5-2008

Machine Learning for Host-based Anomaly Detection

Gaurav Tandon

Machine Learning for Host-based Anomaly Detection

by
Gaurav Tandon

A dissertation submitted to
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Science

Melbourne, Florida
May, 2008
TR-CS-2008-01

Machine Learning for Host-based Anomaly Detection
a dissertation by
Gaurav Tandon

Approved as to style and content

Philip K. Chan, Ph.D.
Associate Professor, Computer Science
Dissertation Advisor

Debasis Mitra, Ph.D.
Associate Professor, Computer Science

Marius C. Silaghi, Ph.D
Assistant Professor, Computer Science

Georgios C. Anagnostopoulos, Ph.D.
Associate Professor, Electrical and Computer Engineering

William D. Shoaff, Ph.D.
Associate Professor and Program Chair
Computer Science

ABSTRACT

Machine Learning for Host-based Anomaly Detection

by

Gaurav Tandon

Dissertation Advisor: Philip K. Chan, Ph.D.

Anomaly detection techniques complement signature based methods for intrusion detection. Machine learning approaches are applied to anomaly detection for automated learning and detection. Traditional host-based anomaly detectors model system call sequences to detect novel attacks. This dissertation makes four key contributions to detect host anomalies. First, we present an unsupervised approach to clean training data using novel representations for system call sequences. Second, supervised learning with system call arguments and other attributes is proposed for enriched modeling. Third, techniques to increase model coverage for improved accuracy are presented. Fourth, we propose spatio-temporal modeling to detect suspicious behavior for mobile hosts.

Experimental results on various data sets indicate that our techniques are more effective than traditional methods in capturing attack-based host anomalies. Additionally, our supervised methods create succint models and the computational overhead incurred is reasonable for an online anomaly detection system.

Acknowledgements

An ancient saying in Sanskrit: **Matri devo bhava (revere your mother), pitri devo bhava (revere your father), acharya devo bhava (revere your teacher)**.

I am indebted to my parents for all their love and blessings, and imbibing the values of life in me. I appreciate all the sacrifices of my mother. She is a continuous source of inspiration for me. I value the lessons of life my father taught me growing up.

I am grateful to my advisor, Philip K. Chan, without whose guidance and support this dissertation would not have been possible. Phil has been a constant source of motivation. I hold his teachings in high regard. I also express gratitude to other committee members, Debasis Mitra, Marius C. Silaghi and Georgios C. Anagnostopoulos, for their insightful comments. Classes from Phil, Marius and Debasis increased my scope of knowledge for my research area. I thank all my teachers who encouraged me to think, question and reason.

Thanks to DARPA for partially supporting this research, and the Computer Science department head Bill Shoaff for providing financial support from the department.

I appreciate the input of all my colleagues at the Laboratory for Learning Research – Matt Mahoney, Hyoung-rae Kim, Rachna Vargiya, Stan Salvador and Muhammad Arshad. Matt also shared his LERAD algorithm source code. The reviewers of my paper submissions also provided valuable feedback. Kuntal Sengupta (Authentec, Mitsubishi Research Lab) provided valuable feedback for a problem we were trying to solve.

This work would not be complete without validating the techniques. I am thankful to all those who made the data sets publicly available – Richard Lippmann and colleagues at MIT Lincoln Labs, Stephanie Forrest et al. at University of New Mexico, James Whittaker (Florida

Institute of Technology, Microsoft), Alex Pentland and Nathan Eagle at MIT Media lab, and David Kotz and his colleagues at Dartmouth College. Geoff Mazeroff (University of Tennessee, Knoxville) clarified my doubts about the FIT-UTK data set. Nathan Eagle (MIT) was kind to answer all my questions regarding the Reality Mining data.

I acknowledge all my colleagues at Tegic/AOL/Nuance, who have been very encouraging. Brad Bargen, Bob Heddle and David Kay helped provide some support for my research through the company.

My aunt Prakash Kapur and uncle Ashok Tandon have constantly guided me, for which I am thankful.

I am grateful to my sisters Neera and Meera and brothers-in-law Narainder and Rajesh for their love and affection. They always stood by me and provided words of wisdom. The cherubic smiles of Nitin, Nikhil, Saahil and Alisha will always be cherished.

This dissertation would not have been possible without my wife Rachna. Her love and wholehearted dedication was instrumental in keeping me focused and determined to keep going.

