeoff must be made between adequate training data and available resources.

1.2.5 Problems with Network-based Anomaly Detection

Network-based anomaly detection methods have held particular promise since the initial idea was proposed in the 1980s [6]. If reasonably good models of normal network behavior could be constructed, a single anomaly detection system could

monitor many computer systems for new attacks. The search for these models has been the focus of anomaly detection research for nearly two decades. Methods have been proposed using statistical models [23], graph-based models [26], neural networks [27, 28], decision trees [29], Markov models [30], and principle component analysis [31]. These methods use attributes of network traffic (packet headers, payload, etc.) as features. The definitions of what is normal and anomalous are based on these features.

Although many have shown promise, no modeling method is acknowledged to be singularly effective for network intrusion detection. Some might require off-line analysis, meaning that the detection of intrusions in real time is not possible. This represents a failure to meet desirable characteristic #8c. Others might require training times or data set sizes that are deemed prohibitive in terms of human effort – a failure of characteristics #5 and #6. Still others suffer from what are perceived as low detection accuracies (failure of #8b). However, what has most frequently emerged as a severe limiting factor in the usefulness of these techniques is the number of false positive alerts generated, namely the incorrect labeling of benign network events as attacks [8]. This is of particular concern because such errors negatively impact the time an administrator might otherwise spend investigating actual attacks. Indeed, investigation into techniques to reduce the false positive rate for network-based anomaly detection is a major area of research [25, 32–34].

McHugh [35] reported that problems continue with the use of systems developed in lab environments when applied to real world environments. These problems range from high false positive rates to what McHugh calls "serendipitous" detections – fortunate coincidences of certain network attribute values that result in a detection, but are not true indications of malicious behavior.

Bellovin proposes a solution to the problem of discriminating anomalous events from malicious events in a humorous RFC [36] issued on April Fool's Day, 2003. He writes:

> Firewalls, packet filters, intrusion detection systems, and the like often have difficulty distinguishing between packets that have malicious intent and those that are merely unusual. We define a security flag in the IPv4 header as a means of distinguishing the two cases.

The proposal details the use of a single binary attribute indicating the "evil intent" of the packet (e.g., 0=not evil, 1=evil). Such an attribute would greatly simplify the work of network security administration, but, of course, no such assistance can be expected in the real world. Anomaly detection in the real world requires understanding protocol behavior to design an effective, practical system.

We do not conclude that all of the previous modeling methods are inadequate for network intrusion detection. On the contrary, some of these techniques have proven to be effective in other domains. What is common to these methods is the use of basic packet header values as features. When examining a particular feature in isolation, one might ask if there is a notion of a normal or anomalous value for the feature. Further, one might ask if an anomalous value is linked with attack behavior. Such distinctions can be made for some packet header values, but no clear distinction can be made for many others. If a given value for a feature is ambiguous, that ambiguity is transferred into the behavior model. The resulting model will be limited in its ability to distinguish normal network events from attack events.

1.3 Thesis Statement

This dissertation presents evidence to show the validity of of the following hypotheses:

1. It is possible to identify *a priori* attributes of network protocols that will contribute to high false positive rates in anomaly detection models when used as features.

2. It is possible and feasible to process the values associated with these previously identified (i.e., noisy) attributes to extract useful features. Models built using

these features exhibit reduced false positive rates and higher true detection rates than models that directly employ packet header values.

3. The newly extracted features enable behavioral modeling of application traffic that is not possible when using packet header values.

1.4 Document Organization

In this dissertation, we examine the issues involved in selecting a feature set for network-based anomaly detection. Related work will be presented in Chapter 2. Chapter 3 presents a novel method of feature extraction that first uses protocol analysis to identify basic packet features that introduce ambiguity, then transforms them into useful features for anomaly detection. Chapter 4 presents empirical results using this feature extraction method in the context of unsupervised learning. In Chapter 5, we present empirical results of the use of supervised learning and our feature extraction method to perform server application profiling. Finally, Chapter 6 presents conclusions, a summary of the contributions of this dissertation, and an outline of future work.

2 RELATED WORK

In this chapter, we discuss the application and evolution of anomaly detection techniques to computer security. We outline the techniques used to build anomaly detection models and present issues that have arisen from several notable implementations. Finally, we include a discussion of the general success of these techniques and their applicability in real world contexts.

2.1 Feature Extraction

2.1.1 Attributes and Features

In this section, we introduce terms used throughout the remainder of this dissertation. We will use the term *attribute* to refer to a particular characteristic of an entity (e.g., computer system, process, network packet) being measured. Examples of attributes include available disk space, CPU utilization, and network packet header fields. We will use the term *feature* to mean a logical construction of attributes for the purpose of building models of behavior. A feature may relate directly to a single attribute, such as the Protocol field in an IP packet header. A feature may also be composed of multiple attributes, or constitute the observation of a particular attribute over time.

Features establish the analytical context for anomaly detection. The purpose of an anomaly detection system is, quite simply, to identify "unusual" patterns in the data. However, it can only do so within the context of the given feature set. Anomaly detection systems do not have the ability to ask for more information beyond the context that is given. Therefore, the careful creation and selection of features is critical to the accuracy and utility of an anomaly detection system.

A feature is said to be *relevant* if it contributes to the detection of the target concept [37]. In our case, the target concept is the identification of computer intrusions. Features that are not relevant are considered *irrelevant*. It is desirable to avoid the use of irrelevant features because they can introduce inaccuracy and a propensity for false positives into anomaly detection models.

2.1.2 Early Models and Feature Sets

Early work in the use of anomaly detection for the purposes of detecting intrusions could be classified as *host-based* anomaly detection, because the measured attributes were taken from within the running system. In [38], Anderson introduced the concept of using anomaly detection to identify incidents of unauthorized usage on a particular type of IBM system. He reasoned that patterns of "normal" behavior could be defined by measuring various system attributes. He describes a user surveillance system that records information about issued jobs, parameters, resources used, and other attributes of requested operations. Statistical analysis is then performed to determine the range of normal behavior for a class of jobs (represented by the mean and standard deviation). The surveillance system then issues alerts when attributes of a given job fall outside this range. Such methods have been used throughout the evolution of later anomaly detection methods.

In [6], Denning formalized the concept and presented general guidelines for audit records (attributes), behavior profiles, and anomaly records. The author defines three classes of features [1] to be used in behavior profiles:

- *Event Counter:* the number of audit records satisfying some property occurring during a time period. An example would be the number of login attempts during an hour.

- *Interval Timer:* the length of time between two related audit records. An example is the length of time between successive logins into an account.

[1] The author uses the term *Metrics*.

- *Resource Measure:* the quantity of resources consumed by some action within a the context of particular audit record. Units of this feature might be expressed in CPU cycles, I/O records, or pages printed. This feature may be viewed as the *cost* of performing a particular action.

Using these definitions, the author then presents sixteen manually-generated features designed to indicate particular session, program, and file system activity. These features included login frequency, execution frequency, and file records read/written.

In addition, Denning introduced a collection of statistical models utilizing the above features intended for building user profiles. We will refer to these models in discussing the analysis methods of other related work.

- *Operational Model:* based on a fixed empirical upper and/or lower operational threshold. An observation is considered anomalous if it falls outside of this threshold.

- *Mean and Standard Deviation Model:* based on the historical mean and standard deviation of an attribute. An observation is considered anomalous if it falls outside a predefined confidence interval.

- *Multivariate Model:* similar to the mean standard deviation model except that it is based on correlations among two or more features.

- *Markov Process Model:* represents each audit record as a state variable and uses a state transition matrix to characterize the relative transition frequencies between states. An new observation (state) is considered anomalous if the probability of the transition from the previous state falls below some predefined threshold.

- *Time Series Model:* uses historical order and inter-arrival times of audit records to estimate the probability of occurrence for the next audit record. Similar to the Markov model, an observation is considered anomalous if the probability of this new event falls below some predefined threshold.

These features and models have been widely used in a variety of anomaly detection systems [15, 23, 39, 40]. However, two open questions raised in Denning's conclusion are fundamental to whether anomaly detection is applicable for identifying intrusions:

> *Soundness of Approach* - Does the approach actually detect intrusions? Is it possible to distinguish anomalies related to intrusions from those related to other factors?
>
> *Choice of Metrics, Statistical Models, and Profiles* - What metrics, models, and profiles provide the best discriminating power? Which is cost-effective? What are the relationships between certain types of anomalies and different methods of intrusion?

Both of these questions relate to the types of system attributes, and ultimately, the features used for anomaly detection. These questions can be rephrased as: What should be measured to identify intrusions with high accuracy? Given that the creation of features in this context is a manual one, how do we know that we have an adequate feature set? Choosing a feature with good predictive power for inclusion in a behavior model is as important as avoiding the choice of a likely irrelevant feature. Indeed, it will be shown in future sections that the choice of features is critical to the accuracy and utility of anomaly detection techniques in intrusion detection.

The computing landscape has changed greatly since the pioneering works of Anderson and Denning were presented. At that time, mainframe computing was the norm. User activity was centralized, and thus wide-scale monitoring of system attributes was feasible. Features were created based on easily observable system attributes and human experience. Since then, computing has become much more distributed – both logically and physically. The tremendous growth in the use of a wide range of Internet protocols has greatly increased the potential for intrusive activity.

The feature sets proposed by Anderson and Denning were manually crafted and consisted of system attributes available through computer auditing mechanisms of the day. However, wide scale network connectivity (and its associated challenges) would soon mean that anomaly detection models and their underlying feature sets would have to evolve to meet the challenges of this new environment.

2.2 Host-Based Anomaly Detection

2.2.1 Manual Feature Extraction

Before the era of wide-spread network connectivity, the focus for intrusion detection was single multi-user systems. Denning [6] outlined how the monitoring of various program execution attributes could be used to detect unusual activity. An early implementation of these concepts was found in IDES, the Intrusion Detection Expert System [41, 42]. This work was subsequently followed by NIDES, the Next-generation Intrusion Detection Expert System [43].

These systems used the profiles concept described in Denning's work [6] to create profiles for each user on the system. A variety of attributes is recorded including the CPU utilization for a particular command and the name of the file accessed by the command. Additional attributes reflect the level of user activity over one, five, and ten minute windows. The profiles also adapt over time to accommodate changes in behavior.

IDES and NIDES were developed on different platforms, and thus they use different data sources for collecting attribute values. IDES was developed for DEC systems and used data from TOPS20 logs [44]. NIDES was developed for SunOS systems and gets its information from the Basic Security Module (BSM) and other accounting generally available in UNIX systems [45].

USTAT [39], the Unix State Transition Analysis Tool, employed the state transition model to classify user behavior. Like NIDES, USTAT was also built on the SunOS platform and used a C2 auditing mechanism (BSM) as its data source. This

data is then processed to form an audit record in the form of a [*Subject, Action, Object*] triplet. A knowledge base of objects with various classification levels is maintained to identify access outside the normal scope of the operation. For example, there is a class of "restricted" file objects, meaning that they should only be read by a program – not a user. The state transition analysis is performed in an *inference engine*. The transitions used in analysis are the actions taken by the user. Such an action might be writing to a particular file. If an action results in the user entering a state outside of the set of appropriate or expected states, such as when a user modifies a file marked as executable, the system triggers an alarm.

Other host-based systems appearing in early 1990s were Wisdom and Sense [46] and SECURENET [27]. Wisdom and Sense employed a rule-based system of classifying user behavior. SECURENET employed artificial neural networks for classification. Both development efforts had ceased by the mid 1990s.

File system state has been used as a basis for building models of intrusion detection, and a number of tools have been developed to monitor a collection of files for changes in file contents [47–49]. Among the most popular file integrity checkers is Tripwire®. Tripwire[2] is used to create a database of cryptographic hashes (message digests) called *file signatures* [50]. With this database in a secure location (usually a write protected medium), Tripwire interrogates files listed in its database and compares a file's current signature to its associated "normal" database entry. The signature properties make it highly unlikely that a malicious user could alter a file to produce a given signature value. Further, the space of available signature values is large (2^{128} possible values for MD5 message digests [51]), making it highly unlikely that two different files would share the same signature. Therefore, Tripwire is able to confidently report a file modification (an anomaly) when signatures do not match. The Seurat [52] system uses patterns of file system updates across hosts to develop normal usage patterns and detect deviations. Typically, a file system monitor is an application run on a periodic basis - perhaps once per day. This means that if a

[2]Tripwire is a registered trademark of Tripwire, Inc.

file was modified during the interval between runs, then again modified back to the original, the modification would go unnoticed. Also, the detection scope of such a system is limited to attacks with file system artifacts.

The manual process of feature creation was continued through each of these projects, each time driven by the content of available data sources, operating platform, and operator experience.

2.2.2 Machine Learning Methods

Teng, et al. [53] applied time-based inductive learning to sequences of events in security audit logs with the goal of identifying intruders. A Time-based Inductive Machine (TIM) [54] was used to generate sequence rules of the form:

$$E1 - E2 - E3 \Longrightarrow (E4 = 0.95; E5 = 0.05)$$

This rule states that if event $E1$ is followed by event $E2$, and $E2$ is followed by $E3$, the probabilities of the next event being either $E4$ or $E5$ are 0.95 and 0.05, respectively. The system relies on a human expert to determine the set of rules that define user profiles. New sequences that do not match previous patterns are considered "unrecognized activities" and must either be investigated by the operator, or possibly used to generate additional rules. Preliminary results were presented for a mainframe system with approximately sixty users. The authors concluded from the results that useful patterns describing use behavior were obtained. However, there were no reported results of false positives or evidence that intruders were identified. It is not clear that a system reliant on a user to manually build appropriate rule sets could scale to an environment where there are many more users, programs, or alert types.

In [55] and [56], Maloof and Michalski describe the use the AQ15c incremental rule learner to perform classification of user profiles and feature selection. The initial feature set was chosen from data available from the UNIX *acctcom* utility. This set contained average, maximum, and minimum values of seven system process

attributes – real time, CPU time, user time, characters transferred, blocks read and written, the CPU factor, and the hog factor. The entropy measure [57] and PROMISE score [58] were then applied to each feature, and those falling below a predefined threshold were discarded. Thirteen features remained and were used for classification. The best performing rule set predicted a user with an accuracy of 96%. The authors discussed the completeness of their feature set in the context of a practical intrusion detection system and indicated that more features describing terminal names and time-of-day usage might be useful. However, their method does show the utility of using machine learning to eliminate irrelevant features.

Platform-dependent properties form the feature set of [59]. The authors used current and historical measurements of over 200 system properties of a host running the Windows 2000 operating system (made available through the *perfmon* performance monitor) to build profiles of user behavior. The resulting feature set consisted of approximately 1500 measurements. A weighted voting algorithm was then used to identify the features that most accurately contributed to user classification. Among the most heavily weighted features were the numbers of print jobs and login attempts.

A number of systems have been based on models built using sequences of executable operations. The seminal work by Forrest, et al. [60] examining sequences of UNIX system calls has spawned a number of projects presenting various refinements of this technique. These include the use of pattern discovery techniques [61], system call policies [62], system call arguments [63], and the augmentation of system call information with information from the program execution stack, such as program counters [64] and return addresses [65]. The work by Lane and Brodley [66, 67] and Oka, et al. [68] present a logical extension of this idea by examining sequences of UNIX commands to build user behavior profiles. All of these techniques incorporate tuning parameters, such as sequence length and smoothing factors, that must be manually adjusted. Discussions of the challenges involved in setting these parameters in practice are presented in [69] and [70].

Feature extraction in the host-based anomaly detection domain has been an evolutionary process. Early systems were entirely dependent on expert knowledge because of their exclusive use of manually extracted features. The introduction of machine learning techniques into the process helped in building models and identifying relevant versus irrelevant features, but the techniques still required a human expert to craft an initial feature set. Systems built on sequences of executable events require the least domain knowledge because they are based on abstract event sequences and can therefore be applied to different platforms. It is this development that provides clues about how feature extraction in the network anomaly detection domain might be applied.

2.3 Network-Based Anomaly Detection

The proliferation of internetworking technologies in the mid-1990s brought about the ramping up of research into network-based anomaly detection. Network-based anomaly detection systems differ from their host-based counterparts in the source of data used for analysis. Whereas host-based systems use audit data supplied by the system being monitored, network-based systems monitor attributes of network traffic among many hosts. Network-based systems give up the visibility of certain details of user behavior only available to the host, but they gain the ability to examine aggregate patterns of system and user behavior.

2.3.1 Manual Feature Extraction

One of the early implementations of a network-based anomaly detection system was NSM, the Network Security Monitor [40]. NSM is actually a set of different tools to collect network data, analyze it, and provide alerts. The features used by NSM are the number of packets and the total amount of data contained in these packets within a set of connections. Connections are modeled as the source and destination IDs, the type of service, and a connection ID. The analysis component

uses a statistical model to build an historical profile that is then used to classify anomalous activity.

Begun in 1993, the NADIR [71] project focused on the design of a system that operates in a classified environment. This system was used at the Los Alamos National Laboratory to detect unauthorized access to sensitive information, both from within the network and externally, on their Integrated Computing Network. It uses an expert system based anomaly detection engine to compile access profiles for approximately 9000 users. In addition, the system also creates access models for individual host workstations (roughly 10,000), and five Cray supercomputers. There is an active component of the system that probes areas of the network looking for vulnerabilities.

In a subsequent progress report [72] on the use of the system, several operational assessments were given. First, the system is said to have "increased the *perceived* odds of being caught." It is also reported that the system generates a sufficiently low number of false positives as to be manageable by one half-time staff person. The technical details supporting these claims were not provided, presumably because of the classified nature of the environment. However, it was disclosed that the expert system does not utilize self-learning because of concerns about the "potential weakness of the method." The process of initially classifying the users and subsequent re-classification is done manually.

Heady, et al. [16] describes a prototype system specifically designed to identify worm propagation behaviors. Its implementation was described in [73]. The system is instrumented to examine the Ethernet source and destination addresses, packet sizes, and time of connections. The anomaly detection component is based on a set of rules describing a statistical profile of network activity. Features used by these rules include the the number of packets per minute and first order statistics of packet sizes.

The Graph-Based Intrusion Detection System [26], or GrIDS, used graphs of connection activity as features. Nodes in the graph represent hosts, whereas edges are weighted with the number of TCP connections between hosts. The system was

designed to identify sweep and worm attacks that show distinct patterns of propagation through a network. A new observation (graph) is considered to be anomalous if it matches propagation patterns of known attacks.

The NetSTAT [74] system can be thought of as the network compliment to USTAT [39]. Both employ state transition analysis to determine the presence of anomalous events, but, of course, NetSTAT observes network traffic. The NetSTAT architecture consists of a network fact base, state transition database, and a collection of supporting tools. The network fact base contains information about the physical topology of the network being monitored. This aids the human operator in tailoring the analysis to the needs of a given network. The state transition database contains the representations of attacks that the system is trying to detect. This architecture also uses probes to distribute the task of monitoring for different attack transitions throughout the network.

A recent addition to the collection of network-based systems is Spade [15]. Spade is an anomaly detection plug-in for Snort – a popular open-source signature based IDS. Spade utilizes the operational model outlined by Denning [6], namely the operator specifies absolute limits on the ranges of normal behavior for a given attribute. The traffic features used by Spade are the source and destination IP address, transport protocol, and port numbers. In essence, these thresholds define what percentage of previously unseen connections will generate an alarm. Because these thresholds must be manually set, the operator must use a "tuning" period to find a threshold that is an acceptable compromise between attack events and false alarms. However, such a method could result in the operator setting the thresholds sufficiently high as to mask true attacks.

As with initial host-based systems, initial feature extraction used in these projects continued to be a manual process. The availability of easily measurable attributes of network traffic formed the basis of these features. However, the fact that an attribute can be easily measured does not mean that it should form the basis of a relevant feature for intrusion detection. In addition, the increase in available attributes has

brought about the inclusion of likely irrelevant features. In Spade, for example, the inclusion of the source IP is dubious – it is unlikely that any meaningful correlation can be drawn between an attack and the address where it originates. If there were such a correlation, anomaly detection would be unnecessary – all traffic from the offending source could be blocked.

In the next section, we examine some attempts to systematically generate features and models of network behavior using machine learning techniques.

2.3.2 Applications of Machine Learning

A number of network-based anomaly detection systems have been based on artificial neural networks [75]. Some systems were designed to identify particular attacks (e.g., UDP Flood [28], SYN Flood and SATAN [76]), while others attempt more general attack classifications. It is not surprising that systems designed to recognize particular attacks performed well (high detection rates with low false positives). However, results for the general systems were mixed. A keyword selection system described by Lippmann and Cunningham [77] was shown to identify novel attacks embedded in Telnet streams, while a system examining host connection activity [78] had a 76% false positive rate when using a model built with many attack types (performance improved for individual attack models). Despite some success, there are a number of practical disadvantages of building models with neural networks. These include the requirement of large amounts of training data (on the order of thousands of attack instances), long training times (often many times longer than other supervised learning methods), and the relative inability of humans to comprehend neural network models (i.e., their "black box" nature) as compared to other types of supervised learning [76].

Lee and Stolfo created a data mining framework for feature selection [79–81]. They applied a rule learner (with the rules effectively forming a decision tree) to identify patterns of both normal and attack behaviors in system call traces and net-

work traffic. The RIPPER [82] algorithm was used to build a set of rules describing associations of attribute values and frequent episodes [83] of events. The features contained in these rules, it was reasoned, represented the relevant features to identify intrusions.

The algorithm chosen by the authors performs *supervised learning*, namely that the learner must be trained using examples of both normal and abnormal (in this case, attack) behaviors. Training data must contain a sufficiently representative sample of both normal and attack behaviors for supervised learning to be effective. A criticism of the use of supervised learning for intrusion detection is that it is generally not feasible to expect that examples of all known attack behaviors can be collected.

However, even if we assume that an adequate training set was used, this process did not eliminate the need for manual intervention. The method operated in the absence of domain knowledge about protocol attributes and thus frequently generated irrelevant rules that required manual pruning. A first attempt to counter this was the designation of essential attributes (called "axis attributes" by the authors) that were required to be present in all rules. While reducing the number of irrelevant rules, the process also removed some attributes that may have proven useful in identifying intrusions. To solve this problem, the authors manually augmented the feature set. One such example can be found in [80] where a "land" feature is added to indicate the presence of a potential Land attack [84]. The authors considered "correlations among non-axis attributes as not interesting" [5]. However, we present rules in Section 4.6 that indicate strong correlations between so called non-axis attributes and attack behaviors. Their analysis focused on TCP traffic (utilizing many of the TCP state flags as features), and it is unclear how effective this method would be in the context of stateless protocols (e.g., UDP and ICMP).

The most serious criticism of this work deals with the required amount and content of the training data. Features are ultimately decided by the frequency of related attributes in the training data. Thus, unless we can be assured that our

training set contains adequate normal/abnormal examples, we cannot be assured that the resulting feature set can be used to identify intrusions. Discussing the applicability of this approach in [79], the authors note:

> There are also much needed improvements to our current approach. First, deciding upon the right set of features is difficult and time consuming. For example, many trials were attempted before we came up with the current set of features and time intervals.

Thus, rather than eliminating the manual creation of features, human effort has merely been shifted to the pruning of irrelevant rules, feature set augmentation, and preparation of an adequate training set. The latter likely represents significantly more effort on the part of a human operator. However, it is not clear that the additional effort produces features that predict intrusions with higher accuracy [14].