Dedication

To my parents, and my wife Rachna.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer security research has two major aspects: intrusion prevention and intrusion detection. Intrusion prevention deals with preventing the occurrence of an attack using authentication and encryption techniques. Intrusion detection, on the other hand, focuses on the detection of successful breach of security. Together, these complementary approaches assist in creating a more secure system.

Intrusion detection systems (IDSs) are generally categorized as misuse-based and anomaly-based. In misuse (signature) detection, systems are modeled upon known attack patterns and the test data is checked for occurrence of these patterns. Examples of signature-based systems include virus detectors that use known virus signatures and alert the user when the system has been infected by the same virus. Such systems have a high degree of accuracy but suffer from the inability to detect novel attacks. Anomaly based intrusion detection (Denning 1987) models normal behavior of applications and significant deviations from this behavior are considered anomalous. Though anomaly detection can detect novel attacks, it has the drawback of not being able to discern intent; it can only signal that some event is unusual, but not necessarily hostile, thus generating false alarms.

Intrusion detection systems can also be categorized as network-based and host-based. As the name suggests, network-based systems monitors network traffic. They sniff data over the network and rules can be hard coded for valid IP addresses and ports. Machine learning algorithms have been used to learn automated rules for various protocol headers. Host-based systems reside in the host itself and typically monitor operating system events, such as system call sequences. This dissertation focuses on host-based anomaly detection.

1.1 Motivation and Problem Statement

There are three focal issues that need to be addressed for a host-based anomaly detection system: cleaning the training data, devising an enriched representation for the model(s), and ensuring sufficient coverage to improve accuracy. All these issues try to improve the performance of an anomaly detection system in their own ways. Additionally, there are security issues targeted at mobile hosts that need to be addressed.

First, traditional techniques that monitor system call sequences rely on *clean* training data to build their model. Current audit sequence is then examined for anomalous behavior using some supervised learning algorithm. An attack embedded inside the training data would result in an erroneous model, since all future occurrences of the attack would be treated as normal. Moreover, obtaining clean data by hand could be tedious. Purging all malicious content from audit data using an automated technique is hence imperative.

Second, normal behavior has to be modeled using features extracted from the training set. Traditional host-based anomaly detection systems focus on system call sequences to build models of normal application behavior. These techniques are based upon the observation that a malicious activity results in an abnormal (novel) sequence of system calls. Recent research (Wagner and Soto 2002; Tan and Maxion 2002) has shown that sequence-based systems can be compromised by conducting mimicry attacks. Such attacks are possible by

astute execution of the exploit by inserting dummy system calls with invalid arguments such that they form a legitimate sequence of events, thereby evading the IDS. A drawback of sequence-based approaches lies in their non-utilization of other key attributes, namely the system call arguments. The efficacy of such systems might be improved upon if a richer set of attributes (return value, error status and other arguments) associated with a system call is used to create the model.

Third, high coverage over training data is desired for a learned model. Assuming that training data is representative of the test data, high coverage is instrumental for high accuracy. But forcing complete data coverage results in overfitting leading to high false alarm rates. It is thus imperative to fine tune the model to maximize coverage without loss in accuracy. Searching the hypothesis space for alternate hypothesis and maintaining a distinction between strong and weak learners might result in the desired configuration of high coverage but not overfitting the model on the training data.

Fourth, hosts can be distinguished as static or mobile. There are certain security issues that only apply to mobile hosts, such as data theft and misuse due to lost or stolen device, and MAC spoofing in 802.11 wireless local area networks. Anomaly detection techniques for static hosts are generally not applicable for such problems. Possible solutions may need to model properties that are characteristic of mobile hosts, such as movement.

The problem under study thus involves an automated system to detect mal-intent on static as well as mobile hosts.

1.2 Approach

The first problem is that of data cleaning. We invalidate the assumption of the availability of attack-free data. Attack-based anomalies need to be purged to make the data suitable for training by supervised learning anomaly detection algorithms. We represent a system call

sequence using patterns (called motifs) and their positions. Further, we map all sequences to a single representation and use an unsupervised algorithm to detect and remove outliers for a clean training data. We also study the effect of filtering the data.

Next, we propose using system call attributes for a richer learned model. We learn multiple representations for system calls and their attributes using a supervised learning algorithm. We evaluate and compare them with traditional methods on multiple data sets. We also present an analysis of the anomalies detected by our techniques.