Luo and Bridges presented a refinement to this method in [85]. It describes the use of "fuzzy" association rules and frequent episodes in an attempt to produce rule sets that are more general than those described in [79]. However, the authors only applied their analysis to TCP data and selected only four features[3] to describe stream behavior. The results indicated that two simulated attacks – one involving IP spoofing and the other a TCP port scan – deviated significantly enough from a baseline rule set as to be considered anomalous. It is also unclear in this case whether the technique is applicable to stateless protocols.

The Packet Header Anomaly Detector, or PHAD [10], is another example of the use of machine learning techniques to identify intrusions, this time using attributes of network packets. An anomalous packet was indicated by its "rarity" in the training data. The implicit assumption is that packets that occur rarely are likely to be intrusions.

Training and testing of this system were performed using the 1998 MIT Lincoln Laboratory Intrusion Detection Evaluation Data Sets [86] – a collection of synthetic

[3]the number of TCP SYN, FIN, RST flags and different destination TCP ports in a 2 second window

data used in the 1998 DARPA IDS Evaluation exercise. A notable outcome of testing was the realization that the inclusion of the IP TTL (Time to Live) field in the feature set increased intrusion detection rates by more than 30%. The implication is that there is a correlation between certain values of the TTL and the presence of an attack. Assuming we could identify a anomalous TTL value, we might simply decide to filter all traffic containing this value. However, there are two problems with this idea. First, an attacker can simply modify the TTL field such that the traffic carries a value other than the anomalous value and thus would evade filtering. Second, the TTL field is decremented according to the number of "hops" the packet traverses in the network. An innocuous packet might incidentally carry this value when it arrives and would be dropped by our filter. Thus, we see that the use of a likely irrelevant feature has serious repercussions on the accuracy and usefulness of an anomaly detection system.

In subsequent work called LERAD [12], Mahoney and Chan try to further remove all domain knowledge by eliminating all notions of protocol attributes and treating network data as a simple sequence of bytes. This method required an aggressive rule pruning strategy during a validation phase to identify and remove irrelevant rules. However, as will be shown in Section 4.4, significant numbers of irrelevant rules can remain following this process.

Specification-based anomaly detection [87,88] uses extended finite state automata based on protocol specifications as features. Models then learn the frequencies of certain transitions and the values associated with certain attributes when said transitions occur. These state specifications are manually developed and, as such, protocol complexity can cause the process to be time consuming. Thus, the user must make a trade off between development effort and an increased possibility that some attacks may be missed [87].

Features have been extracted from packet payload data streams. Kruegel and Vigna [89] use tokens contained in HTTP GET requests to identify potential attacks against web services. Wang and Stolfo [90] use frequency distributions of byte-length

characters to model the content of normal requests. Such techniques are reminiscent of work done in the host-based context examining system call and command sequences.

A classifier based on a distributed genetic algorithm, REGAL [91,92], is presented in [93]. The chosen feature set consisted of IP and TCP packet header values. To address the fact that these features often have a large range of values, the authors manually divided ranges into bins and assigned a bin number according to the packet value. This created a "compressed" representation used in classification experiments. The reported results claim better performance than those by Lee and Stolfo [94] using fewer features and less domain knowledge. However, the idiosyncrasies of their data set [86] that could explain this improvement had not yet been identified by Mahoney and Chan [10]. Further, they did not discuss the applicability of their approach to other protocols.

Bauer, et al. [95] describes the construction and use of a model based on an Evolution Strategy [96] to detect TCP SYN Flood attacks [97]. The initial feature set, also know as "object parameters," was created using IP and TCP header values. Object parameter weights were then manually assigned based on their perceived ability to detect this attack. The final feature set consisted of seven features with non-zero weights. A "fitness value" was computed for each destination IP address/TCP port number pair. The hypothesis was that the pair with the highest fitness value (i.e., the most SYN packets) represented an instance of the SYN Flood. Preliminary results indicated that the model was able to detect an attack in a test set of 49 connections. False positive results were not presented. Further, it is unclear if this method is able to detect other types of DoS attacks against TCP. Finally, because the object parameter weights are manually assigned, it is possible that a feature useful in detecting a particular attack may be overlooked.

2.4 Expressing Anomaly Detection Policy

System security is framed in the context of a usage policy. Assessing security of the system can be difficult when this policy is non-existent or ill-defined. However, expressing policy in some contexts, such as a dynamic enterprise network, can be impractical [23]. In addition, different types of modeling techniques present different challenges to expressing policy.

The designer of a misuse detection system makes explicit policy statements when defining signatures. Each signature indicates a prohibited condition (e.g., the appearance of a buffer overflow character sequence) and expresses a policy violation. The expression of an anomaly detection policy is more challenging, because the goal is to express *all* normal data characteristics and a metric to define anomalies. The first challenge faced by the designer of an anomaly detection system is completeness of the feature extraction process. Does the derived feature set adequately represent characteristics of both normal and abnormal data? The next challenge is the creation of a training data sample. Do the selected data adequately represent the population of events? The final challenge is determining a metric to measure deviations from the normal model. The designer of an anomaly detection system makes implicit policy statements when addressing each of these challenges.

It is this added difficulty in defining an anomaly detection policy that can lead to ambiguous anomalies. An anomaly could be a normal condition excluded from the training data. It could also be an instance of erroneous operation or misconfiguration, rather than malicious intent. However, developing a comprehensive anomaly detection policy is important, because identifying significant anomalies can help in the creation of explicit policy statements (i.e., signatures). In particular, the feature extraction process aids policy creation when derived features reveal distinct differences between permissible and prohibited conditions. We present such a method for network protocol attributes in Chapter 3.

2.5 Summary

Many algorithms and techniques have been proposed to perform anomaly detection in computer security. However, one fundamental question, posed by Denning in 1987 and reiterated by McHugh in 2003, still exists, namely; to what extent do anomalies represent intrusions? The focus of intrusion detection research has largely been the quest for an algorithm that can identify a strong link between anomalous behavior and intrusions. However, no algorithm can find such a link if the underlying features cannot adequately capture the notions of normal and intrusive behaviors.

Although the machine learning solutions outlined here claim to perform feature extraction, a more appropriate term to describe the process is *feature selection*. In all cases, the initial set of features is created by a human expert. This creates two problems. The first is completeness; an expert may leave out an important feature. The second is residual irrelevant features; as will be shown in Sections 3.2 and 4.4, irrelevant features can remain after feature selection attempts.

Throughout the history of anomaly detection in computer security, we have seen the following issues related to feature extraction and selection:

- Features were initially created solely by domain experts.
- Features generated through data mining require large amounts of training data and generate many irrelevant features that must be manually pruned. Thus, the manual process of feature creation has been shifted to a manual process of rule pruning, fine tuning of algorithmic parameters, and additional focus on creating representative training data.
- An important feature selection criteria is to avoid the use of irrelevant features. Can likely irrelevant features be identified a priori?
- Often, systems and techniques developed in controlled lab environments do not find comparable success when used in real world environments. Detections are

often "serendipitous" [35]. Could this be mitigated by devising a method to define better features and evaluate their performance?

The next chapter of this dissertation will focus on techniques of identifying likely relevant and irrelevant features in network traffic.

3 PROTOCOL ATTRIBUTE ANALYSIS AND BEHAVIORAL FEATURE EXTRACTION

In this chapter, we present a method of extracting features for network anomaly detection based on protocol specification. The process identifies protocol attributes that are both desirable and undesirable from an anomaly detection perspective. In addition, we present an attribute preprocessing method that can be used to transform these undesirable features into useful ones.

3.1 Inter-Flow versus Intra-Flow Analysis

In this section, we present our protocol analysis method for identifying the relative applicability of attributes as features for anomaly detection. We call this method *Inter-flow versus Intra-flow Analysis* or IVIA. The first step is to identify the protocol attributes that will be used to partition network traffic data into different flows and permit grouping of similar types of flows. For example, in TCP and UDP, a logical flow is indicated by the source and destination port values carried in the packet header [98, 99]. By partitioning the data on these attributes, we can examine all flows with a TCP source port value of 80 (an HTTP server flow) or a UDP source port value of 53 (a DNS Server), for example. The next step involves examining the remaining attributes to determine whether changes occur in the attribute's value between flows (inter-flow) and/or within a flow (intra-flow). Note that if attribute value changes are observed within a flow, the attribute value also exhibits changes among flows.

Table 3.1 shows a matrix of how different classes of attributes are organized using our analysis method. Attributes whose values do not change are assigned to Quadrant IV. Example attributes for this class include those that specify the name of

Table 3.1

Classification of protocol attributes based on observed changes in attribute values between flows (Inter-flow) and/or within flows (Intra-flow).

		Intra-flow Changes	
		Y	**N**
Inter-flow Changes	**Y**	I *Operationally Variable*	II *Flow Descriptors*
	N	III	IV *Operationally Invariant*

the protocol, and those labeled as "reserved" by the protocol. We call the attributes in Quadrant IV *operationally invariant attributes*. Attributes in Quadrant II are designated to partition flows – for a given flow, all values for an attribute will be identical. Conversely, the values will likely be different when comparing different flows. Thus, we expect inter-flow changes in attribute values, but not intra-flow changes. We call these *logical flow attributes*. Examples include TCP and UDP port numbers, and IP source and destination addresses. Quadrant III will necessarily be empty because any changes observed intra-flow must also be visible inter-flow. Finally, Quadrant I contains attributes whose values change within flows. We refer to these as *operationally variable attributes*.

An example of the application of our analysis to the IP Version 4 protocol can be seen in Table 3.2. Figure 3.1 shows how the various protocol attributes are arranged in the packet header. The *Source* and *Destination* addresses, and *Protocol* are the logical flow descriptors that partition flows. These attributes are placed in Quadrant II. The protocol specifies that the *Version* and *Flags_Reserved* attributes have the same value for all flows, thus they are classified as an operationally invariant attributes and placed in Quadrant IV. The remaining attributes are classified as operationally variable attributes and placed in Quadrant I, because changes in their values can be expected both within and among flows.[1] For example, the *Time to Live* attribute indicates the number of network "hops" the packet can traverse before it is discarded. Given that packets associated with a particular flow can traverse different network paths, the value may change within a flow from one packet to the next. A detailed description of the remaining attributes in this class can be found in [100].

Inter-flow versus Intra-flow analysis requires an analysis *perspective*. The placement of an attribute in an IVIA quadrant can depend on where the attribute values are observed. For example, consider a set of IP attributes observed at a web (HTTP) server. The values for both the *Destination* and *Protocol* attributes should be iden-

[1]Despite the fact that *IP Checksum* attribute changes within flows, it was not included in Quadrant I because its value is derived from other values in the packet header.

Table 3.2
IVIA method for the attributes of the IP Version 4 protocol.

		Intra-flow Changes	
		Y	**N**
Inter-flow Changes	**Y**	I Header Length Service type Total Length Identification Flags_DF Flags_MF Fragment Offset Time to Live Options	II Source Destination Protocol
	N	III	IV Version Flags_Reserved

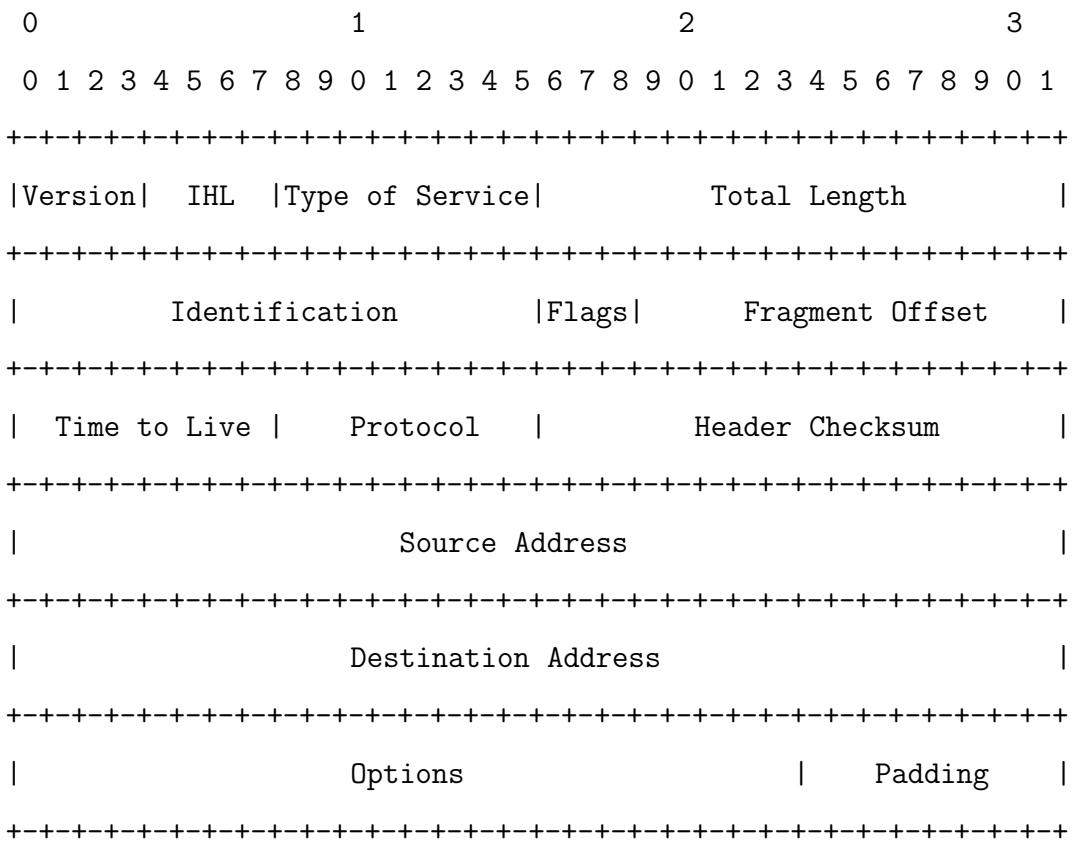

Figure 3.1. IP Version 4 packet header.

tical across all flows. Thus, these attributes could be reclassified as operationally invariant. Such could also be the case when examining attribute values at different protocol layers. For example, ATM cells have uniform length [101], and therefore length could be considered operationally invariant. In this dissertation, we assume that observations are taken at the IP layer of a network with visibility to multiple hosts and network services.

In Section 2.4, we discussed the challenges associated with defining a policy based on anomaly detection. Here we show how the use of IVIA can aid in this process. The quadrant assignment of an attribute gives an indication of how particular attribute values might be enforced through policy. Firewalls are routinely used to filter packets based on attributes in Quadrants II and IV. In the case of IP Version 4, we selectively filter based on the source/destination/protocol of a particular flow. Similarly, we can also use a firewall to drop packets carrying a value other than 4 for the IP *Version*. Quadrant assignment also identifies attributes where such filtering would disrupt the protocol: in particular, those attributes in Quadrant I. Again using IP Version 4 as an example, we cannot arbitrarily enforce certain values for the *Total Length* and *Identification* attributes without limiting the functionality of the protocol.

Quadrant assignment also provides some insight into the applicability of a given feature for anomaly detection. Attributes classified as operationally invariant (Quadrant IV) are useful features because they have only one acceptable value. Any other value should be viewed as suspect. Logical flow attributes (Quadrant II) may be useful for anomaly detection depending on whether policy statements can be made about normal and abnormal flows. A single home user may be able to identify a small set of acceptable hosts and protocols, but a network administrator for a large university may not. Attributes classified as operationally variable (Quadrant I) are of particular interest, because their values can vary significantly and still be considered normal in the context of protocol operation. In the next section, we examine the implications of using these attributes as features for anomaly detection.

3.2 Operationally Variable Attributes

In this section, we examine some practical issues arising from the use of operationally variable attributes as features for anomaly detection. The issues concern the size of the normal value space (and its implications for training data), and the use of these attributes in the context of data mining.

3.2.1 Size of Normal Value Space

Operationally variable attributes can be considered normal over their entire range. Therefore, there is little utility in focusing on discrete values of these attributes as indicators of normal or hostile behavior. If these attributes were used directly to construct an anomaly detection system, any combinations of values identified as anomalous in test data would themselves be normal with respect to the protocol specification. This means that the potential of this system for generating false positive alarms is quite large [11, 80].

To see why, let us assume IVIA on a given protocol has identified n operationally variable attributes labeled O_1 through O_n. The cardinality of a single attribute, $C(O_i)$, is the range of possible normal values for that attribute. Thus, the size of the space of normal values for an instance of the protocol is:

$$\prod_{i=1}^{n} C(O_i) \tag{3.1}$$

By definition, every combination of values in this space is normal with respect to the protocol specification. If we were to build an anomaly detection system for these attributes, we would need an example for every value combination to completely cover the value space. Any combination excluded from training would likely result in a false positive if later encountered by the anomaly detection system.

To understand the scope of this requirement, let us return to the example of IVIA on the IP version 4 protocol. We identified eight operationally variable attributes

– *Header Length, Service Type, Total Length, Identification, DF Flag, MF Flag, Fragment Offset, Time to Live,* and *Options*. The cardinality for each of these attributes ranges from 2 to over 65 thousand.[2] The size of the normal value space for these attributes is 2^{67}, or approximately $1.5e10^{20}$. If we assume it takes a human operator an average of one second to collect and verify a training instance for every combination (a very fast operator!), it would take over $4.7e10^{12}$ person-*years* to complete the training set.

Some combinations of operationally variable attribute values can be logical violations of the protocol. In IP Version 4, a datagram whose *Total Length* is less than the *Header Length* is invalid. Additional examples include contradictory values in state flags, such as a TCP packet with both the *SYN* and *RST* bits set. Other combinations may reveal implementation errors for a particular system. For example, the Land denial of service (DoS) attack [102] caused some network stacks to fail by using a packet with the same value for both the source and destination IP address. Although some of these combinations may be useful for anomaly detection, the number of such combinations is likely to be small in comparison to the size of the overall value space. Using anomaly detection to identify these combinations creates the potential for unacceptably high false positive rates. It is more reasonable to expect that such combinations can be identified by studying the protocol specification.

3.2.2 Data Mining on Operationally Variable Attributes

As was seen in the previous section, the collection of training data using operationally variable attributes presents significant practical problems. We conducted the following experiment to illustrate the drawbacks of approaches that learn a model based on operationally variable attributes. In this experiment, our goal was to build a model of normal IP fragments. This can be seen as a subset of a larger data

[2] *Options* is a variable length attribute. The Options field is not used in our calculation. If Options is used, the size of the value space is 2^{99} or roughly $6.3e10^{29}$.

mining exercise wherein models for multiple protocol behaviors are simultaneously constructed.

We collected approximately 5400 examples of normal fragments from the 1999 DARPA IDS Evaluation Data Sets [103]. The operationally variable attributes for the IP Version 4 protocol (as shown in Quadrant I of Table 3.2) were then extracted from each packet and used as features. With the training data set constructed, we used the Magnum Opus [104] association rule mining algorithm to create a collection of association rules. Association rules are used to identify frequent relationships among attribute values in the data [105, 106]. They are often used to identify buying patterns of shoppers (e.g., those shoppers who bought bread and milk who also bought cereal). In the context of network traffic, the association rules relate to values in the packet header frequently occurring together. This technique has been used to develop a set of rules describing normal traffic [13, 14]. Anomaly alerts are generated when a packet fails to match a sufficient number of these rules.

Table 3.3 shows the top eight ranking rules as generated by Magnum Opus. The rules are sorted by two metrics – *support (S)* and *confidence (C)*. The support metric refers to the percentage of items in the training set whose attributes match the left and right hand side of the rule. The confidence metric indicates the probability of a match of the right hand side (RHS) of the rule given a match of the attributes on the left hand side (LHS).[3]

Using the first rule as an example, we see that roughly 53% of the training instances match the attribute/value combinations in the rule. This rule states that fragments with the MF flag set (indicating it is not the last fragment) and a TTL value of 63 are 98% likely to have a total length of 28 bytes. However, suppose we encounter a fragment that has a TTL value of 62 or a total length of 29 bytes. These values are perfectly valid in the context of the protocol specification, but they would not match the above rule. Here we see the first problem associated with using

[3]Magnum Opus uses the term "strength" to refer to this probability, but we will use the more common term "confidence" in the remainder of this dissertation.

Table 3.3
Association rules generated for IP fragments in the 1999 DARPA IDS Evaluation data sets using operationally variable attributes as features.

ipFlagsMF = 1 & ipTTL = 63 $\Longrightarrow$ ipTLen = 28	$S = 0.526$; $C = 0.981$
ipID < 2817 & ipFlagsMF = 1 $\Longrightarrow$ ipTLen > 28	$S = 0.309$; $C = 0.968$
ipID < 2817 & ipTTL > 63 $\Longrightarrow$ ipTLen > 28	$S = 0.299$; $C = 1.000$
ipTLen > 28 $\Longrightarrow$ ipID < 2817	$S = 0.309$; $C = 1.000$
ipID < 2817 $\Longrightarrow$ ipTLen > 28	$S = 0.309$; $C = 0.927$
ipTTL > 63 $\Longrightarrow$ ipTLen > 28	$S = 0.299$; $C = 0.988$
ipTLen > 28 $\Longrightarrow$ ipTTL > 63	$S = 0.299$; $C = 0.967$
ipTLen > 28 & ipOffset > 118 $\Longrightarrow$ ipTTL > 63	$S = 0.291$; $C = 1.000$

operationally variable attributes as features. Thresholds for distinctions between normal and abnormal values are determined by the supplied training data, but can be arbitrary with respect to the protocol specification.

Looking at the remaining rules in Table 3.3, we see a number of operationally variable attributes appearing as relevant features for describing normal behavior. Examples include *ipID* and *ipTTL* fields. The implication is that the values associated with these attributes can be useful in distinguishing normal from abnormal fragments. However, this is not the case. The *ipID* field is a label indicating ownership by a particular IP datagram. The *ipTTL* field can take any value between 1 and 256. Here we see another problem associated with using operationally variable attributes as features for anomaly detection. Features found to be significant in the training data are often not useful in discriminating malicious from non-malicious events.

If an anomaly detection system were based on the above rules, an alert would be generated if an incoming IP fragment did not match any of the rules. However, it is likely that many normal IP fragments would fail to match these rules and thus result in a false positive alert. Therefore, we conclude that the features used to model IP fragments (the operationally variable attributes) are not able to adequately model the distinction between normal and abnormal fragments.

In conclusion, we avoid the direct use of operationally variable attributes in our models, because 1) they do not have an inherent notion of anomalous values, 2) the collection of adequate training data is time consuming and human intensive, and 3) in the absence of adequate training data, the resulting anomaly detection system has the potential for high false positive rates. In the next section, we show how these attributes can be transformed into useful features for anomaly detection.

3.3 Deriving Behavioral Features

We presented some practical problems associated with using operationally variable attributes as features for anomaly detection. In this section, we present a method to transform operationally variable attributes into features that better capture the notion of anomalous *behavior* of a protocol. To accomplish this, we focus on how the values of a given attribute change during operation. This allows us to better distinguish between normal and abnormal protocol usage than by observing the attribute values directly.

The value of an operationally variable attribute (OVA) at a given instant in time is likely to be normal as defined by the protocol specification. Therefore, the behavior of the attribute must be derived from the transitions in values from one observation to the next. We refer to the description of these transitions as an application of a *behavioral operator*. A behavioral operator is applied to a fixed length sliding window of n packets and quantifies one or more aspects of change in attribute values. This technique is similar to *n-gram analysis* used in data mining of text data [107, 108]. The process uses snapshots of fixed length character sequences to identify frequently occurring words. In our case, we use snapshots of packet sequence values to identify frequently occurring value transitions. The combination of a behavioral operator used with a given OVA value sequence is referred to as a *behavioral feature*. In this section, we present two classes of behavioral operators and show how each class can be used to produce behavioral features.