Further, we question the pruning of learned model and its effect on system accuracy. We argue that pruning may result in loss of coverage, thereby reducing the detection rate. We present techniques to increase coverage for a supervised rule learning algorithm. We list key aspects of rule quality. We propose rule weighting that is based on rule belief, rule replacement that replaces pruned rules for improved coverage, and a hybrid approach that chooses the technique with higher coverage. We also study the effect of coverage on accuracy.

Finally, we model the spatio temporal contextualities for mobile hosts. The underlying assumption is that the user would generally be at the same place around the same time. Any misuse could result in abnormalities in these patterns. We learn a spatio-temporal model for mobile hosts. We also present context clustering for concise models, and evaluate our technique on real data sets for mobile phone usage as well as 802.11 wireless local area networks. We present adaptive modeling to accommodate concept drift and curtail false alarms further.

1.3 Key Contributions

The main contributions are:

- two distinct representations for system call sequences (called motif space and sequence

space respectively), in conjunction with an unsupervised learning algorithm, effectively cleans training data free of attacks;

- enriched representations with system call attributes for improved modeling and higher accuracy with a supervised learning algorithm;
- coverage augmenting techniques for increased detection rates than pruning;
- spatio temporal anomaly detection for mobile hosts to detect misuse.

1.4 Dissertation Organization

This dissertation is organized as follows. Chapter 2 reviews existing literature in the domains of anomaly detection, machine learning and context aware mobile computing. In Chapter 3 we present motif-based representations and use an unsupervised learning algorithm for data cleaning. Chapter 4 presents argument-based representations and supervised learning algorithm for anomaly detection. We propose coverage augmenting techniques in Chapter 5 and demonstrate its effectiveness in improving accuracy. Chapter 6 presents spatio temporal anomaly detection for mobile hosts. We summarize our contributions and conclude in Chapter 7.

Chapter 2

Related Work

Machine learning and data mining techniques have been applied to learn system behavior. Some systems use attack-free or labeled training data for training (supervised learning) whereas others have no prior knowledge of the data (unsupervised learning). Anomaly detection systems can be deployed at the network level, where network traffic is monitored, as well as the host level, which involves operating system events. Hosts can be static (fixed), like desktop computers, or mobile, such as laptops, PDAs and mobile phones. This chapter reviews existing techniques in host-based anomaly detection, machine learning and context aware mobile computing.

2.1 Host-based anomaly detection

System call sequences have primarily been used for fixed-host-based anomaly detection using techniques like n-gram databases, neural networks, finite state automaton, sequence alignment and rule learning. Besides system calls, other features include system call arguments, Windows registries and call stack information. This section differentiates techniques based on the

features used.

2.1.1 System call information

System call sequences have been used effectively for host based intrusion detection. Time-delay embedding (*tide*) (Forrest et al. 1996) used a sliding look-ahead window of a fixed length to record correlations between pairs of system calls. These correlations were stored in a database of normal patterns, which was then used to monitor sequences during the testing phase. Anomalies were accumulated over the entire sequence and an alarm was raised if the anomaly count exceeded the threshold. *tide* forms correlations between pairs of system calls within a certain preset window size. Parallel work (Lane and Brodley 1997a; 1997b) comprised scanning of UNIX command sequences to capture user profiles. Degree of similarity between two different audit sequences was calculated by looking into adjacent events within a fixed size window. Sequence time-delay embedding (*stide*) (Warrender, Forrest, and Pearlmutter 1999) extended *tide* by memorizing all contiguous sequences of predetermined, fixed lengths. An anomaly count was defined as the number of mismatches in a temporally local region. A threshold was set for the anomaly score above which a sequence is flagged anomalous, indicating a possible attack. An extension, called sequence time-delay embedding with frequency threshold ($t - stide$), was similar to stide with the exception that rare sequences were ignored.

Extensions to fixed length sliding windows were provided in (Wespi, Dacier, and Debar 1999; 2000), wherein a scheme was proposed to generate variable length patterns by using Teiresias (Rigoutsos and Floratos 1998), a pattern-discovery algorithm in biological sequences. Sequence was deemed as anomalous when the pattern coverage for the audit sequence was below a threshold. (Jiang, Hua, and Sheu 2002) extended the variable length pattern idea by taking into account both the intra-pattern and the inter-pattern anomalies for detecting

intrusions. Maximal patterns were generated and adjacency lists created for all legitimate paths traversed. A counter was used for all mismatches. An alarm was raised when the counter exceeded the threshold.