3.3.1 Analysis of Steps

In [109], Davis describes an inductive learner for time series data. Data is represented by the maximum, minimum, and average values over an arbitrary time window. We present an extension to this method that describes transitions between observed values (i.e., time sequence data) that we refer to as *steps metrics*. Step

metrics are applied over a sliding window of size n sequential observations for an attribute. Metrics we evaluated include:

- Counters indicating the number of positive, negative, and null steps describing directionality
- A range measurement indicating the difference between the global maximum and minimum values
- The mean and variance of the step size

From a performance perspective, it is important to choose behavioral operators that can be computed quickly to keep up with traffic flow. Using a sliding window, the above quantities can be updated for a new observation in constant time – $O(1)$.

Table 3.4 shows two examples of how these metrics describe different sets of transitions with respect to the IP TTL and TCP ACKNUM attributes. A snapshot of a packet window of length five (t_i to t_{i-4}) is applied to each attribute. The values for the IP TTL are identical and can be represented as five null steps. The behavior of the attribute is revealed, namely that values do not change following the establishment of an initial value. The behavior in this example is indicative of path stability; the number of hops between the source and destination remains constant. The step metrics for TCP ACKNUM reveal a different behavior. The metrics indicate five positive steps with a range of 46. The mean step size and variance are 9.2 and 55.76, respectively. Hence, the values of the TCP ACKNUM behave as a monotonically increasing sequence and indicate that the application is making progress in acknowledgment of bytes in the data stream.

Step metrics provide a convenient way of comparing the behavior of OVA value sequences. For example, we can compare the relative progression of values (step counts) and the magnitude and variability in the steps.

Table 3.4
Step metrics of the IP TTL and TCP ACKNUM attributes.

	IP TTL		TCP ACKNUM	
Time	Value	Step	Value	Step
t_i	63	0	847831157	7
t_{i-1}	63	0	847831150	5
t_{i-2}	63	0	847831145	6
t_{i-3}	63	0	847831139	4
t_{i-4}	63	0	847831135	24
t_{i-5}	63		847831111	

3.3.2 Specification-Based Semantics

All step metrics can be applied to any OVA, but if we consider the protocol specification, we can reduce the total number of potential features to those that we anticipate will be highly useful. In examining the specifications of several protocols, we observe that attributes often fall into one of the following semantic categories:

Length - the length of a message or sub-message

Option - a value indicating a specific handling method should be applied to the message

Progress - the total amount of information exchanged over the course of several messages

We can use the semantic category of an attribute to select the most meaningful behavioral operator(s). For example, the variability of the attributes that represent *length* can be captured by first-order statistics (mean and variance). Attributes that represent an *option* are generally binary values and would necessarily exhibit low entropy. Therefore, we use a percentage metric, i.e., the percentage of observations in the window with a given option set. As with the behavioral operators used in the step metrics, these quantities can be updated for a new observation in constant time.

For attributes from the *progress* semantic category, we are interested in knowing both the current "mile post" as well as the overall progress from the beginning of the sequence. In this case, we apply all six step metric operators. Step metrics are particularly useful in identifying "backtracking" behavior. Such behavior manifests itself as a one or more negative steps and can be indicative of overlapping IP fragmentation and TCP segment evasion techniques [110].

Note that there are some attributes that do not fall into one of defined classes (e.g., IP Time To Live, TCP Urgent Pointer, etc.). We use the default step metrics

for these attributes. In future work, we will expand the set of attribute classes and develop new behavioral operators to fit them.

3.4 Higher Order Behavioral Features

The application of behavioral operators is not limited to operationally variable attributes. Features can also be derived from meta-data such as packet time stamps and logical flow attributes (LFAs). For example, a behavioral feature for time stamps might be the average inter-arrival time across a packet window. Examples for LFAs include the number flows of a particular protocol and the entropy associated with TCP or UDP port values across flows within a time window [111].

In the following chapters of this dissertation, we will present empirical results obtained by performing supervised and unsupervised learning on behavioral features derived from network traffic. The features used are largely based on operationally variable attributes. One higher order behavioral feature, mean inter-arrival time, is used in the context of supervised learning.

4 EMPIRICAL RESULTS USING UNSUPERVISED LEARNING

This chapter presents results of using unsupervised learning with behavioral features [112, 113] to construct network anomaly detection models. Direct comparisons are made between models based on behavioral features and traditional packet header values (operationally variable attributes). We show that models based on behavioral features exhibit higher true detection rates and lower false positive rates than operationally variable attribute models.

The remainder of this chapter is organized as follows. 4.1 describes the general feature creation process. In Section 4.2, we describe the construction of our anomaly detection models. Empirical results comparing the use of behavioral features with traditional methods are presented in Section 4.3. An examination of false positive rates is presented in Section 4.4. Feature coverage of the various models is described in Section 4.5. Section 4.6 presents a discussion of the rules used to identify specific attacks. Related work is presented in Section 4.7. Finally, a summary of the results is presented in Section 4.8.

4.1 Feature Extraction for Supervised and Unsupervised Learning

To build a model of normal behavior, one can apply either *supervised* or *unsupervised* learning algorithms. The applicability of an algorithm depends on the nature of the available data. If one has labeled examples of normal and anomalous data, one can apply supervised learning methods to learn a discriminative model. But more commonly, if one has only examples of normal data, one must apply unsupervised learning techniques to learn a model of normal behavior coupled with a metric to judge whether a new observation sufficiently deviates from this model such that it is considered anomalous. A criticism of the use of supervised learning in the context

of network security is the unrealistic expectation of availability of labeled data for all types of attacks. Indeed, the use of anomaly detection is specifically intended to identify novel attack behaviors – those with no known attack signature.

Both supervised and unsupervised methods for building a model of normal behavior require a set of features that represent the data and underlying behavior. Extraneous features increase the noise in the data and reduce the ability of models built on such features to describe the data [37]. The process of *feature extraction* involves the generation of an initial set of features from attributes of the data. Feature extraction is typically performed by a human who determines what measurements to record of the objects of interest.

Feature Selection is the process of identifying relevant features in a set of initial features [114]. Because selection can only be applied to the candidate pool of features, it is particularly important that the extraction process produce a set of features that collectively represent all the ways normal behavior can manifest itself in the data, and further, that they permit the detection of anomalous behavior.

In the context of unsupervised learning, feature selection involves the systematic examination of model accuracy based on different subsets of the proposed initial feature set. As Dy and Brodley [115] point out, the key challenge is determining a metric for measuring model accuracy. Unlike supervised learning where a reduction in the number of false positives and false negatives in a set of validation data can be used to select a relevant feature subset, in unsupervised learning, it is often unclear what metric to apply. For example, one criterion is to find features that enable the data to be clustered well. Another is to find features that lead to association rules with high support and confidence. However, it is unclear whether either of these metrics selects features relevant for detecting anomalous behavior. Thus we conclude that feature extraction for unsupervised learning methods applied to anomaly detection should not take the approach of including all possible measures of the data, but instead should be based on a careful analysis of the data to find features that model the correct function of the protocol of interest.

4.2 Construction of Models

In this section, we describe the process used to construct our experimental models for anomaly detection. We used the 1999 MIT Lincoln Laboratory IDS Evaluation Data Sets [103] as the basis for our experiments. Some aspects of these data sets have been criticized (e.g., [7] and [116]) as unrealistic. However, we found these data sets useful for our purposes for a number of reasons. The first is that a number of the criticisms (e.g., attack data rates and lack of invalid IP checksums) do not manifest themselves in our feature set. Also, the sets contain a diverse collection of attack instances. Finally, the use of this data permitted direct comparison to previous work utilizing the same sets. The models were built using known attack-free data and subsequently tested on data containing normal and attack traffic. The selection of protocols used in our models was based on the dominant protocols in the data. We selected eighteen different protocols for model construction – eight for TCP, seven for UDP, and three for ICMP.

For each of the eighteen selected protocols, we created three models. The first was built using the traditional operationally variable attributes as features. We then used the techniques described in Chapter 3 to build two behavioral feature models – one utilizing only step metrics and the other using the specification-based features. We refer to these as the *OVA*, *Step*, and *Spec* models, respectively. We selected a window size of five when extracting behavioral features, because empirical results in previous work [112] revealed that useful patterns of behavior can be derived using small packet windows.

The extraction process was as follows. A single tcpdump trace file was first partitioned into individual flows based on the selected logical flow attributes. Features were then extracted from each flow (for each of the three models), and these data files were then assembled into training, validation, and test data sets. The next phase involves the construction of rule sets.

Each model was the result of an unsupervised learning process designed to generate a set of association rules that model the characteristics of the normal data. An association rule is an implication of the form $X \Longrightarrow Y$, where $X \cup Y = \emptyset$ and both X and Y are sets of conditions of the data [117, 118]. Algorithms for finding such rules in a dataset look for rules with both high confidence (the percentage of observations in the data that satisfy X and also satisfy Y) and high support (the percentage of the data that satisfy both X and Y). When judging the performance of rule sets, we looked at the following criteria:

Accuracy - the ability of the rule set to differentiate between normal and malicious events

Coverage - the number (or percentage) of available attributes that are represented in the rule set

Clearly, the accuracy of the rule set is paramount. Collectively, an ideal rule set would exhibit high true positive and extremely low false positive rates. Coverage indicates what percentage of the available feature set is utilized. High coverage alone does not indicate a superior rule set. A rule set that utilizes all attributes but whose rules are based on arbitrary relationships is not useful. However, coverage can provide some clues about efficacy of a given rule set. A rule set can be hurt by low coverage because the lack of feature utilization can impact accuracy. Furthermore, a rule set with low coverage presents a potential attacker with a "softer" target, because the attacker need only be concerned with the few attributes actually being monitored.

We used a variant of the LERAD [12] association rule selection algorithm to create each of the rule sets. LERAD uses an initial training set to generate a set of candidate rules, followed by a validation phase to identify and discard rules that generate false positive alerts on normal data. For our purposes, we used the data from week one for initial training and the data from week three for validation. The rules generated are of the form $X \Longrightarrow Y$ and are coupled with an "anomaly score."

This score is computed as the probability that the left hand side X matches the observation, but the right hand side Y does not, namely $P(\neg Y|X)$. In our version of this algorithm, we used the Magnum Opus [104] association rule mining tool to generate the initial set of rules. Magnum Opus reports the confidence probability, $P(Y|X)$, for each rule. We then computed the anomaly score for each rule using $1 - confidence$. The rule validation phase aggressively eliminates any rule found to cause a false positive on the validation data. To minimize the chance of a given rule being discarded, we selected rules from the training phase with a confidence of one. Thus, the rule's anomaly score (i.e., the probability it would be discarded during validation) should be zero. The initial unfiltered rule sets for all models were limited to 100 rules. The 54 filtered rule sets produced by validation were then applied to unseen network traffic to detect anomalies.

To compute the coverage of a rule set, we consider an attribute "covered" if an extracted feature depends on it. A criticism of this method might be that there can be as many as six behavior features (e.g. the step metrics) for each OVA feature and, therefore, a given attribute is as much as six times more likely to be covered by a behavioral feature. This would be true if the distribution of behavioral feature utilization were uniform across attributes. However, this is not the case. The rule mining and selection processes routinely eliminate many individual behavioral features leaving only the most relevant.

4.3 Attack Detection

The following experiments were designed to compare the relative merits of traditional versus behavioral feature based models. The ability of any anomaly-based model to identify attack behaviors depends on the features selected for the model. If a particular type of anomalous behavior does not manifest itself in the feature set, such behavior is not detectable. Such is the case with our data sets. Certain types of attacks often require examination of sequences of application level commands or sys-

tem calls. These attacks fall into the *R2L* (remote to local) and *U2R* (user to root) categories [119]. In these cases, the methods proposed by Lane and Brodley [120], Hofmeyr, et al. [121] and Warrender, et al. [122] provide features for identifying such attacks. Our experiments focused on attacks that manifested themselves in protocol attributes.

We used the validated rule sets to classify a collection of flows from week two of the data set containing both normal and attack traffic. A list of the attack instances used in our evaluation can be seen in Table 4.1. Each instance is designated by its identification number and name. Listed also are the number of flows and packets associated with the attack. In the nomenclature of the MIT Lincoln Laboratory evaluation [119], these attacks fall into the *Probe* and *DoS* (Denial of Service) categories. If one or more rules failed, an alert was generated and we checked the flow against the list of known attacks. An alert that corresponded to one of these known attacks was considered a true positive. Otherwise, it was considered a false positive. We do not require that a model classify as anomalous every flow associated with an attack. A given attack instance can contain a collection of both normal and attack flows across different protocols. Thus, we require that the one of the eighteen protocol models identify some artifact of the overarching attack to be considered a detection.

Detection results can be seen in Table 4.2. The Spec (specification-based) models identified the most attacks – 11 out of 15, or 73.3%. The Step models detected 10 attacks (66.7%), while the OVA model identified the fewest attacks – 8 out of 15, or 53.3%. As a basis for comparison, the ADAM [14] system was the top performing IDS in the 1999 Lincoln Laboratory evaluation utilizing only anomaly detection techniques.[1] That system reported detections of approximately 45% of Denial of Service attacks and 30% of Probe attacks. The best performing models in our experiments, Spec, detected 83.3% and 66.7% of the DoS and Probe attacks, respectively. Both sets of behavioral feature models detected the Ping of Death [123] attacks (numbers

[1]The EMERALD and UCSB Stat systems performed better, but augmented anomaly detection with misuse (i.e., signature) detection.

Table 4.1

Attack types used for evaluation of unseen data as found in week two of the 1999 MIT Lincoln Laboratory evaluation data sets.

ID	Name	Flows	Packets
1	NTInfoscan	15	72
2	PoD	1	235
8	Port Sweep	8	22
13	Mail Bomb	411	4110
14	IP Sweep	7	49
17	SATAN	19	123
18	Mail Bomb	460	4604
20	IP Sweep	8	8
24	SATAN	17	101
25	Port Sweep	7	28
26	Neptune	70	130
30	IP Sweep	5	5
35	PoD	1	435
36	Neptune	70	121
42	Port Sweep	7	27

Table 4.2

Attack detection results for the OVA, Step, and Specification-based models. A bullet indicates detection.

ID	Name	OVA	Step	Spec
1	NTInfoscan	•	•	•
2	PoD		•	•
8	Port Sweep	•	•	•
13	Mail Bomb			
14	IP Sweep			
17	SATAN	•	•	•
18	Mail Bomb			•
20	IP Sweep			
24	SATAN	•	•	•
25	Port Sweep	•	•	•
26	Neptune	•	•	•
30	IP Sweep			
35	PoD		•	•
36	Neptune	•	•	•
42	Port Sweep	•	•	•

2 and *35*). This is notable because they represented only two out of over 2500 otherwise normal ICMP flows in the data. The OVA models did not detect any ICMP-based attacks.

The majority of attacks were present in TCP traffic. At first glance, it might appear that the OVA models performed comparably to the behavioral feature models in this context. However, when we looked at the rules responsible for the detections,

we saw a different picture. Consider the following example from the port 20 rule set that was used to detect the *SATAN* attack:

$$32120 \leq tcp\_winsize \leq 32736 \Longrightarrow 63 \leq ip\_ttl \leq 64$$

Familiarity with the TCP and IP protocols should reveal that this rule is erroneous. A TCP window size advertisement in the specified range can come from anywhere in the network, and as such has no bearing on a particular IP TTL value. The above rule generated what McHugh calls a "serendipitous" detection [35], because the TTL value for the offending packet happened to be 62. If an attacker knew the desired TTL range, the value could easily modified to defeat the rule. Such examples are found throughout the OVA models for TCP, revealing the "brittle" nature of the OVA rule sets, i.e., the rules are easily broken. As will be seen in Section 4.4, the nature of the OVA rule sets can also have a devastating impact on false positive rates. A detailed discussion of the rules used by behavioral feature models is presented in Section 4.6.

One instance of the *Mail Bomb* and all instances of the *IP Sweep* attack escaped detection by all the models. The reason is that these attacks consisted of large numbers of essentially normal flows, as opposed to individual flows containing anomalous value patterns. Although not a part of these experiments, the higher order behavioral features described in Section 3.4 (i.e., models built using logical flow attributes) could be used to construct models of normal patterns for such flows. For example, a model based on the number of ICMP Echo Request flows within a give time window could be used to identify the large number of flows found in the sweep attack instances.

4.4 False Positive Rates

In this section, we present an analysis of the false positive rates exhibited by each of the models used in our experiments. The results show that models based on

behavioral features exhibit markedly lower false positive rates than those based on operationally variable attributes.

The results are grouped according to protocol and can be seen in Tables 4.3 through 4.5. For each rule set, we indicate the number of flows generating an alert and the number of true positives (TP_f) versus false positives (FP_f). A value of "N/A" in the TP_f column means that there were no attacks present in the data, thus no true positive alert can be generated. The ratio $FP_f/Flows$ indicates the number of flows containing a false positive over the total number of normal flows. The ratio $FP_p/Packets$ indicates the number of packets resulting in a false positive over the total number of normal packets.

TCP traffic represented the majority of the packets present in the data, particularly those using the SSH (port 22), Telnet (port 23), and SMTP (port 25) protocols. A variety of TCP-based attacks were present including the NTInfoscan, port sweeps, mail bombs, SATAN, and Neptune (SYN flood). Some of the attack instances (those involving scanning) appeared simultaneously in several protocol models.

4.4.1 TCP Results

Results of using the TCP rule sets can be seen in Table 4.3. Turning our attention first to the models for ports 20 and 21, we found that these OVA models generated more true positive alerts than either of the behavioral feature models. In combination, the OVA models had more false positive packets, but only slightly more false positive flows than the other two models. On the surface, this might lead one to conclude that the OVA rule sets were superior to those found in the behavioral feature models. However, as was pointed out in Section 4.3, these rule sets largely produced serendipitous detections.

If a rule set has the potential to be "lucky" sometimes, it can also be very unlucky at other times. Such is the case with the OVA models for ports 22, 23, and 25. In each case, one or more rules indicated an alert for every test flow. On average, one in

Table 4.3
True positive and false positive alerts generated using TCP rule sets.

Rule Set	Alerts	TP_f	FP_f	$FP_f/Flows$	$FP_p/Packets$
tcp_20_ova	32	23	9	9/5002	13/27977
tcp_20_step	9	3	6	6/5002	12/27977
tcp_20_spec	9	3	6	6/5002	18/27977
tcp_21_ova	36	30	6	6/742	102/32120
tcp_21_step	4	1	3	3/742	5/32120
tcp_21_spec	13	5	8	8/742	19/32120
tcp_22_ova	All	-	-	523/523	327089/3597979
tcp_22_step	0	0	0	0/523	0/3597979
tcp_22_spec	22	1	21	21/523	3360/3597979
tcp_23_ova	All	-	-	3784/3784	1896796/27348108
tcp_23_step	119	0	119	119/3784	224/27348108
tcp_23_spec	4	0	115	115/3784	252/27348108
tcp_25_ova	All	-	-	17045/17045	206622/1977118
tcp_25_step	9	4	5	5/17045	5/1977118
tcp_25_spec	29	27	2	2/17045	2/1977118
tcp_37_ova	26	24	2	2/183	2/1284
tcp_37_step	24	24	0	0/183	0/1284
tcp_37_spec	25	25	0	0/183	0/1284
tcp_79_ova	12	12	0	0/936	0/5561
tcp_79_step	22	22	0	0/936	0/5561
tcp_79_spec	2	2	0	0/936	0/5561
tcp_80_ova	10	1	9	9/5829	11/33774
tcp_80_step	0	0	0	0/5829	0/33774
tcp_80_spec	4	1	3	3/5829	3/33774

ten packets was found to be anomalous. These rule sets clearly have no practical use, because such a tremendous output of false alarms would inundate a human operator.

The poor performance of the aforementioned OVA rule sets severely impacted the overall false positive rates for the TCP OVA models. The false positive flow rate was 64.75%, and the false positive packet rate was 7.36%. Both of the models based on behavioral features exhibited dramatically lower false positive rates for both flows and packets. The flow and packet rates for the Spec models were 0.47% and 0.011%. The Step models achieved the lowest overall flow and packet false positive rates with 0.40% and 0.00075%, respectively. Given that all models detected the same TCP-based attacks (albeit serendipitously in the case of the OVA models), the low false positive rates of the behavioral features models make them more suitable in practice.

4.4.2 UDP Results

There were no detectable attacks present in the UDP flows of the week two data used for our classification experiments.[2] Therefore, we focused our evaluation on the number of false positive alerts generated by each model. The results can be seen in Table 4.4.[3]

In general, we notice that the OVA rule sets generate fewer false positives than seen when viewing TCP flows. In the case of port 138, the OVA model had fewer false alarms than both of the models based on behavioral features. However, this development exposes another weakness in the OVA rule sets. Recall that in the rule validation phase, rules found to generate a false positive on normal data are removed from the set. The OVA rule set for port 138 contained only four rules involving only two features (while competing models contain in excess of ninety rules using nearly all the features). The low number of rules and associated low feature coverage greatly diminished the utility of this model. No results could be obtained for ports 123 and

[2]There were two instances of the Land attack [102], but these were not launched against ports monitored by one of our models.

[3]We eliminated the *True Positives* column for clarity because no true alarms were possible.

Table 4.4
False positive alerts generated using UDP rule sets.

Rule Set	False Positives	$FP_f/Flows$	$FP_p/Packets$
udp_53_ova	0	0/34615	0/110270
udp_53_step	0	0/34615	0/110270
udp_53_spec	0	0/34615	0/110270
udp_123_ova	N/A	N/A	N/A
udp_123_step	0	0/6544	0/54723
udp_123_spec	0	0/6544	0/54723
udp_137_ova	4	4/772	27/12674
udp_137_step	0	0/772	0/12674
udp_137_spec	1	1/772	173/12674
udp_138_ova	0	0/761	0/1427
udp_138_step	1	1/761	18/1427
udp_138_spec	1	1/761	17/1427
udp_161_ova	3	3/5	5/1950
udp_161_step	0	0/5	0/1950
udp_161_spec	0	0/5	0/1950
udp_514_ova	0	0/308	0/651
udp_514_step	0	0/308	0/651
udp_514_spec	0	0/308	0/651
udp_520_ova	N/A	N/A	N/A
udp_520_step	0	0/17	0/39534
udp_520_spec	0	0/17	0/39534

520, because no acceptable rule set could be created. We provide details of this in Section 4.5 when discussing feature coverage.

The overall UDP false positive rates were lower than those for the TCP models. The Step model again achieved the lowest flow and packet rates with 0.0023% and 0.008136%, respectively. The OVA models achieved lower packet rates than the Spec model (taking into account the absence of two port models), but that was attributable to a single flow that generated many packet false alarms (port 137). The OVA models had the highest false positive flow rate with 0.0192%.

4.4.3 ICMP Results

The false positive rates for ICMP models are shown in Table 4.5. There were two examples of the Ping of Death [123] attack in our test data visible when examining the *Echo Reply* traffic. No attacks were present in the *Destination Unreachable* or *Echo Request* data. As such, ideal anomaly detection models based on these protocols should indicate the two attacks in the *Echo Reply* data, but no attacks in the remaining sets.

Six of the nine rule sets generated zero false positive alerts including all of the *Echo Request* models. The *dest_unreach_ova* had only one false positive alert on a flow (constituting one false positive packet observation). This model had a lower false positive rate than the *dest_unreach_step* model, but as we saw in one collection of UDP models, the OVA model had significantly fewer rules. Despite this, however, the OVA model still exhibited more false positives than the *dest_unreach_spec* model.