Artificial neural networks (ANNs) were employed for both anomaly and misuse detection (Ghosh and Schwartzbard 1999). A backpropagation network was implemented to classify novel inputs based on similarity of known data. A leaky bucket algorithm emphasized temporally co-located anomalies and an alarm was raised beyond a threshold. In (Sekar et al. 2001), each system call was noted alongwith the program counter at the time of the call. The program counter represented the state of the finite state automaton FSA whereas the system call corresponded to the transition in the FSA. During runtime, the matched state was captured by the current state of the FSA. The existence of transition from the current state to the new state was verified, failure of which led to an anomaly.

Probabilistic suffix trees were created from system call sequences in (Mazeroff et al. 2003). Probabilities were associated with every transition. Trees were compared based upon the individual nodes and cumulative probability was computed. An automaton was created using the suffix tree with the suffix as the nodes and the associated probabilities along the transitions. The lower the probability of a new trace, the higher was its anomaly score.

A sequence alignment technique from bioinformatics (Smith-Waterman Algorithm) was adopted in (Coull et al. 2003) to compute semi-global alignment between system call sequences. Scores were computed by the dynamic programming algorithm. A matrix of size $m \times n$ for sequences of length m and n respectively was created. Each element in the matrix had a score for two sequences being aligned. Local constraints were applied to compute the scores at every position. Reward was given for a match and penalty was applied for inserting a gap in either sequence.

Text categorization technique was adopted for intrusion detection in (Liao and Vemuri

2002). Program behavior was characterized using system call frequencies. A word-by-document matrix is used for a collection of documents. k-nearest neighbor classifier algorithm ranked the program's neighbors and used the class labels of the k most similar neighbors to predict the class of the current program. Similarity was calculated using Euclidean distance and a cutoff threshold was used to assign the new document to a known class.

2.1.2 System call and related features

All the above mentioned techniques for host based systems used only system call information. (Wagner and Soto 2002) modeled malicious sequences by adding "no-ops" (system calls having no effect). Even though an original malicious sequence might not be accepted by an IDS, a modified version with no-ops was shown to evade an IDS. Some variants (like combining a series of system calls into a single one or using replacing one with another) were used to achieve an attacker's goals of converting a malicious sequence into an acceptable one.

Individual system calls and their respective arguments were examined in (Kruegel et al. 2003). For each system call, a profile is created. The models were based upon string length, character frequency distribution, structural information (for strings) using Markov models and Bayesian probability, and token finding.

2.1.3 Non system call features

Anomaly detection using Windows registries was demonstrated in (Apap et al. 2002; Stolfo et al. 2004). They used a technique similar to (Mahoney and Chan 2002). The process, query, key, response and result value formed the feature space. Single attribute values as well as pairs were checked for consistency. (Shavlik and Shavlik 2004) used a wide variety of system features including processor time, working set size and number of semaphores. A Winnows (Littlestone 1988) based variant was used to learn useful attributes for anomaly detection.

The authors in (Feng et al. 2003) proposed a method that dynamically extracted return address information from the call stack and used it for anomaly detection. Program counter information was recorded at each system call and return addresses from call stack were stored into a virtual stack list. The concept of virtual path abstracts the execution between two system calls. The virtual stack list was traversed to determine the control flow. The return address and virtual path tables were used for online detection.

2.2 Machine learning

Machine learning techniques are categorized as supervised and unsupervised. Supervised learning uses labeled data for training whereas there is no prior knowledge of data labels in unsupervised learning.

2.2.1 Supervised learning

An association rule mining algorithm called Aprioiri was proposed in (Agrawal and Srikant 1994). Generally used for market basket analysis, all rules that exceeded a user defined support and confidence were added to the ruleset. Set of items exceeding the minimum support were collected. Rules of the form $\{s => I$ - $s\}$ were generated for each subset s of the itemset I that exceeded the minimum confidence for the transactions in the database.

Rule learning is an important aspect of supervised learning. Incremental Reduced Error Pruning (IREP) (Furnkranz and Widmer 1994) adopted a greedy strategy that added one rule at a time to the ruleset. The conditions that maximized the information gain were added to the rule until the rule covered no negative examples from the growing dataset. Too specific rules resulted in overfitting the training data. Hence rule pruning was adopted. Pruning consisted of deleting conditions that maximized the error. After a rule was found, all (positive and negative) examples that were covered by the rule were deleted. These steps

were repeated till no more positive examples were left or the rule found had a large error rate.