It is the *Echo Reply* rule sets where the largest disparity in accuracy can be seen among the feature extraction methods. The *echo_reply_ova* rule set indicated four anomalous flows – all were false positives. The following OVA rule was found to be generating false alarms.

$$ip\_id > 40896 \Longrightarrow ip\_flags\_df == 0$$

Table 4.5
True positive and false positive alerts generated using ICMP rule sets.

Rule Set	Alerts	TP_f	FP_f	$FP_f/Flows$	$FP_p/Packets$
echo_reply_ova	4	0	4	4/852	5011/8171
echo_reply_step	2	2	0	0/852	0/8171
echo_reply_spec	2	2	0	0/852	0/8171
dest_unreach_ova	1	N/A	1	1/1023	1/6702
dest_unreach_step	13	N/A	13	13/1023	249/6702
dest_unreach_spec	0	N/A	0	0/1023	0/6702
echo_request_ova	0	N/A	0	0/512	0/8214
echo_request_step	0	N/A	0	0/512	0/8214
echo_request_spec	0	N/A	0	0/512	0/8214

This rule states that when the *Identification* field is greater than 40896, we expect the *Don't Fragment* bit to be set. This rule exemplifies the problem of performing data mining on OVAs. What seems normal in the context of the training data represents only a small fraction of the total normal value space. Unfortunately, this rule set was not able to identify the two Ping of Death attacks because the rules and their associated features are inadequate for distinguishing between normal and abnormal IP fragments. In contrast, both the *echo_reply_step* and *echo_reply_spec* rule sets were able to identify both of the abnormal flows representing the Ping of Death attacks. Further, they were able to do so without generating any false positives on the remainder of the *Echo Reply* data.

4.4.4 False Positive Summary

The comparisons presented in this section demonstrate that models built with operationally variable attributes as features exhibit high false positive rates compared to models built using behavioral features. The combined results of the eighteen protocol models[4] are shown in Table 4.6. Approximately 30% of the flows and 7% of the packets observed by the OVA models were found to be anomalous with respect to the training set. This represents a false positive flow rate of over 148 times greater than the Spec model, and over 156 times greater than the Step model. Differences among false positive packet rates were more dramatic. The rates for OVA models were approximately 632 and 4,766 times greater than the rates than the Spec and Step models, respectively.

4.5 Feature Coverage

In this section, we examine the feature coverage for the rule sets to explain the performance of a given model. In general, we find that behavioral feature rule sets

[4]Two UDP models were excluded from the OVA results because no adequate rule set could be generated

Table 4.6
Overall false positive flow and packet rates for all anomaly detection models.

	OVA	Step	Spec
FP_f	21390	137	157
FP_f Rate	29.77%	0.19%	0.20%
FP_p	2435587	513	3844
FP_p Rate	7.34%	0.00154%	0.0116%

achieved higher feature coverage and larger validated rule sets than those based on operationally variable attributes. This higher coverage permits more aspects of the traffic to be examined.

4.5.1 TCP Rules

Features were extracted from a collection of nineteen protocol attributes – eight for IP and eleven for TCP. The logical flow attributes were chosen to be the IP source and destination addresses and the TCP source and destination port numbers. The eight TCP protocol models were distinguished by their destination port number value.

Feature coverage for the validated rule sets can be seen in Table 4.7. On average, the models based on step metrics exhibited the highest coverage percentage (12.75 of nineteen attributes or 67.1%). The Step rule sets also were the largest on average with 80.88 rules per set.

A curious rule set was that produced for port 22 using the step metrics. The validation phase left only four rules of the original one hundred. Upon further investigation, it was discovered that two validation flows (out of slightly more than two hundred) were responsible for this occurrence. Some of the observations in these flows had significantly larger datagram lengths than those seen in the training data. However, the other behavioral feature based model (Spec) did not experience as marked a rule set reduction.

4.5.2 UDP Rules

The features used for these models were derived from nine attributes – eight from IP and one from UDP. The flow attributes were chosen to be the IP source and destination addresses and the UDP port numbers. The summary of rule statistics for UDP models can be seen in Table 4.8. Whereas the OVA and behavioral feature models were generally comparable in the amount of feature coverage when exam-

Table 4.7

Feature coverage and final rule counts for TCP protocol models following validation (based on nineteen attributes – eight IP and eleven TCP).

		OVA		Step		Spec	
Port	Application	Coverage	Rules	Coverage	Rules	Coverage	Rules
20	FTP Data	10	77	15	88	10	92
21	FTP	13	96	17	93	15	86
22	SSH	11	59	4	4	10	40
23	Telnet	13	61	18	70	8	42
25	SMTP	12	96	13	100	10	78
37	Time	8	68	16	100	10	100
79	Finger	13	95	10	92	10	92
80	HTTP	12	86	9	100	12	100

Table 4.8

Feature coverage and final rule counts for UDP protocol models following validation (based on nine attributes – eight IP and one UDP).

	OVA		Step		Spec	
Port	Coverage	Rules	Coverage	Rules	Coverage	Rules
53	5	16	7	100	6	93
123	N/A	0 (6)	9	100	4	100
137	4	18	9	80	5	79
138	2	4	3	92	6	96
161	2	8	9	100	6	100
514	2	6	9	100	6	100
520	N/A	0	9	100	4	53

ining TCP, UDP OVA models consistently exhibited problems with rule generation and preservation through the validation phase. This can be seen readily as most models generated less than twenty rules that met our minimum confidence criteria. This requirement is important because it helps insure that rules pass through validation. Our first attempt to mine rules for port 123 yielded no rules using the setting for minimum confidence. When we removed the requirement, six rules resulted. However, none of these rules survived the validation phase. Hence, no OVA classification was possible for port 123 flows. The initial rule set for port 520 met a similar fate; all rules were eliminated during validation and thus no classification could take place. The low rule count (and associated low feature coverage) proved to have a detrimental effect on the utility of nearly all of the OVA models.

Rules were generated for the behavioral feature models without difficulty with most models seeing all rules pass through validation. The largest single reduction was 47% (port 520 Spec).

Table 4.9

Feature coverage and final rule counts for ICMP protocol models following validation (based on eight IP attributes).

	OVA		Step		Spec	
Type	Coverage	Rules	Coverage	Rules	Coverage	Rules
Echo Reply	4	5	8	100	6	99
Destination Unreach.	4	24	7	100	4	93
Echo Request	3	4	8	100	6	100

4.5.3 ICMP Rules

Features were extracted from eight IP attributes with the ICMP *Type* and *Code* and IP source and destination addresses designated as flow attributes. This produced three models based on *Echo Reply*, *Destination Unreachable*, *Echo Request* flows. Descriptions of the available ICMP attributes can be found in [124]

The feature coverage for the validated ICMP rule sets can be seen in Table 4.9. We first observe that none of the models based on OVA features utilizes more than 50% of the available attributes. In contrast, the model based solely on step metrics utilized all attributes for two models and seven out of eight for the third. The coverage of specification-based models were roughly between the previous two.

The lower coverage of the OVA models was caused by a relatively small number of rules surviving the validation phase, with some models losing as much as 96% of the initial rule set. As can also be seen in Table 4.9, OVA models retained relatively few rules compared to the behavioral feature models.

4.6 Survey of Detection Rules

In this section, we present examples of rules obtained from the Specification-based (Spec) behavioral feature model that were instrumental in detecting attacks.

The presentation of these rules is meant as a comparison to the problematic OVA model rules previously discussed in this chapter. The detected attacks as shown in Table 4.2 are listed along with a representative sample of rules. Comprehensive lists of rules detecting each attack can be found in Appendix A.

4.6.1 Mail Bomb

The Mail Bomb attack was crafted for the MIT Lincoln Laboratory 1999 IDS Evaluation [103, 119]. It is a collection many large delivery requests designed to create a denial of service attack against an SMTP server. One of two Mail Bomb attacks was identified by the Spec model for port 25 (SMTP) using the following rules.

$$\begin{aligned} (tcp\_hlen\_variance == 0) \quad & \wedge \\ (4 \leq ip\_id\_pos \leq 5) \quad & \wedge \\ (0 \leq tcp\_seqnum\_null \leq 1) \quad & \wedge \\ (0 \leq tcp\_acknum\_null \leq 1) \quad & \Longrightarrow \quad tcp\_acknum\_pos > 3 \end{aligned}$$

This rule expresses expected behavior in the progression of the TCP acknowledgment number relative to the behavior of the TCP sequence number, IP ID, and TCP header length. In this example, the suffixes of *_null* and *_pos* refer to the number of null and positive steps taken by values of the attribute. The Mail Bomb attack was detected because it displayed an unexpected number of positive steps in the TCP acknowledgment number.

4.6.2 Neptune

The Neptune attack is an example of TCP SYN Flooding [125]. When a TCP connection is initiated, the server allocates resources to handle the incoming connection. The objective of this attack is to cause a denial of service by creating many

"half-open" connections and, in turn, exhausting server resources. The following rules came from the port 37 model (although, this attack was detected by several other models).

$$(1 \leq ip\_ttl\_null \leq 2) \implies (0.5 \leq tcp\_flags\_ack\_percent \leq 0.67)$$
$$(1 \leq ip\_ttl\_null \leq 2) \implies (0.33 \leq tcp\_flags\_syn\_percent \leq 0.5)$$

These rules indicate behavioral relationships between the number of null steps exhibited by the IP TTL attribute and the percentage of TCP SYN and ACK packets in a packet window. The Neptune attack issues many connection that do not complete the TCP three-way handshake, and as such the flows associated with this attack display unexpected percentages of TCP SYN and ACK packets.

4.6.3 NTInfoscan

NTInfoscan is a probing attack designed to query the services running on a Windows NT victim host [119]. It attempts to access a variety of services including FTP, Telnet, HTTP, as well as account and file system information. The following rules from the ports 80 and 21 models respectively detected the attack.

$$\begin{aligned}(0 \leq tcp\_seqnum\_null \leq 1) \quad &\wedge \\ (1 \leq tcp\_seqnum\_pos \leq 2) \implies &\ (1 \leq ip\_tos\_null \leq 3) \\ (ip\_ttl\_null == 5) \quad &\wedge \\ (tcp\_acknum\_null < 1) \implies &\ (tcp\_acknum\_pos > 4)\end{aligned}$$

The attack exhibited usual patterns in the IP Type of Service and TCP acknowledgment number. In particular, the second rule indicates a lack of progress in acknowledgment of packets. The rule states that if five packets have been seen (represented by five null steps in the IP TTL), five positive steps in the acknowledgment number are also expected.

4.6.4 Ping of Death

The Ping of Death (PoD) [123] is a denial of service attack that exploits a vulnerability in some ICMP implementations. The attack uses IP fragmentation to produce a datagram that exceeds the maximum datagram size defined by the protocol (64KB). This attack was detected only by the behavioral feature models in our experiments.

$$(ip\_id\_pos > 0) \implies (ip\_foffset\_null > 0)$$
$$(ip\_ttl\_null > 0) \implies (ip\_foffset\_null > 0)$$

These rules represent a sample of twenty-six rules describing relationships between the behavior of the IP Fragment Offset and various other IP header attributes. The PoD attack uses the offset as a means to deliver the fragment that exceeds the IP datagram boundary. The attack was detected because of the unusual pattern exhibited in the number of null steps taken by the offset value.

4.6.5 Port Sweep

Port Sweep is a probing attack designed to identify potentially vulnerable services on a host [119]. Its operation is similar to the NTInfoscan, but can be directed at a variety of different hosts. As with NTInfoscan, the attack was detected by a number of models. The following example rules came from port 79.

$$\begin{aligned}
(2 \leq ip_foffset_null \leq 4) &\implies (0.2 \leq tcp_flags_syn_percent \leq 0.33) \\
(tcp_flags_rst_percent == 0.0) &\wedge \\
(2 \leq ip_ttl_null \leq 4) &\implies (0.2 \leq tcp_flags_syn_percent \leq 0.33) \\
(3 \leq size \leq 5) &\wedge \\
(0 \leq tcp_acknum_null \leq 2) &\wedge \\
(0 \leq tcp_acknum_pos \leq 1) &\implies (0.2 \leq tcp_flags_syn_percent \leq 0.33)
\end{aligned}$$

This collection of rules shows the complex relationships that can be represented using behavioral features. The rules indicate the expected percentage of TCP SYN packets in a packet window relative to the behavior of several IP and TCP attributes, and the number of packets in the window (e.g., $size == 3$). The Port Sweep attack uses a variety of unusual packets to probe, and thus was detected because these patterns do not match those of normal flows.

4.6.6 SATAN

The Security Administrator Tool for Analyzing Networks (SATAN) [126] is used to assess vulnerabilities of a host or network. In the context of these experiments, an attacker uses this tool to perform a probing attack. Similar to other probe attacks, it employs a variety to malformed packets directed at various protocol services to identify potential host vulnerabilities. This attack was detected by more models than any other attack in our experiments. These example rules come from the port 21 model.

$$\begin{aligned}
(ip_id_pos == 5) \quad &\wedge \\
(3 \leq tcp_acknum_pos \leq 4) \quad &\Longrightarrow \quad (1 \leq tcp_acknum_null \leq 2) \\
(tcp_winsize_mean == 32120) \quad &\wedge \\
(tcp_acknum_null < 1) \quad &\Longrightarrow \quad (tcp_acknum_pos > 4) \\
(ip_tos_range == 16) \quad &\wedge \\
(tcp_acknum_null < 1) \quad &\Longrightarrow \quad (tcp_acknum_step_mean > 28.4)
\end{aligned}$$

The first rule is indicative of progress. Given a window of five packets representing five positive steps in the IP ID with three or four positive steps in the TCP acknowledgment number, we expect to see only one or two null steps in the TCP acknowledgment number. The fact that this rule was violated means that some TCP acknowledgments were duplicated, and could be an example of overlapping evasion techniques [110]. The second rule also relates to TCP acknowledgment num-

ber progress, this time to the mean value of the TCP Window Size. The final rule describes the behavior of the mean of the acknowledgment number step size. The large number of unusual packets generated by this attack often caused a violation of as many as twenty rules for a given packet.

4.7 Related Work

The Packet Header Anomaly Detector (PHAD) [10] and its successor, LERAD [12], are examples of the use of machine learning techniques to identify intrusions using attributes of network packets. An anomalous packet is indicated by its "rarity" in the training data. Training and testing of this system was performed using the 1998 MIT Lincoln Laboratory Intrusion Detection Evaluation Data Sets [86]. The authors identified an artifact in this data set wherein the inclusion of the IP TTL (Time to Live) attribute in the feature set increased intrusion detection rates by more than 30%. This was caused by the use of a small set of TTL values for injected attacks. The authors correctly reasoned that TTL values would be unlikely to identify attacks in real network traffic. However, such is also the case for other operationally variable attributes.

A number of systems have employed manually crafted features that attempt to model aspects of connection behavior. These features include the number of connections between hosts [15] and the number of bytes transferred between hosts [40,127] in a given time frame. Models based on such features are focused on certain types of attacks, such as worm propagation. As such, these features may not provide the descriptive power needed for a broad collection of attack behaviors.

Works by Lee and Stolfo [79, 80] and Barbara, et al. [14] are examples of the application of data mining to high level network events to select relevant features from an initial feature set. In both cases, the initial feature set represents a collection of packet header values (such as network addresses and port numbers), status flags (indicating a the state of the connection), and other features manually created and

assembled by a human operator. In both of these projects, analysis was confined to the TCP protocol. However, the ability to use these techniques diminishes when one moves away from connection-oriented protocols. In contrast, this dissertation has demonstrated the utility of our method across several protocols (including stateless protocols, UDP and ICMP) while noting that behavioral features can be derived from the attributes of any structured protocol.

4.8 Results Summary

The results in this chapter demonstrate how, through the application of the behavioral feature extraction process outlined in Chapter 3, we were able to create models for anomaly detection using unsupervised learning with higher true positive rates and lower false positive rates compared to models built on traditional packet header values, namely operationally variable attributes.

The 1999 MIT Lincoln Laboratory IDS evaluation used a maximum threshold of ten false positive alerts per day [119]. The test data used in our experiments contained approximately 80,000 flows spread over a five day period, or 16,000 flows per day. Both behavioral models would generate thirty-two false positive flow alerts per day given their false positive flow rates as shown in Table 4.6, thereby exceeding the threshold. However, nearly all of these false positive alerts were produced by the Telnet protocol models (See Table 4.3). Excluding this protocol, both behavioral feature models fall below the ten false positives per day threshold while maintaining the same detection rates. We believe false positive rate reductions are possible using rule set optimization, and we describe future work exploring the methods in Section 6.2. The OVA models significantly exceeded this threshold, generating approximately 4,800 false positive alerts per day. Worse yet, the nature of rules based on OVA values leaves no room for improvement (See Section 3.2).

OVA rule sets were often so noisy that the majority of initial rules were discarded during the validation phase. In two cases, no OVA model could be generated because

all potential rules were eliminated. Despite this aggressive method of eliminating potential sources of false positives, OVA rule sets consistently generated more false positive alerts, with fewer detections, than a comparable behavioral feature model. In three cases, OVA models generated alerts for every examined flow. Effectively, this meant that five of the eighteen OVA models were completely unusable in practice. The poor performance of OVA models can be attributed to the inadequate descriptive power and low feature coverage of the OVA rule sets. These undesirable characteristics indicate that operationally variable attributes should not be used in practice.

The two behavioral feature models exhibited roughly the same performance in terms of accuracy and coverage. The specification-based models achieved slightly higher detection accuracy, but this was coupled with higher false positive rates (although still orders of magnitude less than OVA models). However, the Spec model achieved this with fewer features than the Step models. Therefore, the selection of one of these two models would represent a tradeoff between a tolerable rate of false positives and the desire for higher accuracy.

Operationally variable attributes can be identified *a priori* using the IVIA protocol analysis method described in Chapter 3. Their transformation into behavioral features eliminates the pitfalls identified experimentally in this chapter and provides superior performance.

5 BEHAVIORAL AUTHENTICATION OF SERVER FLOWS

In this chapter, we present an approach to classifying server traffic based on supervised learning of behavioral features. This classification method is independent of port label and therefore provides a more accurate classification in the presence of malicious activity. An empirical evaluation illustrates that models of both aggregate protocol behavior and host-specific protocol behavior obtain high classification accuracies.

5.1 The Need for Authentication of Server Flows

Administrators tasked with determining adherence to a usage policy require an understanding of the nature of the information flowing into and out of a system or network. Without this understanding, they cannot know if information is being compromised, if their system resources are being used appropriately, or if an attacker is using a service for unauthorized access to a host. In this chapter we address the problem of server flow authentication – the on-going identification of server type for a stream of network packets. Specifically, we address the question of whether we can correctly identify the TCP application protocol of a flow based on features that measure the behavior of the flow.

The traditional method of determining the client-server protocol is by inspecting the source and destination port numbers in the TCP header. The mappings between port numbers and their corresponding service are well known [128]. For example, HTTP server traffic uses port 80, and SMTP server traffic uses port 25. In essence, we rely on a correct *labeling* of the traffic to accurately determine its type. The binding between the port label and the type of traffic is a *binding by convention*. This label is also used as a basis for firewall filtering and intrusion detection [15,129].

There are several different attack scenarios where the port number may not be indicative of the true nature of the server traffic. The following are three examples of why this may be so.

Proxies: These servers are used to consolidate access to a particular service for a group of users. For example, web proxies are used to handle all HTTP client requests to external servers. However, there are also proxies that exist for the specific purpose of evading firewall filtering rules for a set of applications [130]. In this case the proxy takes traffic that would normally be dropped by the firewall and re-maps the port numbers to make the traffic appear to be HTTP traffic. Because HTTP traffic is routinely allowed to pass through firewalls, the user of this proxy is able to circumvent the network policy.

Server Backdoors: When a server has been compromised, the attacker often places a "backdoor" in one or more of the running services [131]. The purpose is to provide the attacker with a portal that he/she can use to regain access at a later time. Traffic from this portal will have the same port number label as legitimate traffic for the compromised service. The attacker may replace the binary of an authorized service X with a binary that can function as both X and Telnet. When a packet is received from a particular source IP, the rogue server knows to execute Telnet, otherwise it executes service X.

User-Installed Servers: This category includes the installation of unauthorized Telnet, HTTP, or other servers for some illicit purpose. It also represents the increasing numbers of peer-to-peer file sharing networks [132]. These servers are initiated by the user and can be configured to use almost any port. This category also includes the recent appearance of "super worms" – worms that propagate via e-mail and carry their own mail server [133]. Once installed, these worms utilize their rogue mail server to forward unsolicited e-mail messages i.e., Spam. Without prior knowledge of a port to service mapping, the true nature of the traffic cannot be determined.

Each of these scenarios represents an instance where the port number label fails to accurately indicate the type of traffic. Worse yet, it is precisely these scenarios where an accurate identification of the traffic would reveal a compromised service or policy violation. Thus, there exists a need to classify traffic associated with a particular service, what we will henceforth refer to as a *server flow*, using a method other than a label that is easily modified, ambiguous, or conceals unauthorized activity.

Significant effort has been invested in the design of tools for detecting the presence of unauthorized services on a host. These range from file system integrity tools that detect modification to server application files (e.g., Tripwire [50]) to tools that look for artifacts of successful intrusions (e.g., ChkRootKit [134]). Successful use of these tools requires proper configuration and, in some cases, a suspicion that an attack has occurred. But, *the fact that a machine running these tools was compromised casts doubt on the information these tools report.* For example, if a Linux system has been compromised using an unauthorized Loadable Kernel Module (LKM) [135], it may be impossible to detect this from *inside* the compromised host [136]. This raises the distinct possibility that an unauthorized service can go undetected indefinitely.

For situations where we cannot trust the results from a compromised system, or the operator is unaware of a successful attack, it would be beneficial to have an external auditor for the purpose of ensuring proper server operation and/or detecting unauthorized services. The identification method used by this auditor should eschew port number labels. Rather, the identification should be indicative of the proper *behavior* of a given server flow.

In this chapter, we investigate how server flows can be classified based on their behavior. The result is a system that monitors network traffic to check conformity with expected network services and to detect service anomalies. The remainder of this chapter is organized as follows. Section 5.2 investigates whether behavioral characteristics of server flows can be measured. Section 5.3 discusses how features measuring these characteristics can be used for server identification. Section 5.4 presents an empirical evaluation that illustrates that we can discriminate among

servers based on characteristics of their flow behavior. Section 5.5 discusses how our classification method can be integrated with network intrusion detection systems. Methods an attacker might use to subvert our classification system are presented in Section 5.6. Related work is presented in Section 5.7. Finally, conclusions and future work are discussed in Section 5.8.

5.2 Understanding the Nature of Server Flows

The key issue in the behavioral authentication of server flows is the characteristics or *features* of the traffic should be monitored. We cannot rely on the contents of the payload as a source of features in environments where there are concerns about user privacy, or where encryption is used to hide the data carried in network packets. Rather, we examine the packet header and the operational characteristics of the traffic itself to define our feature set.

For the purposes of our analysis and experiments, we focused on the HTTP, FTP, Telnet, SMTP, and SSH application protocols. These protocols are well understood, stable, widely implemented, and represent the vast majority of user traffic [137].

Based on our initial observations, we concluded that features based on the TCP state flags (URG - Urgent, ACK - Acknowledgment, PSH - Push, RST - Reset, SYN - Synchronize, and FIN - Finish) [98] can operationally differentiate server flow behavior. For example, HTTP traffic generally contains far fewer packets with the PSH flag than does Telnet traffic. Specifically, for each of the flags, we calculate the percentage of packets in a window of size n packets with that flag set. In addition to these six features we calculate the mean inter-arrival time and the mean packet length for the window of n packets. During monitoring, these features are used by the classification method to determine whether the previous n packets match the learned behavior of the server flows.