Another greedy separate-and-conquer strategy for forming propositional rules was presented in (Cohen 1995). The algorithm, called RIPPER (Repeated Incremental Pruning to Produce Error Reduction), extended IREP by introducing support for missing attributes, numerical variables and multiple classes. An alternative rule value metric, a stopping heuristic for addition of further rules based on the description length and a postprocessing rule optimization phase were introduced. SLIPPER (Cohen and Singer 1999) improved the efficiency of RIPPER by introducing a boosting mechanism. SLIPPER generated a weighted ruleset. Examples covered by a rule were not immediately removed but were given lower weights.

A set of instance-based learning (IBL) algorithms were defined in (Aha, Kibler, and Albert 1991) with a similarity function to determine the proximity between two instance, an instance selection function for selecting examples from instances, and a classification function that determined the relation of a new case with learned cases. IBL1 stored all example instances and computed the closest instance. IBL2 discarded instances in the training set that were not correctly classified. IBL3 made assumptions about the data and uses statistical methods to weed out irrelevant or noisy instances. Two common instance-based learning techniques are the nearest neighbor pattern classification (Cover and Hart 1967) and the k nearest neighbor approach. The former took into only the nearest neighbor whereas the latter considered a collection of the k nearest points and used a voting mechanism to select between them.

Time series modeling has also generated a lot of interest in the data mining community. (Keogh and Pazzani 2000) introduced a new algorithm called Piecewise DTW (PDTW) wherein data was reduced into equal sized frames. The mean value of the data falling within a frame was calculated and this average value was used to represent all data within that frame. This resulted in a piecewise constant approximation of the entire sequence and a speed up in the execution time. A local derivative of the data was used in DTW instead of the raw data

in (Keogh and Pazzani 2001). Warping using slope tend to map corresponding peaks and valleys correctly between the two time series and was not effected by differences in the y-axis values between the two time series.

Many data mining and machine learning techniques are based on batch learning, where accumulated sets of instances are used for training. Typically, an online, incremental system is required which can update the model with every instance. In (Blum 1997), variants of the weighted-majority (Littlestone and Warmuth 1994) and Winnows (Littlestone 1988) algorithms were used to learn and predict certain attributes of the Calendar APprentice (CAP) data (Mitchell et al. 1994), which comprised of calendar information for two different users. The underlying idea was to use pairs (or triples) of distinct attribute values and the predictions in the last k times the attribute values were present together in an instance. All the applicable experts voted upon the arrival of a new instance and a global algorithm predicted the final outcome as the ones with maximum votes. Incorrectly predicting experts were penalized by reducing their weights to half. For Winnows, if an expert predicted correctly even though the global algorithm did not, the weight of the expert was increased by 50% of their current value. A generic dynamic weighted majority algorithm to learn concept drift was presented in (Kotler and Maloof 2003). Normalized weights were used by the experts to vote and a global algorithm aggregated these votes to predict the best outcome.

Associating weights with rules attempts to characterize the quality of the rules. One aspect of quality is *predictiveness*, which quantifies how likely the consequent occurs when the antecedent is observed; that is, how accurately the antecedent predicts the consequent. *Predictiveness* is commonly measured by estimating $P(consequent|antecedent)$. Another aspect of quality is *belief*, which measures the level of trust for the rule; that is, how believable the entire rule is. For example, in association rules (Agrawal and Srikant 1994), each rule has a confidence value and a support value — the confidence value estimates *predictiveness*,

while the support value approximates *belief*. Many learning algorithms, including RIPPER (Cohen 1995) and CN2 (Clark and Niblett 1989), use *predictiveness* to formulate rules during the learning (training) process and/or provide confidence values for their predictions during the prediction (test) process. Ensemble methods, including Weighted Majority (Littlestone and Warmuth 1994) and Boosting (Schapire 1990; Freund and Schapire 1996), use *belief* to combine predictions from multiple learned models. Pruning is an approach to reduce overfitting the training data. After learning a decision tree and converting each path in the tree into rules, Quinlan (Quinlan 1993) removes conditions from the antecedent of a rule if the estimated accuracy improves. Furnkranz (Furnkranz 1997) has a review of various rule pruning techniques. For rule learning algorithms, many studies demonstrate the efficacy of using weights (*predictiveness* and/or *belief*) over not using weights as well as pruning over not pruning. However, we are not aware of studies in comparing using weights and pruning, particularly in anomaly detection.