5.3 Classification of Server Flows

In this section we describe how we can view behavioral authentication of server flows as a supervised machine learning problem. In supervised learning, the learner is given a set of observations each labeled as one of k classes. The learner's task is to form a classifier from the *training set* that can be used to classify previously unseen (and unlabeled) observations as one of the k classes. A criticism of many anomaly detection systems based on data mining/machine learning is that they assume that they are dealing with a supervised learning problem. That is, the learner will be given examples of both normal and attack data [138]. It is unrealistic to think that one will receive labeled attack data for a particular host because the act of generating labeled data requires significant human effort. In such cases, one applies unsupervised learning to form a model of expected behaviors. During monitoring one looks for anomalies with respect to the learned model.

However, server authentication can be naturally cast as either a supervised learning task or an anomaly detection task. To cast the problem as a supervised learning problem we must choose k possible server applications, collect training data for each, and then apply a supervised learning algorithm to form a classifier. Given a new server flow we can then classify it as one of these k types of servers. To cast the problem as an anomaly detection problem we look at each service individually. For each of the k server applications of interest we form a model of normal behavior. Given a new server flow, we compare the new flow to each of the models to determine whether it conforms to any of these models. Casting the problem as an anomaly detection problem uses the same framework as user behavioral authentication [67, 120, 139]. In user authentication the goal is to identify whether the user is behaving normally with respect to a learned profile of behavior.

We chose to investigate server flow authentication based on the supervised learning framework, because we assume a policy exists specifying the services that are to be run on a given host. A drawback of this assumption is that if an attacker replaces

or alters an existing service it may not behave similarly to any of the permitted services, and would therefore be difficult to classify. However, it is unlikely that it will behave *identically* to any of the permitted services. We plan to examine this conjecture in future work.

5.4 An Empirical Evaluation

Our experiments are designed to investigate whether we can classify server flows based on features of behavior. We first describe the data used in the experiments and the supervised learning algorithm we chose. We then present experimental results with learning aggregate flows and by-host flows using both synthetic and real network traffic.

5.4.1 Data Sources

The first data set chosen for our experiments is the 1999 MIT Lincoln Labs Intrusion Detection Evaluation Datasets [103]. Although created for a specific evaluation exercise, these datasets have subsequently been widely used for research into other later intrusion detection systems not part of the original evaluation [10, 13, 31].

The data represent five weeks of simulated network traffic from a fictional Air Force base. Weeks one through three constitute the *training* data used by anomaly-based intrusion detection systems to model behavior. The data in week one and week three are attack-free. There are five network trace files for each week – one for each business day representing network usage from approximately 8:00 AM to 5:00 PM. Each file is in libpcap format (readable with tcpdump), then compressed using gzip. On average, each week consists of roughly 1 GB of compressed data representing 22 million network packets. We used data from week one in our training sets and data from week three in our test sets. Note that we do not use the attack data because our purpose is to evaluate whether we can classify server behavior – not whether

we can detect intrusions. Our assumption is that the intrusion has already occurred and that an attacker has implemented one of the scenarios described in Section 5.1.

In addition to the MIT Lincoln Laboratory data, we include experiments using data obtained from our own network. The purpose here is to test the applicability of our method on "real world" network traffic. In particular, we are interested in classifying traffic from some of the newer peer-to-peer file sharing protocols – something that the Lincoln Laboratory data sets do not contain. Some concerns have been raised about the artificial nature of the Lincoln Laboratory data [7], and thus an additional objective was to identify any marked differences between experiments with these two data sets.

5.4.2 Decision Tree Classifier

We chose to use decision trees because they provide a comprehensible representation of their classification decisions. Although techniques such as boosting [140, 141] or support vector machines [142] might obtain slightly higher classification accuracy, they require more computation during classification and further they obscure the decision making process.

A decision tree is a tree structure where each internal node denotes a test on a feature, each branch indicates an outcome of the test, and the leaf nodes represent class labels. An example decision tree is shown in Figure 5.1. To classify an observation, the *root* node tests the the value of feature A. If the outcome is greater than some value x, the observation is given a label of *Class 1*. If not, we descend the right subtree and test the value for feature B. Tests continue until a leaf node is reached. The label at the leaf node provides the class label for that observation.

We chose to use the C5.0 decision tree algorithm [143] – a widely used and tested implementation. For details regarding the specifics of C5.0 the reader is referred to [143, 144]. Here we provide only the key aspects of the algorithm related to decision tree estimation, particularly as it pertains to feature selection. The most

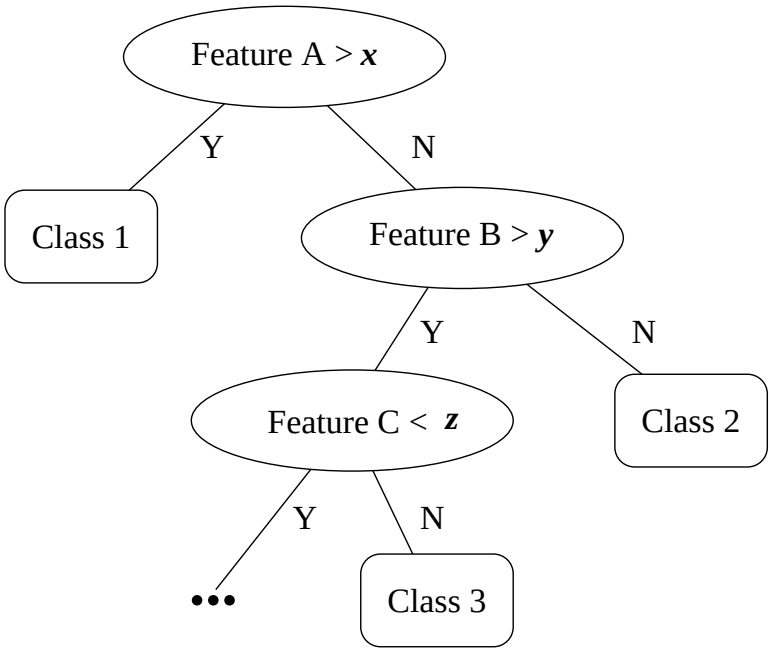

Figure 5.1. Decision tree abstraction showing how the values associated with certain features determine the class label. In this example, observations with feature A greater than the value x are assigned a class label of *Class 1*. Other classifications are based on the values of features B and C.

important element of the decision tree estimation algorithm is the method used to estimate splits at each internal node of the tree. To do this, C5.0 uses a metric called the *information gain ratio* that measures the reduction in entropy in the data produced by a split. In this framework, the test at each node within a tree is selected based on splits of the training data that maximize the reduction in entropy of the descendant nodes. Using this criteria, the training data is recursively split such that the gain ratio is maximized at each node of the tree. This procedure continues until each leaf node contains only examples of a single class or no gain in information is given by further testing. The result is often a large, complex tree that overfits the training data. If the training data contains errors, then overfitting the tree to the data in this manner can lead to poor performance on unseen data. Therefore, the tree must be pruned to reduce classification errors when data outside of the training set are to be classified. To address this problem C5.0 uses confidence-based pruning, and details can be found in [143].

When using the decision tree to classify unseen examples, C5.0 supplies both a class label and a confidence value for its prediction. The confidence value is a decimal number ranging from zero to one – one meaning the highest confidence – and it is given for each instance.

5.4.3 Aggregate Server Flow Model

Our first experiment was designed to determine to what extent FTP, SSH, Telnet, SMTP, and HTTP traffic can be differentiated using a decision tree classifier. We used the data from week one of the Lincoln Laboratory data to build our training dataset. The set was created by first randomly selecting fifty server flows for each of the five protocols. Each server flow consists of the packets from a server to a particular client host/port. The largest flow contained roughly 37,000 packets, and the smallest flow contained 5 packets. The 250 flows represented a total of

approximately 290,000 packets. We refer to this as an *aggregate model* because the collection of flows came from many different servers.

The fact that this data is certified as attack-free meant that we could have confidence in the port numbers as indicative of the type of traffic. We used the server port to label each flow in the training set. Each server flow was then used to generate data observations based on our feature set. The result is a data set consisting of approximately 290,000 labeled observations. We repeated this process for each of seven packet window sizes. The window size is an upper bound on the number of packets used to compute the means and percentages. If an individual flow contains fewer packets than the packet window size, the number of available packets is used to calculate each observation.

Each of the seven training sets was then used to build a decision tree using C5.0. We constructed test sets in the same manner – fifty server flows from each protocol were randomly selected from week three of the Lincoln Laboratory data. These were then passed to our feature extraction algorithm using the same seven window sizes.

Before describing how a tree is used to classify a flow, we give an example of a portion of a decision tree generated by C5.0 in Figure 5.2. In this example, the root node tests the percentage of packets in the packet window with the FIN flag set (tcpPerFIN). If this percentage exceeds 1%, a test is made on the percentage of packets with the PSH flag set (tcpPerPSH). If this value is less than or equal to 40%, the observation is classified as "www", indicating HTTP traffic. The numbers in parenthesis indicate the number of training observations classified with this leaf node. Other tests can be seen involving the mean inter-arrival time (meanIAT) and mean packet length (meanIPTLen).

During testing, the class label for a given flow was calculated by summing the confidence values for each observation in the flow. The class with the highest total confidence was assigned to that flow. The classification results are shown in Table 5.1. For each of seven window sizes, we report the percentage of correctly classified

```
tcpPerFIN > 0.01:
:...tcpPerPSH <= 0.4: www (45)
:   tcpPerPSH > 0.4:
:   :...tcpPerPSH <= 0.797619: smtp (13)
:       tcpPerPSH > 0.797619: ftp (38)
tcpPerFIN <= 0.01:
:...meanIAT > 546773.2:
    :...tcpPerSYN <= 0.03225806: telnet (6090)
    :   tcpPerSYN > 0.03225806:
    :   :...meanipTLen > 73.33: ftp (21)
    :       meanipTLen <= 73.33:
    :       :...tcpPerPSH > 0.7945206: smtp (8)
```

Figure 5.2. Portion of a decision tree generated by C5.0.

Table 5.1

Classification accuracy of the aggregate model decision trees on unseen individual server flows. Each value represents the percentage of correctly classified flows out of the fifty flows for each protocol

Window Size	FTP	SSH	Telnet	SMTP	WWW
1000	100%	88%	94%	82%	100%
500	100%	96%	94%	86%	100%
200	98%	96%	96%	84%	98%
100	100%	96%	96%	86%	100%
50	98%	96%	96%	82%	100%
20	100%	98%	98%	82%	98%
10	100%	100%	100%	82%	98%

server flows out of the set of fifty flows for each protocol. As can be seen in the table, the classification accuracy ranges from 82% to 100%.

In general, the classification accuracy was lower for SMTP server flows than for other protocols. We examined the misclassified flows in more detail and discovered that these flows were generally 2-4 times longer than correctly classified flows. Longer SMTP server flows represented longer periods of interaction, and thus contain increasing numbers of observations classified as Telnet or FTP. In these few cases, our feature set is not adequate for discerning the behavior of these flows.

It is more desirable to use a smaller window size because this decreases the time to detect that a service is behaving abnormally. Indeed for SSH we see that too large a packet window size (1000) hurts classification accuracy. For FTP, SSH and Telnet, a window size as small as ten packets achieves 100% classification accuracy.

Because the proposed method would be used to monitor traffic in real time, we did a rough calculation of classification time. The average length of time used by

C5.0 to classify an entire flow was 70mS.[1] Training is done offline so computation time is of lesser importance, but note that the average length of time used by C5.0 to create each decision tree was 22 seconds. Finally, we need to address the storage requirements for maintaining a window of n values to compute the value of each of the features. We can approximate the value created by storing all n values by retaining only the mean for each feature, μ_{F_i} and using the following update rule for each new packet:

$$\frac{(n-1)\mu_{F_i} + new_{F_i}}{n}$$

In future work we will investigate whether this technique significantly degrades performance.

We conclude from our experimental results that the behavior of server flows for the five protocols can be differentiated using a decision tree classifier built on aggregate flows. We will later discuss how this method can be used to complement an intrusion detection system.

5.4.4 Host-Specific Models

Our second experiment addresses whether creating models for specific hosts provides better performance than the aggregate model. There are three advantages to using host-specific models:

1. By creating models for individual server flows, we can monitor these flows for changes in behavior.

2. A host-specific model can capture the implementation subtleties of a particular service running on a host. This resolution is missing in the aggregate model consisting of many server flows.

[1]The hardware platform used for building the decision trees and classifying observations was a 500Mhz Dual Pentium III PC with 772MB of RAM running Red Hat Linux (kernel version 2.4.18).

Table 5.2
Number of flows used in training and test sets for each host model.

Host	Training Flows	Test Flows
172.16.112.100	20	20
172.16.112.50	30	25
172.16.113.50	35	23
172.16.114.50	10	20
197.218.177.69	25	35

3. The training examples in an aggregate model will be dominated by the server generating the most traffic. This may dilute examples from other servers. The host-specific model solves this problem.

We first identified a set of hosts in the Lincoln Laboratory data that each ran three or more server protocols. Training data for each host was collected by randomly selecting server flows from week one for each of the protocols running on these hosts. The number of flows used in each model is dependent on the number of available flows for a given host. For each host, we chose the same number of flows for each protocol. Table 5.2 lists the number of training and test flows.

Based on our results using the aggregate models, we chose a packet window size of 100 for generating observations. The selection was driven by the fact that SMTP accuracy was greatest using this window size with the aggregate models, and other protocol classifications accuracies were between 96% and 100%. We then trained a decision tree for each host that could be used to differentiate the server flows coming from that host. Test data was collected from week three in the same manner as the training data.

The results in Table 5.3 indicate that, in general, the host specific models achieve approximately the same classification accuracy as the aggregate models. One difference observed is that classification accuracy varies by protocol. For example, the

Table 5.3

Classification accuracy of host model decision trees on unseen server flows. Each row reports the host address and the percentage of correctly classified flows for each protocol. Fields with a – indicate there was no traffic of this protocol type for this host.

Host	FTP	SSH	Telnet	SMTP	WWW
172.16.112.100	95%	–	100%	90%	100%
172.16.112.50	92%	100%	84%	100%	–
172.16.113.50	100%	–	100%	100%	–
172.16.114.50	100%	95%	100%	95%	95%
197.218.177.69	100%	–	100%	100%	–

classification accuracy of Telnet flows for host *172.16.112.50* is 84%, whereas the classification of Telnet flows in the aggregate models averaged 96.2%. Examination of the packets in the misclassified flows revealed an interesting phenomenon. We often observed large time gaps between packets. The time gaps indicate lapses in user activity where the Telnet server is not echoing characters or supplying responses to commands. In our framework, a single large gap can radically alter the values for the mean inter-arrival time of packets, thus resulting in misclassification of the subsequent observations. We refer to this as the *Water Cooler Effect* – the user temporarily leaves the interactive session, then resumes a short while later. We are investigating the sensitivity of our classifiers to this effect. One possible solution would be to subdivide flows based on some time gap threshold and use the interactive sub-flows to build our classifiers.

5.4.5 Models from Real Network Traffic

In this section we present experiments with real network traffic. We collected a number of server flows using the protocols described. We augmented this set to

include flows from hosts acting as Kazaa servers. Kazaa [132, 145] is a peer-to-peer file sharing system that is growing in popularity [146, 147]. Peer-to-peer network traffic was not part of the Lincoln Laboratory dataset.

Our goal was to determine if there was a significant difference in classification accuracy when using synthetic versus real traffic. We observed classification accuracies by protocol ranging from 85% to 100% for both the aggregate and host models. The peer-to-peer traffic was classified correctly for 100% of the unseen flows. This is an especially interesting result because Kazaa flows carry a port label that is user-defined. Thus, we are able to correctly classify peer-to-peer flows behaviorally – without the use of the port number. These results indicate that our classification method is effective for real network traffic. The range of accuracies match those observed with the synthetic data. Thus, we can identify no appreciable difference in the per-flow behavior in the synthetic Lincoln Laboratory data versus those in real network traffic.

5.5 Classification for Intrusion and Misuse Detection

The two types of classification models presented here give rise to new functionality in the context of intrusion and misuse detection. Aggregate models are intended to classify a flow based on the general behavior of many flows of a given type. The question the aggregate model tries to answer is: What other flows does this flow resemble? In contrast, host models are based on the previously observed behavior of flows for a specific host. Given an unseen flow, the host models are intended to answer the question: Does this flow resemble previous server flows from this host?

Intrusion/misuse detection systems and firewalls are designed to identify a priori actions that are harmful to a system or network. An IDS may passively monitor traffic and generate an alert in the presence of an attack condition. Firewalls actively drop network packets that violate some network policy. Our classification method attempts to identify activities indicative of intrusion or misuse *after* such an event has

occurred. Working in concert with a priori mechanisms, we attempt to determine at any moment in time whether there is an impending attack or artifacts of a successful attack.

Figure 5.5 shows how our classification methods can be integrated into a network with an existing IDS. The organization uses servers to provide network services (internally, externally, or both) to some community of users. Our host-flow classification system monitors the output of these servers directly. The purpose is to determine if currently observed flows continue to behave as expected. If an attacker manages to take control of a particular service, he/she will need to interact with the server in such a way as to exactly match the previous behavior. A trojaned web server that behaves similar to a Telnet sever for all but a select group of host addresses would not match the expected host model, and thus be detected.

The network carries additional user traffic to servers that are external to the organization. This traffic is monitored with the aggregate model. Here, we classify the flow generally and compare this to the port label. Observation of traffic that resembles Telnet to some non-standard server port may be an indication of an installed backdoor. Traffic labeled as web traffic (with a server port of 80) that behaves more like Telnet traffic may indicate the presence a proxy used to evade firewall rules. A peer-to-peer client operating at some user-defined port may be a violation of the network policy. In each of these cases the aggregate classifier can indicate if a given flow behaves in a manner consistent with its port label. It may not be necessary to monitor every flow – the system could be configured to randomly select a flow and attempt to classify it. If this flow generally matches a flow that is unusual or undesirable for a port range, it can be identified and investigated.

Our method can operate on its own physical system, or it may be a component of an IDS or firewall. The decision will be driven by the number of flows the system will be expected to monitor. Following the construction of the models, the system makes simple and rapid classifications.

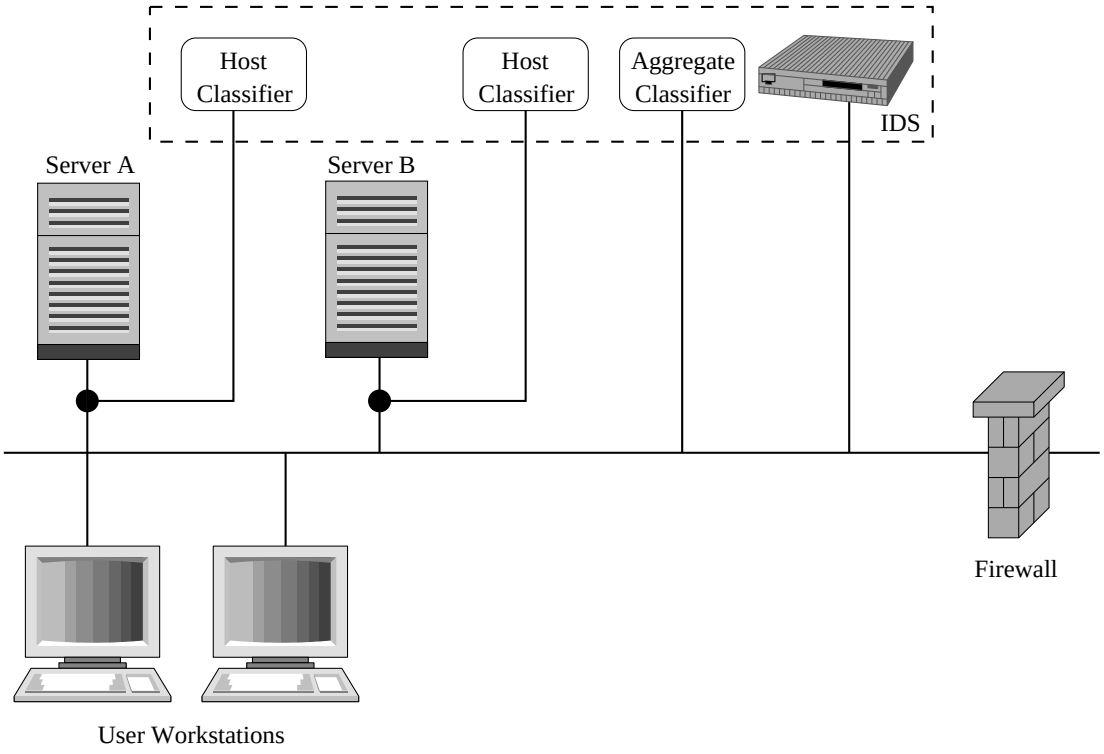

Figure 5.3. Network placement of the host and aggregate classifiers. Host classifiers monitor specific server flows for deviations from expected behavior. Aggregate classifiers monitor user traffic to determine if flow behavior matches generalized behavior of other flows of the same type.

5.6 Subverting Classification

Given the presence of a monitoring system described above, we examined ways an attacker could manipulate a session to affect classification of a server flow. We have previously seen one such example, albeit an innocuous one, in our discussion of the Water Cooler Effect. Here, the user suspends interaction thus causing variation in the arrival time of packets and hence a potentially large fluctuation in the mean inter-arrival time measured across the packet window for the server flow. An attacker can do the same thing. However, it is not clear that he/she would be able to cause a *particular* classification to be chosen. It is more likely that he/she will alter the observations as to cause some indeterminant class to be chosen. If a host classifier is being used, the deviation from the expected behavior would trigger an alarm.

Another method might involve the use of extraneous TCP flags in packets sent to the server. An example might be the use of the URG flag in HTTP packets. The distribution of TCP flags in the corresponding server flow may or may not be affected, based on the implementation of the server on that host. As with the effects of timing, we are investigating the sensitivity of classifiers to this manipulation in future work.

5.7 Related Work

There are a number of commercial products that attempt to identify flow type [148–151]. These are primarily used in the context of bandwidth allocation. For example, a network administrator creates a policy stating that web traffic must not exceed a certain percentage of total bandwidth and uses one of these products to selectively drop traffic when that policy is violated. Many details of the classification methods used by these products are not publicly available because they are proprietary. Thus, we are unable to compare our method to those used in these products. It is unclear whether any of these products can correctly classify flows in the presence of malicious activity (as described in Section 5.1). One company, Packeteer [148],

reports that their product uses information "from all seven layers of the protocol stack" to create an application signature that is used to classify flows. As stated previously, such a system may or may not be appropriate in an environment where payload encryption is used or where there are concerns about user privacy.

We also identified a component of the Snort IDS [15] that is used to classify server flows. However, this system relies on the port number and detection of the TCP 3-way handshake. As stated previously, in the presence of a proxy or compromised service, this system is unlikely to classify a flow correctly.

With respect to our feature set, the NATE (Network Analysis of Anomalous Traffic Events) [31] system is also based on TCP flags. NATE uses principle component analysis to identify that the TCP state flags can detect certain types of attacks. Our method differs in two respects. First, the NATE system attempts to model differences between normal traffic and attack traffic. They do not attempt to model differences in behavior between protocols. The second difference involves NATE's use of clustering to identify anomalies. This method must be done off-line, thus limiting the usefulness of the system in environments requiring near real-time detection. In contrast, once a decision tree has been created, our system can monitor packets in real-time.

5.8 Summary

We presented a novel approach for defining a set of features to model operational behavior of server flow traffic. We demonstrated through the use of the C5.0 decision tree algorithm that our features can differentiate the behavior of server protocols with an accuracy of 82% to 100%. We illustrate empirically that aggregate models can classify an unseen server flow as belonging to a family of previously seen flows, and that host models can determine whether flows from a given server match the behavior of previously seen flows from that server. These classifiers can augment traditional intrusion detection systems to detect artifacts of successful attacks. Our

techniques of classification are independent of packet labellings and are thus immune to techniques that modify port numbers to conceal activity.

The decision tree classifiers can be sensitive to fluctuations in the inter-arrival time of packets. This was exemplified in what we call the Water Cooler Effect. We plan to investigate how this sensitivity can be mitigated to increase the classification accuracy for certain protocols.

6 CONCLUSIONS, SUMMARY, AND FUTURE WORK

The idea of anomaly detection for computer security was believed to be a practical solution for identifying malicious behavior since it was proposed twenty years ago [38]. Early implementations were host based and relied on available accounting information as data sources. The wide-scale proliferation of networking technologies brought the promise that network-based anomaly detection could provide additional capabilities. However, the selection of appropriate models has proven to be a significant challenge.

Models consist of a feature set and an analysis method. The feature sets based on network traffic have either been crafted by human experts (often designed to identify specific types of attacks) or were taken directly from raw network data. Many analysis methods have been proposed, and seemingly all have been applied to both types of feature sets. These methods include statistical analysis, artificial neural networks, and a variety of machine learning and data mining techniques. Despite these many attempts, notable observers (e.g., [35] and [8]) pronounced that existing techniques generated far too many false positive alarms to be practical.

It is illogical to conclude that these analysis methods are the source of the false positive alerts because of the success that these methods have shown in other domains. Rather, these problems indicated that an examination of features and feature extraction techniques was required.

In this dissertation, we presented a protocol analysis methodology to assist in the selection of features for network anomaly detection. We identified classes of protocol attributes based on operational behavior using our *Inter-flow versus Intra-flow Analysis* (IVIA) method. These classes provide a helpful starting point for further analysis and feature extraction. Those attributes identified as *operationally invariant* make useful features because they can carry only one acceptable value.

Conversely, attributes found to be *operationally variable* were shown not to make useful features because of the vast normal value space associated with them.

Avoiding the selection of operationally variable attributes (OVAs) as features removes a significant source of false positive alerts. However, most protocol attributes are likely to be OVAs, so their absence also reduces the ability to model some aspects of behavior. To remedy this problem, we presented two preprocessing transformations designed to represent the behavior of the values associated with OVAs. These transformations have the added benefit that they are computable in constant time, and are therefore computationally inexpensive. We refer to these new transformed features as *behavioral features*.

We next presented experimental results comparing the ability of three models (one using OVAs and the remaining two using behavioral features) to detect a collection of Probing and Denial of Service (DoS) attacks. Although the OVA-based models detected some attacks, the detections were shown to be serendipitous, meaning that detections were based on characteristics unrelated to the mechanics of the attack. Furthermore, the OVA-based models exhibited dramatically higher false positive alarm rates. Rule sets for OVA models exhibited lower feature coverage, and in some cases it was impossible to generate a rule set that satisfied our design constraints. These results revealed that the use of OVAs as features presents a number of practical problems.

Models based on behavioral features were shown to be superior in terms of both detection and false positive rates. False positive rates were two to three orders of magnitude lower than OVA-based models. In contrast to OVA models, detections made by behavioral feature models exhibited strong links to attack behavior. Additional models were used to classify application server flow behavior.

The thesis hypotheses statements listed in Section 1.3 were shown to be true. First, we showed that attributes likely to generate false positives can be identified prior to model construction using IVIA (Sections 3.1 and 3.2). We then showed that these operationally variable attributes can feasibly be transformed into behavioral

features (Sections 3.3 and 3.4). The experimental results presented in Chapter 4 support the hypothesis of superior performance of behavioral feature models. Finally, the experimental results presented in Chapter 5 demonstrate that behavioral feature models can be used to model application server flow behavior without reliance on the port number label carried in the packet header.

This dissertation presents a guide for selection and extraction of features for anomaly detection models. We based our analysis on current protocols, but the method provides sufficient generality to be applicable to future protocols, as well. It is hoped that these techniques will enable future researchers and practitioners to create more accurate and useful network-based anomaly detection systems.

6.1 Summary of Contributions

- Presented a classification method for network protocol attributes based on operational behavior within and among network flows.
- Described the properties of two classes of attributes (operationally variant and invariant) and their suitability as features for anomaly detection.
- Presented a collection of data preprocessing operations that transform operationally variable attributes into behavioral features.
- Described operators for the creation of higher order behavioral features.
- Showed through empirical results that behavioral feature models exhibit higher detection rates and lower false positive rates than those based on operationally variable attributes.
- Identified sources of so-called serendipitous detections in OVA model rule sets.
- Presented examples of behavioral feature rule sets that show correlation between attack mechanisms and feature relationships.

- Showed through empirical results that behavioral feature models can be used to model application server flow behavior independent of port number label carried within packets.
- Described an enhanced intrusion detection system that monitors the behavior of incoming and outgoing server flows.

6.2 Future Work

This dissertation examined the features of network traffic used for anomaly detection and showed the significant performance gains possible when operationally variable attributes are transformed into behavioral features. However, the use of this technique revealed a number of other future research opportunities. We present some of them here.

6.2.1 Behavioral Operators

We presented two types of behavioral operators in this dissertation. The first is a collection of *step metrics* designed to capture the behavior of a generic value sequence. This method has a number of advantages including, (1) it can be applied to any value sequence, and (2) the individual metrics are not computationally expensive. The second type is derived from the protocol specification and is based on the semantics of a given attribute. These *specification-based metrics* produce fewer features than the step metrics, requiring less computation when constructing behavior models. In addition, the models based on specification-based features achieved the highest true detection rate in our experiments. The specification-based metrics are a "shorthand" or optimization for describing the behavior of an attribute.

Future work in this area could explore new types of behavioral operators. These might be used to present alternate representations of generic value sequences, or new semantic optimizations. Examples of the latter include the entropy of certain

packet handling options (e.g., IP Type of Service) and the correctness of checksum fields (i.e., the percentage of correct IP Checksums). Computational complexity and performance comparisons to existing operator/attribute combinations can be made when debating the merits of future behavioral operators.

6.2.2 Enhancing Association Rule Models

The use of association rules for the experiments involving unsupervised learning (Chapter 4) prompted additional questions of how this technique might be used more effectively and efficiently. The following is a discussion of these observations.

Recall that association rules [117,118] are primarily used to discover "interesting" patterns in the data. In these cases, the objective is not to discover relationships among *all* features, but to identify those relationships that meet certain support and confidence thresholds. However, in the context of network anomaly detection, it is unclear what such thresholds should be. Further, when using association rules to describe normal network traffic behavior, we are interested discovering as many feature value interdependencies as possible.

A direction for future work in this area could be to explore the *completeness* of the value space as defined by a set of association rules. Given a set of training data, the task for this new rule mining algorithm would be to identify a minimal rule set that completely covers the value space. Such a set would create a mesh of relationships among features in normal traffic and permit examination of many aspects of unseen traffic.

Another issue related to support and confidence thresholds appeared when implementing our version of the LERAD algorithm [12]. This algorithm generates a set of rules defining characteristics of normal packet values. A new packet observation fails if it matches the left-hand side of a rule, but not the right-hand side. However, when an observation does not match the left-hand side, the algorithm simply moves on to the next rule. In the extreme case, an observation that does not match any

rules would not generate any alerts. However, such an observation clearly cannot be considered normal with respect to the training data. An area of future work is to integrate provisions for the number or percentage of rules that must be satisfied for an observation to be considered normal.

Our final remark concerns the application of rules to new observations. In our experiments, Magnum Opus [104] often generated rules with the following structure.

$$\begin{aligned} A &\implies B \\ &\dots \\ B &\implies C \end{aligned}$$

In this example, condition B is checked twice when applying rules sequentially to a new observation. An optimization would be to coalesce these rules into one to remove the duplicate check of condition B.

$$A \implies B \implies C$$

We intend to explore how such rules can be combined to form "rule trees" that can then be applied to new observations more efficiently.

6.2.3 Misuse / Anomaly Detection Integration

The realization that association rules are sets of normal conditions prompted us to consider their integration with sets of abnormal (i.e., misuse) conditions. Association rules are a human-readable representation of conditional relationships among feature values. When applied to normal network traffic, the rules represent a set of conditions that describe the normal value space. Conditional rules are also used in the context of misuse detection. Here, the conditions represent signatures of attack behavior.

We can represent an instance of a misuse condition as M_i. Examples of misuse conditions are the presence of a specified character string in a packet payload or the

output of a sensor [18]. A misuse signature, S_m is the conjunction of of one or more conditions that generates an alert.

$$If(M_1 \wedge M_2 \wedge \ldots \wedge M_n) \quad then \quad ALERT$$

An anomaly detection condition can be represented by A_i and represents the left and right-hand sides of an association rule. An anomaly detection "signature", S_a, represents a set of one or more association rules defining the normal space for the named features.

$$If \neg(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \quad then \quad ALERT$$

Organizing conditions in this manner presents an opportunity for the integration of behavioral feature monitoring into an existing signature-based network intrusion detection system. Several candidate systems are being considered. The new system would combine signature-based models with anomaly detection based on behavior feature models to produce a hybrid system capable of identifying both known and novel attacks. The system could further provide a feedback mechanism wherein user response to alerts is used to tailor the association rules employing behavioral features.

LIST OF REFERENCES

LIST OF REFERENCES

[1] CERT Coordination Center. CERT/CC Statistics 1988-2005. `http://www.cert.org/advisories/CA-1997-28.html`, May 2005.

[2] Lawrence A. Gordon, Martin P. Loeb, William Lucyshyn, and Robert Richardson. 2004 CSI/FBI Computer Crime and Security Survey. Technical report, Computer Security Institute, 2004.

[3] Internet Systems Consortium. ISC Internet Domain Survey. `http://www.isc.org/index.pl?/ops/ds/host-count-history.php`, May 2005.

[4] Miniwatts International. Internet World Stats. `http://www.internetworldstats.com/stats.htm`, May 2005.

[5] Wenke Lee and Wei Fan. Mining System Audit Data: Opportunities and Challenges. *SIGMOD Rec.*, 30(4):35–44, 2001.

[6] Dorothy E. Denning. An Intrusion-Detection Model. In *IEEE Transactions on Software Engineering*, volume SE-13, pages 222–232, February 1987.

[7] John McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000.

[8] Stefan Axelsson. The Base-Rate Fallacy and Its Implications for the Difficulty of Intrusion Detection. In *ACM Conference on Computer and Communications Security*, pages 1–7, 1999.

[9] D. J. Marchette. A Statistical Method for Profiling Network Traffic. In *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, pages 119–128, 1991.

[10] Matthew Mahoney and Philip K. Chan. PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic. Technical Report CS-2001-4, Florida Institute of Technology, 2001.

[11] Matthew V. Mahoney. Network Traffic Anomaly Detection Based on Packet Bytes. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 346–350. ACM Press, 2003.

[12] M. Mahoney and P. Chan. Learning Models of Network Traffic for Detecting Novel Attacks. Technical Report CS-2002-08, Florida Institute of Technology, 2002.

[13] W. Lee and S. Stolfo. A Framework for Constructing Features and Models for Intrusion Detection Systems. *ACM Transactions on Information and System Security*, 3(4):227–261, November 2000.

[14] D. Barbara, J. Couto, S. Jajodia, and N. Wu. ADAM: A Testbed for Exploring the use of Data Mining in Intrusion Detection. *SIGMOD Record*, 30(4):15–24, 2001.

[15] Martin Roesch and Chris Green. Snort – The Open Source Network Intrusion Detection System. `http://www.snort.org/`, 2003.

[16] Richard Heady, George Luger, Arthur Maccabe, and Mark Servilla. The Architecture of a Network Level Intrusion Detection System. Technical Report CS90-20, Department of Computer Science, University of New Mexico, August 1990.

[17] Biswanath Mukherjee, Todd L. Heberlein, and Karl N. Levitt. Network Intrusion Detection. *IEEE Network*, 8(3):26–41, May/June 1994.

[18] Diego Zamboni. *Using Internal Sensors For Computer Intrusion Detection*. PhD thesis, Purdue University, August 2001.

[19] Mark Crosbie and E. H. Spafford. Active Defense of a Computer System using Autonomous Agents. Technical report, Department of Computer Sciences, 1995. CSD-TR-95-008.

[20] Stefan Axelsson. Research in Intrusion-Detection Systems: A Survey. Technical Report 98–17, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, December 1998.

[21] Stefan Axelsson. Intrusion Detection Systems: A Survey and Taxonomy. Technical Report 99-15, Chalmers Univ., March 2000.

[22] Thomas E. Daniels and Eugene H. Spafford. Identification of Host Audit Data to Detect Attacks on Low-Level IP Vulnerabilities. *Journal of Computer Security*, 7(1):3–35, 1999.

[23] P. A. Porras and P. G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the 20th NIST-NCSC National Information Systems Security Conference*, pages 353–365, 1997.

[24] Eugene H. Spafford and Diego Zamboni. Intrusion Detection using Autonomous Agents. *Computer Networks*, 34(4):547–570, 2000.

[25] Anita K. Jones and Robert S. Sielken. Computer System Intrusion Detection: A Survey. Technical report, University of Virginia Computer Science Department, 1999.

[26] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – A Graph-Based Intrusion Detection System for Large Networks. In *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[27] Paul Spirakis, Sokratis Katsikas, Dimitris Gritzalis, Francois Allegre, John Darzentas, Claude Gigante, Dimitris Karagiannis, P. Kess, Heiki Putkonen, and Thomas Spyrou. SECURENET: A Network-oriented Intelligent Intrusion Prevention and Detection System. *Network Security Journal*, 1(1), November 1994.

[28] Zheng Zhang, Jun Li, C. N. Manikopoulos, Jay Jorgenson, and Jose Ucles. HIDE: a Hierarchical Network Intrusion Detection System using Statistical Preprocessing and Neural Network Classification. In *IEEE Workshop on Information Assurance and Security*, 2001.

[29] C. Sinclair, L. Pierce, and S. Matzner. An Application of Machine Learning to Network Intrusion Detection. In *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC '99)*, pages 371 – 377, Dec 1999.

[30] Juan M. Estévez-Tapiador, Pedro García-Teodoro, and Jesús E. Díaz-Verdejo. Measuring Normality in HTTP Traffic for Anomaly-Based Intrusion Detection. *Computer Networks*, 45(2):175–193, 2004.

[31] Carol Taylor and Jim Alves-Foss. NATE: Network Analysis of Anomalous Traffic Events; A Low Cost Approach. In *Proceedings of the 2001 Workshop on New Security Paradigms*, pages 89–96. ACM Press, 2001.

[32] Stefanos Manganaris, Marvin Christensen, Dan Zerkle, and Keith Hermiz. A Data Mining Analysis of RTID Alarms. *Computer Networks*, 34(4):571–577, 2000.

[33] Christopher Krügel, Darren Mutz, William Robertson, and Fredrik Valeur. Bayesian Event Classification for Intrusion Detection. In *ACSAC*, pages 14–23. IEEE Computer Society, 2003.

[34] Tadeusz Pietraszek. Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection*, pages 102–124, 2004.

[35] John McHugh. Pyrite or Gold: It Takes More Than a Pick and Shovel. In *Second Annual Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, September 2003.

[36] S. Bellovin. RFC 3514: The Security Flag in the IPv4 Header, April 2003.

[37] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant Features and the Subset Selection Problem. In *International Conference on Machine Learning*, pages 121–129, 1994.

[38] James P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical Report 79F296400, James P. Anderson Co., Fort Washington, PA, April 1980.

[39] Koral Ilgun. USTAT: A Real-Time Intrusion Detection System for UNIX. In *Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy*, pages 16–28, Oakland, CA, 1993.

[40] L. Todd Heberlein, Gihan V. Dias, Karl N. Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A Network Security Monitor. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 296–304. IEEE, May 1990. 7-9th, Oakland, CA.

[41] Teresa F. Lunt, Ann Tamaru, Fred Gilham, R. Jagannathan, Peter G. Neumann, and Caveh Jalali. IDES: A Progress Report. In *Proceedings of the Sixth Annual Computer Security Applications Conference*, Sep 1990.

[42] Teresa F. Lunt, Ann Tamaru, Fred Gilham, R. Jagannathan, Caveh Jalali, Peter G. Neumann, Harold S. Javitz, Alfonso Valdes, and Thomas D. Garvey. A Real-Time Intrusion-Detection Expert System (IDES). Technical report, SRII, Feb 1992.

[43] Debra Anderson, Chris Dodd, Fred Gilham, Caveh Jalali, Ann Tamaru, and Mabry Tyson. Next Generation Intrusion Detection Expert System (NIDES): User Manual for Security Officer User Interface (SOUI) Version 1 – Alpha Release. Technical report, SRII, March 1993.

[44] H. Debar, M. Dacier, and A. Wepsi. A Revised Taxonomy for Intrusion-Detection Systems. Technical report, IBM Research – Zurich Research Laboratory, 1999.

[45] SRI. System Requirements for Installation of NIDES Beta-Update Release. `http://www.sdl.sri.com/projects/nides/requirements.html`, July 2005.

[46] H. S. Vaccaro and G. E. Liepins. Detection of Anomalous Computer Session Activity. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 208–209, Oakland, California, May 1989. IEEE Computer Society Press.

[47] Daniel Farmer and Eugene H. Spafford. The COPS Security Checker System. In *Proceedings of the Summer Conference*, pages 165–190, Berkley, CA, 1990. USENIX Association.

[48] David K. Hess, David R. Safford, and Douglas Lee Schales. The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment. Technical report, Texas A&M University, 1993.

[49] David Vincenzetti and Massimo Cotrozzi. ATP – Anti Tampering Program. In *Proceedings of the Fourth USENIX Security Symposium*, pages 79–89, Santa Clara, CA, 1993.

[50] G. H. Kim and G. Spafford. The Design and Implementation of Tripwire: A File System Integrity Checker. In *ACM Conference on Computer and Communications Security*, pages 18–29, 1994.

[51] R. L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm. Technical report, Internet Activities Board, April 1992.

[52] Yinglian Xie, Hyang-Ah Kim, David R. O'Hallaron, Michael K. Reiter, and Hui Zhang. Seurat: A Pointillist Approach to Anomaly Detection. In Jonsson et al. [152], pages 238–257.

[53] H. S. Teng, K. Chen, and S. C-Y. Lu. Security Audit Trail Analysis using Inductively Generated Predictive Rules. In *Proceedings of the Sixth Conference on Artificial intelligence Applications*, pages 24–29, Piscataway, NJ, 1990. IEEE Press.

[54] K. Chen. *An Inductive Engine for the Acquisition of Temporal Knowledge*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1988.

[55] M. A. Maloof and R. S. Michalski. A Partial Memory Incremental Learning Methodology and its Application to Computer Intrusion Detection. Reports of the Machine Learning and Inference Laboratory MLI 95-2, Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA, 1995.

[56] Marcus A. Maloof and Ryszard S. Michalski. A Method for Partial-Memory Incremental Learning and its Application to Computer Intrusion Detection. In *TAI '95: Proceedings of the Seventh International Conference on Tools with Artificial Intelligence*, page 92, Washington, DC, 1995. IEEE Computer Society.

[57] R. Quinlan. Learning Efficient Classification Procedures and their Application to Chess End Games. In R. S. Michalsk, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 463–482. Morgan Kaufmann, 1983.

[58] P. W. Baim. Automated Acquisition of Decision Rules: The Problems of Attribute Construction and Selection. Master's thesis, Department of Computer Science, University Illinois at Urbana-Champaign, 1984.

[59] Jude W. Shavlik and Mark Shavlik. Selection, Combination, and Evaluation of Effective Software Sensors for Detecting Abnormal Computer Usage. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *Knowledge Discovery and Data Mining*, pages 276–285. ACM, 2004.

[60] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A Sense of Self for Unix Processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128, May 1996.

[61] Andreas Wespi, Marc Dacier, and Hervé Debar. Intrusion Detection using Variable-Length Audit Trail Patterns. In Hervé Debar, Ludovic Mé, and Shyhtsun Felix Wu, editors, *Recent Advances in Intrusion Detection*, volume 1907 of *Lecture Notes in Computer Science*, pages 110–129. Springer, 2000.

[62] N. Provos. Improving Host Security with System Call Policies. In *Proceedings of the 12th USENIX Security Symposium*, August 2003.

[63] Christopher Krügel, Darren Mutz, Fredrik Valeur, and Giovanni Vigna. On the Detection of Anomalous System Call Arguments. In Einar Snekkenes and Dieter Gollmann, editors, *ESORICS*, volume 2808 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2003.

[64] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors. In *IEEE Symposium on Security and Privacy*, pages 144–155, 2001.

[65] Henry Hanping Feng, Oleg M. Kolesnikov, Prahlad Fogla, Wenke Lee, and Weibo Gong. Anomaly Detection using Call Stack Information. In *IEEE Symposium on Security and Privacy*, pages 62–. IEEE Computer Society, 2003.

[66] Terran Lane and Carla E. Brodley. Sequence Matching and Learning in Anomaly Detection for Computer Security. In Fawcett, Haimowitz, Provost, and Stolfo, editors, *AI Approaches to Fraud Detection and Risk Management*, pages 43–49. AAAI Press, 1997.

[67] T. Lane. *Machine Learning Techniques for the Computer Security Domain of Anomaly Detection*. PhD thesis, Purdue University, W. Lafayette, IN, August 2000.

[68] Mizuki Oka, Yoshihiro Oyama, Hirotake Abe, and Kazuhiko Kato. Anomaly Detection using Layered Networks Based on Eigen Co-occurrence Matrix. In Jonsson et al. [152], pages 223–237.

[69] Debin Gao, Michael K. Reiter, and Dawn Xiaodong Song. On Gray-Box Program Tracking for Anomaly Detection. In *USENIX Security Symposium*, pages 103–118. USENIX, 2004.

[70] Kymie M. C. Tan and Roy A. Maxion. Why 6? Defining the Operational Limits of STIDE, an Anomaly-Based Intrusion Detector. In *IEEE Symposium on Security and Privacy*, pages 188–201, 2002.

[71] Judith Hochberg, Kathleen Jackson, Cathy Stallings, J. F. McClary, David DuBois, and Josephine Ford. NADIR: An Automated System for Detecting Network Intrusions and Misuse. *Computers and Security*, 12(3):235–248, May 1993.

[72] Kathleen A. Jackson. A NADIR Progress Report. `http://seclab.cs.ucdavis.edu/projects/cmad/4-1996/pdfs/Jackson.pdf`, April 1996.

[73] Richard Heady, George Luger, Arthur Maccabe, Mark Servilla, and John Sturtevant. The Prototype Implementation of a Network Level Intrusion Detection System. Technical Report CS91-11, Department of Computer Science, University of New Mexico, April 1991.

[74] Giovanni Vigna and Richard A. Kemmerer. NetSTAT: A Network-Based Intrusion Detection Approach. In *Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC)*, December 1998.

[75] Jeremy Frank. Artificial Intelligence and Intrusion Detection: Current and Future Directions. In *Proceedings of the 17th National Computer Security Conference*, Baltimore, MD, 1994.

[76] J. Cannady. Artificial Neural Networks for Misuse Detection. In *Proceedings of the 1998 National Information Systems Security Conference (NISSC'98) October 5-8 1998. Arlington, VA.*, pages 443–456, 1998.

[77] Richard P. Lippmann and Robert K. Cunningham. Improving Intrusion Detection Performance using Keyword Selection and Neural Networks. *Computer Networks (Amsterdam, Netherlands: 1999)*, 34(4):597–603, 2000.

[78] A. Bivens, M. Embrechts, C. Palagiri, R. Smith, and B. Szymanski. Network-Based Intrusion Detection using Neural Networks. In *Artificial Neural Networks in Engineering*, volume 12, 2002.

[79] Wenke Lee and Salvatore Stolfo. Data Mining Approaches for Intrusion Detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[80] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. A Data Mining Framework for Building Intrusion Detection Models. In *IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[81] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. Adaptive Intrusion Detection: A Data Mining Approach. *Artificial Intelligence Review*, 14(6):533–567, 2000.

[82] William W. Cohen. Fast Effective Rule Induction. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 9–12, 1995. Morgan Kaufmann.

[83] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[84] CERT Coordination Center. IP Denial-of-Service Attacks, December 1997.

[85] Jianxiong Luo and Susan Bridges. Mining Fuzzy Association Rules and Fuzzy Frequency Episodes for Intrusion Detection. *International Journal of Intelligent Systems*, 15(8):687–703, 2000.

[86] MIT Lincoln Laboratory. 1998 DARPA Intrusion Detection Evaluation Data Set. `http://www.ll.mit.edu/IST/ideval/data/1998/1998\_data\_index.html`, 1998.

[87] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-Based Anomaly Detection: A New Approach for Detecting Network Intrusions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 265–274, New York, NY, 2002. ACM Press.

[88] Ajay Gupta and R. Sekar. An Approach for Detecting Self-Propagating Email using Anomaly Detection. In Vigna et al. [153], pages 55–72.

[89] Christopher Kruegel and Giovanni Vigna. Anomaly Detection of Web-Based Attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 251–261, New York, NY, 2003. ACM Press.

[90] Ke Wang and Salvatore J. Stolfo. Anomalous Payload-Based Network Intrusion Detection. In Jonsson et al. [152], pages 203–222.

[91] Attilio Giordana and Filippo Neri. Search-Intensive Concept Induction. *Evolutionary Computation*, 3(4):375–419, 1995.

[92] Filippo Neri and Lorenza Saitta. Exploring the Power of Genetic Search in Learning Symbolic Classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(11):1135–1141, 1996.

[93] Filippo Neri. Mining TCP/IP Traffic for Network Intrusion Detection by using a Distributed Genetic Algorithm. In Ramon López de Mántaras and Enric Plaza, editors, *ECML*, volume 1810 of *Lecture Notes in Computer Science*, pages 313–322. Springer, 2000.

[94] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. Mining in a Data-flow Environment: Experience in Network Intrusion Detection. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 114–124, New York, NY, 1999. ACM Press.

[95] Dennis C. Bauer, James Cannady, and Raymond C. Garcia. Detecting Anomalous Behavior: Optimization of Network Traffic Parameters Via an Evolutionary Strategy. In *IEEE SoutheastCon 2001*, pages 34–39, 2001.

[96] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[97] CERT. CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks, 1996.

[98] John Postel. RFC 793: Transmission Control Protocol, September 1981.

[99] John Postel. RFC 768: User Datagram Protocol, August 1980.

[100] John Postel. RFC 791: Internet Protocol, September 1981.

[101] The ATM Forum. *ATM User-Network Interface Specification (Version 3.0)*. Prentice-Hall, Inc., Upper Saddle River, NJ, 1993.

[102] NIST. Land IP DoS Attack. `http://icat.nist.gov/icat.cfm?cvename=CVE-1999-0016`, 1999.

[103] MIT Lincoln Laboratory. 1999 DARPA Intrusion Detection Evaluation Data Set. `http://www.ll.mit.edu/IST/ideval/data/1999/1999\_data\_index.html`, 1999.

[104] G. I. Webb. Magnum Opus Rule Mining Tools. `http://www.rulequest.com/MagnumOpus-info.html`, 2004.

[105] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, May 26-28, 1993*, pages 207–216. ACM Press, 1993.

[106] Rakesh Agrawal and Ramakrishnan Srikant. Mining Sequential Patterns. In *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[107] Claudia Pearce and Charles K. Nicholas. Generating a Dynamic Hypertext Environment with N-Gram Analysis. In Bharat K. Bhargava, Timothy W. Finin, and Yelena Yesha, editors, *CIKM*, pages 148–153. ACM, 1993.

[108] Stephen Huffman and Marc Damashek. Acquaintance: A Novel Vector-Space N-Gram Technique for Document Categorization. In *Proceedings of TREC 3*, 1994.

[109] J. H. Davis. CONVART: A Program for Constructive Induction on Time Dependent Data. Master's thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1981.

[110] Thomas H. Ptacek and Timothy N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical Report T2R-0Y6, Secure Networks, Inc., Calgary, Alberta, Canada, 1998.

[111] Alfonso Valdes and Martin Fong. Scalable Visualization of Propagating Internet Phenomena. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, pages 124–127, New York, NY, 2004. ACM Press.

[112] James P. Early, Carla E. Brodley, and Catherine Rosenberg. Behavioral Authentication of Server Flows. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC '03)*. Applied Computer Security Associates (ACSA), Dec 2003.

[113] James P. Early and Carla E. Brodley. Behavioral Features for Network Anomaly Detection. In M. A. Maloof, editor, *Machine Learning and Data Mining in Computer Security: Methods and Applications*. Springer-Verlag, 2005.

[114] Ron Kohavi and George H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[115] Jennifer G. Dy and Carla E. Brodley. Feature Subset Selection and Order Identification for Unsupervised Learning. In *Proceedings of the 17th International Conference on Machine Learning*, pages 247–254. Morgan Kaufmann, San Francisco, CA, 2000.

[116] Matthew V. Mahoney and Philip K. Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In Vigna et al. [153], pages 220–237.

[117] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[118] Ramakrishnan Srikant and Rakesh Agrawal. Mining Generalized Association Rules. *Future Generation Computer Systems*, 13(2–3):161–180, 1997.

[119] Richard Lippmann, David Fried, Isaac Graf, Joshua Haines, Kristopher Kendall, David McClung, Dan Weber, Seth Webster, Dan Wyschogrod, Robert Cunningham, and Marc Zissman. Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, Los Alamitos, CA, 2000. IEEE Computer Society Press.

[120] T. Lane and C. E. Brodley. Temporal Sequence Learning and Data Reduction for Anomaly Detection. *ACM Transactions on Computer Security*, 2(3):295–331, 1999.

[121] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion Detection using Sequences of System Calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[122] Christina Warrender, Stephanie Forrest, and Barak A. Pearlmutter. Detecting Intrusions using System Calls: Alternative Data Models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[123] NIST. Ping of Death – DoS Attack. `http://icat.nist.gov/icat.cfm?cvename=CVE-1999-0128`, 1999.

[124] J. Postel. RFC 792: Internet Control Message Protocol, September 1981.

[125] daemon9 AKA route. Project Neptune. Phrack Magazine. Volume 7. Issue 48(13), 1996.

[126] Dan Farmer and Wietse Venema. Improving the Security of Your Site by Breaking Into It. `http://www.alw.nih.gov/Security/Docs/admin-guide-to-cracking.101.html`, June 2005.

[127] J. B. D. Cabrera, B. Ravichandran, and R. K. Mehra. Statistical Traffic Modeling for Network Intrusion Detection. In *Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 466–473. IEEE Computer Society, 2000.

[128] J. Reynolds and J. Postel. RFC 1700: Assigned Numbers, October 1994.

[129] P. A. Porras and P. G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the 20th NIST-NCSC National Information Systems Security Conference*, pages 353–365, 1997.

[130] iNetPrivacy Software Inc. Antifirewall. `http://www.antifirewall.com/intro.htm`, 2003.

[131] J. Boxmeyer. ONCTec – List of Possible Trojan/Backdoor Port Activity. `http://www.onctek.com/trojanports.html`, 2003.

[132] Evangelos P. Markatos. Tracing a Large Scale Peer-to-Peer System: an Hour in the Life of Gnutella. In *Proceedings of the Second IEEE International Symposium on Cluster Computing and the Grid*, pages 56 – 65. IEEE, 2002.

[133] Saul Hansell. E-mail's Backdoor Open to Spammers. *New York Times : May 20, 2003*, May 2003.

[134] N. Murilo and K. Steding-Jessen. chkrootkit: A Tool that Locally Checks for Signs of a Rootkit. `http://www.chkrootkit.org/`, 2003.

[135] Toby Miller. Detecting Loadable Kernel Modules (LKM). `http://www.linuxsecurity.com/resource_files/host_security/lkm.htm`, June 2005.

[136] E. Skoudis. *Counter Hack: A Step-by-Step Guide to Computer Attacks and Effective Defenses*. Prentice Hall, 2002.

[137] Henning Schulzrinne. Internet Technical Resources – Traffic Statistics. `http://www.cs.columbia.edu/~hgs/internet/traffic.html`.

[138] S. Barbara and S. Jajodia. *Applications of Data Mining in Computer Security*. Kluwer Academic Publishers, 2002.

[139] T. Lane and C. E. Brodley. Temporal Sequence Learning and Data Reduction for Anomaly Detection. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 150–158. Association for Computing Machinery, Nov 1998.

[140] Y. Freund. Boosting a Weak Learning Algorithm by Majority. *Information and Computation*, 121(2):256–285, 1995.

[141] R. Shapire. A Brief Introduction to Boosting. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[142] K. Bennett and C. Campbell. Support Vector Machines: Hype or Hallelujah? *SIGKDD Explorations*, 2:1–13, 2000.

[143] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[144] Ross Quinlan. Data Mining Tools See5 and C5.0. `http://www.rulequest.com/see5-info.html`, 2004.

[145] Sharman Networks Ltd. . Kazaa Media. `http://www.kazaa.com/us/`, 2003.

[146] J. St. Sauver. Percentage of Total Internet2 Traffic Consisting of Kazaa/Morpheus/FastTrack – University of Oregon. In *Collaborative Computing in Higher Education: Peer-to-Peer and Beyond Workshop*, 2002.

[147] Cornell University Student Assembly Committee on Information and Technologies and ResNet. Cornell Internet Usage Statistics. `http://www.cit.cornell.edu/computer/students/bandwidth/charts.html`, 2001.

[148] Packeteer, Inc. Packeteer PacketShaper. `http://www.packeteer.com/resources/prod-sol/PSDS.pdf`, 2003.

[149] Captus Networks. Captus IPS 4000 Series. `http://www.captusnetworks.com/`, 2003.

[150] Stampede Technologies, Inc. TurboGold Enterprise Edition. `http://www.stampede.com/productsclienttoserverfeaturesTraffic.html`, 2003.

[151] NetScreen Technologies, Inc. NetScreen-5000 Series. `http://www.netscreen.com/products/datasheets/ds_ns_5000.jsp`, 2003.

[152] Erland Jonsson, Alfonso Valdes, and Magnus Almgren, editors. *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings*, volume 3224 of *Lecture Notes in Computer Science*. Springer, 2004.

[153] Giovanni Vigna, Erland Jonsson, and Christopher Krügel, editors. *Recent Advances in Intrusion Detection, 6th International Symposium, RAID 2003, Pittsburgh, PA, September 8-10, 2003, Proceedings*, volume 2820 of *Lecture Notes in Computer Science*. Springer, 2003.

APPENDIX

Appendix: Attack Detection Rulesets

This appendix contains a comprehensive list of rules used to identify attack flows listed in Section 4.3. Rules are organized first by model name, then by attack identification number and name.

The feature named *size* refers to the number of observations in the packet window. Suffixes are attached to the names of protocol attributes to designate feature metrics. The following suffixes are used.

_null - Number of null steps

_pos - Number of positive steps

_neg - Number of negative steps

_range - Difference between the global maximum and minimum values

_step_mean - Mean of the step size

_step_variance - Variance of the step size

_mean - Value mean (used for *length* attributes)

_variance - Value variance (used for *length* attributes)

_percent - Percentage of observations in packet window with the attribute set (used for *options* attributes)

A.1 TCP Port 20

A.1.1 17 - SATAN

$$(5.20 \leq tcp\_hlen\_mean \leq 5.50) \implies (40.80 \leq ip\_len\_mean \leq 42.0)$$

$$
\begin{aligned}
(0.20 \leq tcp\_flags\_syn\_percent \leq 0.50) \quad & \wedge \\
(ip\_tos\_range == 8) \quad & \Longrightarrow \quad (40.80 \leq ip\_len\_mean \leq 42.0) \\
(ip\_tos\_range == 8) \quad & \wedge \\
(1 \leq tcp\_urg\_null \leq 4) \quad & \Longrightarrow \quad (40.80 \leq ip\_len\_mean \leq 42.0) \\
(ip\_tos\_range == 8) \quad & \wedge \\
(1 \leq ip\_ttl\_null \leq 4) \quad & \Longrightarrow \quad (40.80 \leq ip\_len\_mean \leq 42.0) \\
(ip\_tos\_range == 8) \quad & \wedge \\
(1 \leq ip\_foffset\_null \leq 4) \quad & \Longrightarrow \quad (40.80 \leq ip\_len\_mean \leq 42.0) \\
(ip\_tos\_range == 8) \quad & \wedge \\
(1 \leq ip\_id\_pos \leq 4) \quad & \Longrightarrow \quad (40.80 \leq ip\_len\_mean \leq 42.0) \\
(0 \leq ip\_tos\_null \leq 3) \quad & \wedge \\
(ip\_tos\_range == 8) \quad & \Longrightarrow \quad (40.80 \leq ip\_len\_mean \leq 42.0) \\
(ip\_tos\_range == 8) \quad & \wedge \\
(0 \leq tcp\_seqnum\_null \leq 3) \quad & \Longrightarrow \quad (40.80 \leq ip\_len\_mean \leq 42.0) \\
(0.0 \leq ip\_len\_variance \leq 3.56) \quad & \wedge \\
(5.20 \leq tcp\_hlen\_mean \leq 5.50) \quad & \wedge \\
(ip\_tos\_range == 8) \quad & \Longrightarrow \quad (ip\_tos\_step\_variance > 10.240) \\
(0.0 \leq ip\_len\_variance \leq 3.56) \quad & \wedge \\
(0.20 \leq tcp\_flags\_syn\_percent \leq 0.50) \quad & \wedge \\
(ip\_tos\_range == 8) \quad & \Longrightarrow \quad (ip\_tos\_step\_variance > 10.240)
\end{aligned}
$$

A.2 TCP Port 21

A.2.1 1 - NTInfoscan

$$
\begin{aligned}
(3 \leq tcp\_acknum\_pos \leq 4) \quad & \wedge \\
(tcp\_urg\_null == 5) \quad & \Longrightarrow \quad (1 \leq tcp\_acknum\_null \leq 2) \\
(ip\_foffset\_null == 5) \quad & \wedge \\
(3 \leq tcp\_acknum\_pos \leq 4) \quad & \Longrightarrow \quad (1 \leq tcp\_acknum\_null \leq 2) \\
(ip\_ttl\_null == 5) \quad & \wedge \\
(3 \leq tcp\_acknum\_pos \leq 4) \quad & \Longrightarrow \quad (1 \leq tcp\_acknum\_null \leq 2) \\
(ip\_id\_pos == 5) \quad & \wedge \\
(3 \leq tcp\_acknum\_pos \leq 4) \quad & \Longrightarrow \quad (1 \leq tcp\_acknum\_null \leq 2) \\
(ip\_tos\_null == 5) \quad & \wedge \\
(3 \leq tcp\_acknum\_pos \leq 4) \quad & \Longrightarrow \quad (1 \leq tcp\_acknum\_null \leq 2) \\
(ip\_tos\_null == 5) \quad & \wedge \\
(tcp\_acknum\_null < 1) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 4) \\
(ip\_id\_pos == 5) \quad & \wedge \\
(tcp\_acknum\_null < 1) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 4) \\
(tcp\_acknum\_null < 1) \quad & \wedge \\
(tcp\_urg\_null == 5) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 4) \\
(ip\_foffset\_null == 5) \quad & \wedge \\
(tcp\_acknum\_null < 1) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 4) \\
(ip\_ttl\_null == 5) \quad & \wedge \\
(tcp\_acknum\_null < 1) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 4) \\
(tcp\_winsize\_mean == 32120.0) \quad & \wedge \\
(ip\_tos\_null == 5) \quad & \wedge
\end{aligned}
$$

$$
\begin{aligned}
(1 \leq tcp\_acknum\_null \leq 2) &\implies (3 \leq tcp\_acknum\_pos \leq 4) \\
(tcp\_winsize\_mean == 32120.0) &\wedge \\
(ip\_tos\_pos == 0) &\wedge \\
(1 \leq tcp\_acknum\_null \leq 2) &\implies (3 \leq tcp\_acknum\_pos \leq 4) \\
(tcp\_winsize\_mean == 32120.0) &\wedge \\
(ip\_tos\_step\_mean == 0.0) &\wedge \\
(1 \leq tcp\_acknum\_null \leq 2) &\implies (3 \leq tcp\_acknum\_pos \leq 4) \\
(tcp\_winsize\_mean == 32120.0) &\wedge \\
(ip\_tos\_step\_variance == 0.0) &\wedge \\
(1 \leq tcp\_acknum\_null \leq 2) &\implies (3 \leq tcp\_acknum\_pos \leq 4) \\
(tcp\_flags\_syn\_percent == 0.0) &\wedge \\
(tcp\_seqnum\_null == 2) &\wedge \\
(1 \leq tcp\_acknum\_null \leq 2) &\implies (3 \leq tcp\_acknum\_pos \leq 4) \\
(tcp\_seqnum\_null == 2) &\wedge \\
(1 \leq tcp\_acknum\_null \leq 2) &\wedge \\
(tcp\_urg\_null == 5) &\implies (3 \leq tcp\_acknum\_pos \leq 4) \\
(ip\_foffset\_null == 5) &\wedge \\
(tcp\_seqnum\_null == 2) &\wedge \\
(1 \leq tcp\_acknum\_null \leq 2) &\implies (3 \leq tcp\_acknum\_pos \leq 4) \\
(ip\_ttl\_null == 5) &\wedge \\
(tcp\_seqnum\_null == 2) &\wedge \\
(1 \leq tcp\_acknum\_null \leq 2) &\implies (3 \leq tcp\_acknum\_pos \leq 4) \\
(ip\_id\_pos == 5) &\wedge \\
(tcp\_seqnum\_null == 2) &\wedge \\
(1 \leq tcp\_acknum\_null \leq 2) &\implies (3 \leq tcp\_acknum\_pos \leq 4) \\
(tcp\_winsize\_mean == 32120.0) &\wedge
\end{aligned}
$$

$$
\begin{aligned}
(ip\_flags\_df\_percent == 1.0) \quad & \wedge \\
(tcp\_seqnum\_step\_mean < 7.0) \quad & \Longrightarrow \quad (tcp\_seqnum\_null > 2) \\
(tcp\_winsize\_mean == 32120.0) \quad & \wedge \\
(tcp\_flags\_fin\_percent == 0.0) \quad & \wedge \\
(tcp\_seqnum\_step\_mean < 7.0) \quad & \Longrightarrow \quad (tcp\_seqnum\_null > 2) \\
(tcp\_winsize\_mean == 32120.0) \quad & \wedge \\
(0.40 \leq tcp\_flags\_psh\_percent \leq 0.60) \quad & \wedge \\
(tcp\_seqnum\_step\_mean < 7.0) \quad & \Longrightarrow \quad (tcp\_seqnum\_null > 2) \\
(ip\_tos\_null == 5) \quad & \wedge \\
(tcp\_seqnum\_null == 2) \quad & \wedge \\
(1 \leq tcp\_acknum\_null \leq 2) \quad & \Longrightarrow \quad (3 \leq tcp\_acknum\_pos \leq 4)
\end{aligned}
$$

A.2.2 17 - SATAN

$$
\begin{aligned}
(tcp\_winsize\_mean == 32120.0) \quad & \wedge \\
(ip\_tos\_null == 5) \quad & \wedge \\
(1 \leq tcp\_acknum\_null \leq 2) \quad & \Longrightarrow \quad (3 \leq tcp\_acknum\_pos \leq 4) \\
(tcp\_winsize\_mean == 32120.0) \quad & \wedge \\
(ip\_tos\_pos == 0) \quad & \wedge \\
(1 \leq tcp\_acknum\_null \leq 2) \quad & \Longrightarrow \quad (3 \leq tcp\_acknum\_pos \leq 4) \\
(tcp\_winsize\_mean == 32120.0) \quad & \wedge \\
(ip\_tos\_step\_mean == 0.0) \quad & \wedge \\
(1 \leq tcp\_acknum\_null \leq 2) \quad & \Longrightarrow \quad (3 \leq tcp\_acknum\_pos \leq 4) \\
(tcp\_winsize\_mean == 32120.0) \quad & \wedge
\end{aligned}
$$

$$
\begin{aligned}
(ip\_tos\_step\_variance == 0.0) \quad & \wedge \\
(1 \leq tcp\_acknum\_null \leq 2) \quad & \Longrightarrow \quad (3 \leq tcp\_acknum\_pos \leq 4) \\
(tcp\_winsize\_mean == 32120.0) \quad & \wedge \\
(ip\_flags\_df\_percent == 1.0) \quad & \wedge \\
(tcp\_seqnum\_step\_mean < 7.0) \quad & \Longrightarrow \quad (tcp\_seqnum\_null > 2) \\
(tcp\_winsize\_mean == 32120.0) \quad & \wedge \\
(tcp\_flags\_fin\_percent == 0.0) \quad & \wedge \\
(tcp\_seqnum\_step\_mean < 7.0) \quad & \Longrightarrow \quad (tcp\_seqnum\_null > 2) \\
(tcp\_winsize\_mean == 32120.0) \quad & \wedge \\
(0.40 \leq tcp\_flags\_psh\_percent \leq 0.60) \quad & \wedge \\
(tcp\_seqnum\_step\_mean < 7.0) \quad & \Longrightarrow \quad (tcp\_seqnum\_null > 2)
\end{aligned}
$$

A.3 TCP Port 22

A.3.1 25 - Port Sweep

$$
\begin{aligned}
(ip\_tos\_null == 5) \quad & \wedge \\
(tcp\_acknum\_null < 2) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(ip\_tos\_pos == 0) \quad & \wedge \\
(ip\_foffset\_null == 5) \quad & \wedge \\
(tcp\_acknum\_null < 2) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(ip\_tos\_step\_variance == 0.0) \quad & \wedge \\
(ip\_foffset\_null == 5) \quad & \wedge \\
(tcp\_acknum\_null < 2) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(ip\_tos\_step\_mean == 0.0) \quad & \wedge
\end{aligned}
$$

$$
\begin{aligned}
(ip\_foffset\_null == 5) \quad & \wedge \\
(tcp\_acknum\_null < 2) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(ip\_tos\_pos == 0) \quad & \wedge \\
(tcp\_acknum\_null < 2) \quad & \wedge \\
(tcp\_urg\_null == 5) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(ip\_tos\_step\_variance == 0.0) \quad & \wedge \\
(tcp\_acknum\_null < 2) \quad & \wedge \\
(tcp\_urg\_null == 5) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(ip\_tos\_step\_mean == 0.0) \quad & \wedge \\
(tcp\_acknum\_null < 2) \quad & \wedge \\
(tcp\_urg\_null == 5) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3)
\end{aligned}
$$

A.4 TCP Port 23

A.4.1 17 - SATAN

$$
\begin{aligned}
(size == 5) \quad & \wedge \\
(2 \leq tcp\_acknum\_null \leq 3) \quad & \Longrightarrow \quad (2 \leq tcp\_acknum\_pos \leq 3) \\
(0.40 \leq tcp\_flags\_psh\_percent \leq 0.60) \quad & \wedge \\
(2 \leq tcp\_acknum\_null \leq 3) \quad & \Longrightarrow \quad (2 \leq tcp\_acknum\_pos \leq 3) \\
(2 \leq tcp\_seqnum\_pos \leq 3) \quad & \wedge \\
(2 \leq tcp\_acknum\_null \leq 3) \quad & \Longrightarrow \quad (2 \leq tcp\_acknum\_pos \leq 3) \\
(2 \leq tcp\_seqnum\_pos \leq 3) \quad & \wedge \\
(2 \leq tcp\_acknum\_null \leq 3) \quad & \Longrightarrow \quad (2 \leq tcp\_seqnum\_null \leq 3)
\end{aligned}
$$

A.4.2 24 - SATAN

$$
\begin{aligned}
(size == 5) \quad & \wedge \\
(2 \leq tcp\_acknum\_null \leq 3) \quad & \Longrightarrow \quad (2 \leq tcp\_acknum\_pos \leq 3) \\
(0.40 \leq tcp\_flags\_psh\_percent \leq 0.60) \quad & \wedge \\
(2 \leq tcp\_acknum\_null \leq 3) \quad & \Longrightarrow \quad (2 \leq tcp\_acknum\_pos \leq 3) \\
(2 \leq tcp\_seqnum\_pos \leq 3) \quad & \wedge \\
(2 \leq tcp\_acknum\_null \leq 3) \quad & \Longrightarrow \quad (2 \leq tcp\_acknum\_pos \leq 3) \\
(2 \leq tcp\_seqnum\_pos \leq 3) \quad & \wedge \\
(2 \leq tcp\_acknum\_null \leq 3) \quad & \Longrightarrow \quad (2 \leq tcp\_seqnum\_null \leq 3)
\end{aligned}
$$

A.5 TCP Port 25

A.5.1 8 - Port Sweep

$$
\begin{aligned}
(tcp\_flags\_ack\_percent < 0.80) \quad & \Longrightarrow \quad (tcp\_flags\_syn\_percent > 0.20) \\
(tcp\_flags\_ack\_percent < 0.80) \quad & \Longrightarrow \quad (tcp\_hlen\_mean > 5.20) \\
(0.0 \leq tcp\_flags\_syn\_percent \leq 0.20) \quad & \Longrightarrow \quad (0.80 \leq tcp\_flags\_ack\_percent \leq 1.0) \\
(5.0 \leq tcp\_hlen\_mean \leq 5.20) \quad & \Longrightarrow \quad (0.80 \leq tcp\_flags\_ack\_percent \leq 1.0)
\end{aligned}
$$

A.5.2 18 - Mail Bomb

$$
(tcp\_hlen\_variance == 0.0) \quad \wedge
$$

$$
\begin{aligned}
(4 \leq ip\_id\_pos \leq 5) \quad & \wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad & \wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(tcp\_hlen\_variance == 0.0) \quad & \wedge \\
(4 \leq ip\_foffset\_null \leq 5) \quad & \wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad & \wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(size == 5) \quad & \wedge \\
(tcp\_hlen\_variance == 0.0) \quad & \wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad & \wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(tcp\_hlen\_variance == 0.0) \quad & \wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad & \wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad & \wedge \\
(4 \leq tcp\_urg\_null \leq 5) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(tcp\_hlen\_variance == 0.0) \quad & \wedge \\
(4 \leq ip\_ttl\_null \leq 5) \quad & \wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad & \wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(tcp\_hlen\_variance == 0.0) \quad & \wedge \\
(4 \leq ip\_tos\_null \leq 5) \quad & \wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad & \wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(tcp\_flags\_ack\_percent < 0.80) \quad & \Longrightarrow \quad (tcp\_flags\_syn\_percent > 0.20) \\
(tcp\_flags\_ack\_percent < 0.80) \quad & \Longrightarrow \quad (tcp\_hlen\_mean > 5.20) \\
(tcp\_flags\_syn\_percent > 0.20) \quad & \Longrightarrow \quad (tcp\_hlen\_mean > 5.20)
\end{aligned}
$$

$$(0.0 \leq tcp\_flags\_syn\_percent \leq 0.20) \implies (0.80 \leq tcp\_flags\_ack\_percent \leq 1.0)$$

$$(5.0 \leq tcp\_hlen\_mean \leq 5.20) \implies (0.80 \leq tcp\_flags\_ack\_percent \leq 1.0)$$

$$(5.0 \leq tcp\_hlen\_mean \leq 5.20) \implies (0.0 \leq tcp\_flags\_syn\_percent \leq 0.20)$$

A.5.3 26 - Neptune

$$\begin{aligned}
(tcp_hlen_variance == 0.0) \quad & \wedge \\
(4 \leq ip_id_pos \leq 5) \quad & \wedge \\
(0 \leq tcp_seqnum_null \leq 1) \quad & \wedge \\
(0 \leq tcp_acknum_null \leq 1) \implies & (tcp_acknum_pos > 3) \\
(tcp_hlen_variance == 0.0) \quad & \wedge \\
(4 \leq ip_foffset_null \leq 5) \quad & \wedge \\
(0 \leq tcp_seqnum_null \leq 1) \quad & \wedge \\
(0 \leq tcp_acknum_null \leq 1) \implies & (tcp_acknum_pos > 3) \\
(size == 5) \quad & \wedge \\
(tcp_hlen_variance == 0.0) \quad & \wedge \\
(0 \leq tcp_seqnum_null \leq 1) \quad & \wedge \\
(0 \leq tcp_acknum_null \leq 1) \implies & (tcp_acknum_pos > 3) \\
(tcp_hlen_variance == 0.0) \quad & \wedge \\
(0 \leq tcp_seqnum_null \leq 1) \quad & \wedge \\
(0 \leq tcp_acknum_null \leq 1) \quad & \wedge \\
(4 \leq tcp_urg_null \leq 5) \implies & (tcp_acknum_pos > 3) \\
(tcp_hlen_variance == 0.0) \quad & \wedge \\
(4 \leq ip_ttl_null \leq 5) \quad & \wedge
\end{aligned}$$

$$
\begin{aligned}
(0 \leq tcp\_seqnum\_null \leq 1) \quad & \wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(tcp\_hlen\_variance == 0.0) \quad & \wedge \\
(4 \leq ip\_tos\_null \leq 5) \quad & \wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad & \wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad & \Longrightarrow \quad (tcp\_acknum\_pos > 3) \\
(tcp\_flags\_ack\_percent < 0.80) \quad & \Longrightarrow \quad (tcp\_flags\_syn\_percent > 0.20) \\
(tcp\_flags\_ack\_percent < 0.80) \quad & \Longrightarrow \quad (tcp\_hlen\_mean > 5.20) \\
(tcp\_flags\_syn\_percent > 0.20) \quad & \Longrightarrow \quad (tcp\_hlen\_mean > 5.20) \\
(0.0 \leq tcp\_flags\_syn\_percent \leq 0.20) \quad & \Longrightarrow \quad (0.80 \leq tcp\_flags\_ack\_percent \leq 1.0) \\
(5.0 \leq tcp\_hlen\_mean \leq 5.20) \quad & \Longrightarrow \quad (0.80 \leq tcp\_flags\_ack\_percent \leq 1.0) \\
(5.0 \leq tcp\_hlen\_mean \leq 5.20) \quad & \Longrightarrow \quad (0.0 \leq tcp\_flags\_syn\_percent \leq 0.20)
\end{aligned}
$$

A.5.4 36 - Neptune

$$
\begin{aligned}
(tcp\_flags\_ack\_percent < 0.80) \quad & \Longrightarrow \quad (tcp\_flags\_syn\_percent > 0.20) \\
(tcp\_flags\_ack\_percent < 0.80) \quad & \Longrightarrow \quad (tcp\_hlen\_mean > 5.20) \\
(tcp\_flags\_syn\_percent > 0.20) \quad & \Longrightarrow \quad (tcp\_hlen\_mean > 5.20) \\
(0.0 \leq tcp\_flags\_syn\_percent \leq 0.20) \quad & \Longrightarrow \quad (0.80 \leq tcp\_flags\_ack\_percent \leq 1.0) \\
(5.0 \leq tcp\_hlen\_mean \leq 5.20) \quad & \Longrightarrow \quad (0.80 \leq tcp\_flags\_ack\_percent \leq 1.0) \\
(5.0 \leq tcp\_hlen\_mean \leq 5.20) \quad & \Longrightarrow \quad (0.0 \leq tcp\_flags\_syn\_percent \leq 0.20)
\end{aligned}
$$

A.6 TCP Port 37

A.6.1 8 - Port Sweep

$$
\begin{aligned}
(1 \leq ip\_tos\_null \leq 2) &\implies (0.50 \leq tcp\_flags\_ack\_percent \leq 0.67) \\
(1 \leq ip\_tos\_null \leq 2) &\implies (0.33 \leq tcp\_flags\_syn\_percent \leq 0.50) \\
(1 \leq ip\_tos\_null \leq 2) &\implies (16316.0 \leq tcp\_winsize\_mean \leq 21584.0) \\
(1 \leq ip\_tos\_null \leq 2) &\implies (5.33 \leq tcp\_hlen\_mean \leq 5.50) \\
(1 \leq ip\_tos\_null \leq 2) &\implies (41.33 \leq ip\_len\_mean \leq 42.0) \\
(1 \leq ip\_foffset\_null \leq 2) &\implies (0.50 \leq tcp\_flags\_ack\_percent \leq 0.67) \\
(1 \leq ip\_foffset\_null \leq 2) &\implies (0.33 \leq tcp\_flags\_syn\_percent \leq 0.50) \\
(1 \leq ip\_foffset\_null \leq 2) &\implies (16316.0 \leq tcp\_winsize\_mean \leq 21584.0) \\
(1 \leq ip\_foffset\_null \leq 2) &\implies (5.33 \leq tcp\_hlen\_mean \leq 5.50) \\
(1 \leq ip\_foffset\_null \leq 2) &\implies (41.33 \leq ip\_len\_mean \leq 42.0) \\
(1 \leq ip\_ttl\_null \leq 2) &\implies (0.50 \leq tcp\_flags\_ack\_percent \leq 0.67) \\
(1 \leq ip\_ttl\_null \leq 2) &\implies (0.33 \leq tcp\_flags\_syn\_percent \leq 0.50) \\
(1 \leq ip\_ttl\_null \leq 2) &\implies (16316.0 \leq tcp\_winsize\_mean \leq 21584.0) \\
(1 \leq ip\_ttl\_null \leq 2) &\implies (5.33 \leq tcp\_hlen\_mean \leq 5.50) \\
(1 \leq ip\_ttl\_null \leq 2) &\implies (41.33 \leq ip\_len\_mean \leq 42.0) \\
(1 \leq ip\_id\_pos \leq 2) &\implies (0.50 \leq tcp\_flags\_ack\_percent \leq 0.67) \\
(1 \leq ip\_id\_pos \leq 2) &\implies (0.33 \leq tcp\_flags\_syn\_percent \leq 0.50) \\
(1 \leq ip\_id\_pos \leq 2) &\implies (16316.0 \leq tcp\_winsize\_mean \leq 21584.0) \\
(1 \leq ip\_id\_pos \leq 2) &\implies (5.33 \leq tcp\_hlen\_mean \leq 5.50) \\
(2 \leq size \leq 3) &\implies (0.50 \leq tcp\_flags\_ack\_percent \leq 0.67) \\
(2 \leq size \leq 3) &\implies (0.33 \leq tcp\_flags\_syn\_percent \leq 0.50) \\
(1 \leq ip\_id\_pos \leq 2) &\implies (41.33 \leq ip\_len\_mean \leq 42.0)
\end{aligned}
$$

$$(2 \leq size \leq 3) \implies (16316.0 \leq tcp\_winsize\_mean \leq 21584.0)$$
$$(2 \leq size \leq 3) \implies (5.33 \leq tcp\_hlen\_mean \leq 5.50)$$
$$(2 \leq size \leq 3) \implies (41.33 \leq ip\_len\_mean \leq 42.0)$$

A.6.2 42 - Port Sweep

$$(1 \leq ip\_tos\_null \leq 2) \implies (1 \leq ip\_id\_pos \leq 2)$$
$$(1 \leq ip\_tos\_null \leq 2) \implies (0.50 \leq tcp\_flags\_ack\_percent \leq 0.67)$$
$$(1 \leq ip\_tos\_null \leq 2) \implies (0.33 \leq tcp\_flags\_syn\_percent \leq 0.50)$$
$$(1 \leq ip\_tos\_null \leq 2) \implies (5.33 \leq tcp\_hlen\_mean \leq 5.50)$$
$$(1 \leq ip\_tos\_null \leq 2) \implies (41.33 \leq ip\_len\_mean \leq 42.0)$$
$$(1 \leq ip\_foffset\_null \leq 2) \implies (1 \leq ip\_id\_pos \leq 2)$$
$$(1 \leq ip\_foffset\_null \leq 2) \implies (0.50 \leq tcp\_flags\_ack\_percent \leq 0.67)$$
$$(1 \leq ip\_foffset\_null \leq 2) \implies (0.33 \leq tcp\_flags\_syn\_percent \leq 0.50)$$
$$(1 \leq ip\_foffset\_null \leq 2) \implies (16316.0 \leq tcp\_winsize\_mean \leq 21584.0)$$
$$(1 \leq ip\_foffset\_null \leq 2) \implies (5.33 \leq tcp\_hlen\_mean \leq 5.50)$$
$$(1 \leq ip\_foffset\_null \leq 2) \implies (41.33 \leq ip\_len\_mean \leq 42.0)$$
$$(1 \leq ip\_ttl\_null \leq 2) \implies (1 \leq tcp\_urg\_null \leq 2)$$
$$(1 \leq ip\_ttl\_null \leq 2) \implies (1 \leq ip\_foffset\_null \leq 2)$$
$$(1 \leq ip\_ttl\_null \leq 2) \implies (1 \leq ip\_id\_pos \leq 2)$$
$$(1 \leq ip\_ttl\_null \leq 2) \implies (1 \leq ip\_tos\_null \leq 2)$$
$$(1 \leq ip\_ttl\_null \leq 2) \implies (0.50 \leq tcp\_flags\_ack\_percent \leq 0.67)$$
$$(1 \leq ip\_ttl\_null \leq 2) \implies (0.33 \leq tcp\_flags\_syn\_percent \leq 0.50)$$

$$(1 \leq ip\_ttl\_null \leq 2) \implies (16316.0 \leq tcp\_winsize\_mean \leq 21584.0)$$
$$(1 \leq ip\_ttl\_null \leq 2) \implies (5.33 \leq tcp\_hlen\_mean \leq 5.50)$$
$$(1 \leq ip\_ttl\_null \leq 2) \implies (41.33 \leq ip\_len\_mean \leq 42.0)$$
$$(1 \leq ip\_ttl\_null \leq 2) \implies (2 \leq size \leq 3)$$
$$(1 \leq ip\_id\_pos \leq 2) \implies (0.50 \leq tcp\_flags\_ack\_percent \leq 0.67)$$

A.7 TCP Port 79

A.7.1 8 - Port Sweep

$$(2 \leq ip\_foffset\_null \leq 4) \implies (2 \leq ip\_id\_pos \leq 4)$$
$$(3 \leq size \leq 5) \quad \wedge$$
$$(0 \leq tcp\_acknum\_null \leq 2) \quad \wedge$$
$$(0 \leq tcp\_acknum\_pos \leq 1) \implies (0.20 \leq tcp\_flags\_syn\_percent \leq 0.33)$$
$$(2 \leq ip\_foffset\_null \leq 4) \implies (0.20 \leq tcp\_flags\_syn\_percent \leq 0.33)$$
$$(2 \leq ip\_tos\_null \leq 4) \implies (2 \leq ip\_id\_pos \leq 4)$$
$$(2 \leq ip\_tos\_null \leq 4) \implies (0.20 \leq tcp\_flags\_syn\_percent \leq 0.33)$$
$$(2 \leq tcp\_urg\_null \leq 4) \implies (2 \leq ip\_id\_pos \leq 4)$$
$$(2 \leq tcp\_urg\_null \leq 4) \implies (0.20 \leq tcp\_flags\_syn\_percent \leq 0.33)$$
$$(2 \leq ip\_ttl\_null \leq 4) \quad \wedge$$
$$(0 \leq tcp\_acknum\_null \leq 2) \implies (2 \leq ip\_id\_pos \leq 4)$$
$$(2 \leq ip\_ttl\_null \leq 4) \quad \wedge$$
$$(0 \leq tcp\_acknum\_null \leq 2) \implies (0.20 \leq tcp\_flags\_syn\_percent \leq 0.33)$$

$$
\begin{aligned}
(tcp\_flags\_rst\_percent == 0.0) \quad &\wedge \\
(2 \leq ip\_ttl\_null \leq 4) \quad &\Longrightarrow \quad (2 \leq ip\_id\_pos \leq 4) \\
(tcp\_flags\_rst\_percent == 0.0) \quad &\wedge \\
(2 \leq ip\_ttl\_null \leq 4) \quad &\Longrightarrow \quad (0.20 \leq tcp\_flags\_syn\_percent \leq 0.33) \\
(2 \leq ip\_ttl\_null \leq 4) \quad &\wedge \\
(ip\_ttl\_step\_mean == 0.0) \quad &\Longrightarrow \quad (2 \leq ip\_id\_pos \leq 4) \\
(2 \leq ip\_ttl\_null \leq 4) \quad &\wedge \\
(ip\_ttl\_step\_mean == 0.0) \quad &\Longrightarrow \quad (0.20 \leq tcp\_flags\_syn\_percent \leq 0.33) \\
(2 \leq ip\_ttl\_null \leq 4) \quad &\wedge \\
(ip\_ttl\_range == 0) \quad &\Longrightarrow \quad (2 \leq ip\_id\_pos \leq 4) \\
(2 \leq ip\_ttl\_null \leq 4) \quad &\wedge \\
(ip\_ttl\_range == 0) \quad &\Longrightarrow \quad (0.20 \leq tcp\_flags\_syn\_percent \leq 0.33) \\
(2 \leq ip\_ttl\_null \leq 4) \quad &\wedge \\
(ip\_ttl\_step\_variance == 0.0) \quad &\Longrightarrow \quad (2 \leq ip\_id\_pos \leq 4) \\
(2 \leq ip\_ttl\_null \leq 4) \quad &\wedge \\
(ip\_ttl\_step\_variance == 0.0) \quad &\Longrightarrow \quad (0.20 \leq tcp\_flags\_syn\_percent \leq 0.33) \\
(2 \leq ip\_ttl\_null \leq 4) \quad &\wedge \\
(ip\_ttl\_pos == 0) \quad &\Longrightarrow \quad (2 \leq ip\_id\_pos \leq 4) \\
(2 \leq ip\_ttl\_null \leq 4) \quad &\wedge \\
(ip\_ttl\_pos == 0) \quad &\Longrightarrow \quad (0.20 \leq tcp\_flags\_syn\_percent \leq 0.33) \\
(3 \leq size \leq 5) \quad &\wedge \\
(0 \leq tcp\_acknum\_null \leq 2) \quad &\wedge \\
(0 \leq tcp\_acknum\_pos \leq 1) \quad &\Longrightarrow \quad (2 \leq ip\_id\_pos \leq 4)
\end{aligned}
$$

A.8 TCP Port 80

A.8.1 1 - NTInfoscan

$$
\begin{aligned}
(1 \leq ip\_id\_pos \leq 3) \quad &\wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad &\wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad &\Longrightarrow \quad (1 \leq tcp\_urg\_null \leq 3) \\
(1 \leq ip\_id\_pos \leq 3) \quad &\wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad &\wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad &\Longrightarrow \quad (1 \leq ip\_offset\_null \leq 3) \\
(1 \leq ip\_id\_pos \leq 3) \quad &\wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad &\wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad &\Longrightarrow \quad (1 \leq ip\_ttl\_null \leq 3) \\
(1 \leq ip\_id\_pos \leq 3) \quad &\wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad &\wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad &\Longrightarrow \quad (1 \leq ip\_tos\_null \leq 3) \\
(1 \leq ip\_id\_pos \leq 3) \quad &\wedge \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad &\wedge \\
(0 \leq tcp\_acknum\_null \leq 1) \quad &\Longrightarrow \quad (2 \leq size \leq 4) \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad &\wedge \\
(1 \leq tcp\_seqnum\_pos \leq 2) \quad &\Longrightarrow \quad (1 \leq tcp\_urg\_null \leq 3) \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad &\wedge \\
(1 \leq tcp\_seqnum\_pos \leq 2) \quad &\Longrightarrow \quad (1 \leq ip\_offset\_null \leq 3) \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad &\wedge \\
(1 \leq tcp\_seqnum\_pos \leq 2) \quad &\Longrightarrow \quad (1 \leq ip\_ttl\_null \leq 3) \\
(0 \leq tcp\_seqnum\_null \leq 1) \quad &\wedge
\end{aligned}
$$

$$
\begin{aligned}
(1 \leq tcp\_seqnum\_pos \leq 2) &\implies (1 \leq ip\_tos\_null \leq 3) \\
(0 \leq tcp\_seqnum\_null \leq 1) &\quad \wedge \\
(1 \leq tcp\_seqnum\_pos \leq 2) &\implies (2 \leq size \leq 4)
\end{aligned}
$$

A.9 ICMP Echo Reply

A.9.1 2 and 35 - Ping of Death

$$
\begin{aligned}
(ip\_ttl\_null > 0) &\implies (ip\_foffset\_null > 0) \\
(size > 1) &\implies (ip\_foffset\_null > 0) \\
(ip\_id\_pos > 0) &\implies (ip\_foffset\_null > 0) \\
(ip\_id\_step\_mean > 0.0) &\implies (ip\_foffset\_null > 0) \\
(ip\_tos\_null > 0) &\implies (ip\_foffset\_null > 0) \\
(ip\_foffset\_null == 0) &\implies (ip\_ttl\_null == 0) \\
(ip\_foffset\_null == 0) &\implies (ip\_id\_step\_mean == 0.0) \\
(ip\_foffset\_null == 0) &\implies (ip\_id\_range == 0) \\
(ip\_foffset\_null == 0) &\implies (ip\_id\_pos == 0) \\
(ip\_foffset\_null == 0) &\implies (ip\_tos\_null == 0) \\
(ip\_foffset\_null == 0) &\implies (size == 1) \\
(ip\_id\_range > 0) &\implies (ip\_foffset\_null > 0) \\
(ip\_id\_step\_variance > 0.0) &\implies (ip\_foffset\_null > 0) \\
(ip\_foffset\_null == 0) &\implies (ip\_id\_step\_variance == 0.0)
\end{aligned}
$$

VITA

VITA

James Patrick Early was born on May 4, 1959 in Philadelphia, Pennsylvania to Timothy and Martha Early. He attended Central High School of Philadelphia and graduated with the 236th class earning Barnwell academic honors. In May of 1982, he joined MCI Telecommunications and spent the next sixteen years working in network operations and engineering. While at MCI, he was a recipient of the Executive Council Award (1992) and Peak Achiever Award (1997). He earned his Bachelor of Science degree in Computer Science from Towson State University in 1996. He was the 1996 recipient of the Mary Hudson Scarborough Award to the university's outstanding Computer Science undergraduate. He earned his Master of Science degree in Computer Science from Purdue University in 2001. In 2003, he was awarded a Ph.D. fellowship from the Intel Corporation based on his research in network intrusion detection.