Rules can generally be learned in two ways: generate and test strategy vs. data driven approach. Generate and test adopts a hypothesis driven approach, where a general to specific search is usually performed in the hypothesis space. CN2 (Clark and Niblett 1989) is a general to specific beam search algorithm that maintains the k best candidates at each step, where rules are refined based on their accuracy on the data set. Apriori (Agrawal, Imielinski, and Swami 1993) learns all association rules above user defined confidence and support thresholds. It is also a generate and test approach, with only a subset of rules generated at each step are considered for specialization in the subsequent iteration. Using a general-to-specific search, ITRULE (Smyth and Goodman 1991; 1992) learns k (user-specified) rules that have the highest information content based on the J-measure (mutual-information based). Different candidate rules are formed with each attribute as the consequent. According to information-theoretic bounds, specializations of the current rules are not explored if they will not yield

higher information content than the top k rules found so far. Contrary to generate and test approach, hypotheses generation is constrained by specific data instances in data driven approach. AQ15 (Michalski et al. 1986) generates a rule to cover a specific attribute value, and these rules are specialized at each step. After each rule, the algorithm picks another attribute value not covered to initiate search in the hypothesis space.

2.2.2 Unsupervised learning

Clustering techniques are categorized as partitioning and hierarchical. Partitioning clustering techniques partition the data into clusters by using some optimizing criterion. Centroid based approaches, like k-means, and medioid based approaches (k-medioids) are examples. But they fail on data where points of a cluster are closer to the center of another cluster.

Hierarchical techniques have a single cluster at top of hierarchy, and clusters with one point each at the bottom. Agglomorative hierarchical techniques start from the bottom and work their way up to the top. CURE (Guha, Rastogi, and Shim 1998) used a number of representative points to represent a cluster instead of the center. Cluster similarity was then measured by using the closest representative points across the two clusters. Shrinking factor was used to converge towards center to overcome the effect of outliers. This technique was shown to successfully find clusters of arbitrary shapes and sizes. ROCK (RObust Clustering using linKs) (Guha, Rastogi, and Shim 2000) took into account the aggregate inter connectivity across clusters. Links were defined as the number of common neighbors between points. A random sample set was selected and a hierarchical algorithm used to cluster them. The clusters merged were the ones that had the maximum inter connectivity. Remaining samples were then assigned to these clusters. Chameleon (Karypis, Han, and Kumar 1999) used both inter connectivity and closeness. It determined the neighborhoods dynamically. Smaller clusters were merged only if the relative interconnectivity and relative closeness functions were

maximized.

Outlier detection techniques are generally taxonomised as distance and density based. (Ramaswamy, Rastogi, and Kyuseok 2000) found n topmost outliers based upon distance from the k^{th} nearest neighbor. A partition based algorithm was proposed which divided the input into clusters. Bounds were computed for every point in each partition. Parititions that did not contain the top n outliers were pruned and outliers were obtained from the remaining partitions.

LOF (Breunig et al. 2000), a density based outlier finding algorithm, focused on the detection of local outliers. A concept of a local neighborhood was defined, and each object was assigned a score (for being an outlier) based upon its density as compared to the density of its neighbors.

2.3 Context aware mobile computing

(Chen and Kotz 2000) define context as a set of environmental states and settings that either determines an application's behavior or in which an application event occurs and is interesting to the user. Context aware applications use data from low level sensors to extract features providing useful information. Contexts are associated with these features. Actions are then learnt using some machine learning algorithm on features from one or multiple sensors. (Ailisto et al. 2002) model context aware applications into 5 layers:

- sensor - getting raw data
- data processing object - obtain useful information from data (features)
- semantic processing object - conversion to meaningful form for inferring context
- inference object - learning algorithm

- application - action, service or other usable result

Active badge system (Want et al. 1992) used IR-based badges to determine the location of an employee and forward his call to the nearest telephone. The badge worn by the person emitted IR signals at regular intervals. These signals were noted by sensors placed at various positions within the building and the information passed on to a centralized system which then regulated the calls accordingly. Conference Assistant (Dey et al. 1999) gathered interests of conference attendees by means of a form and directed them to talks related to their areas of interests. The system retrieved time and location of events, facilitated note-taking and presented information about the presentations.

A radio frequency system was proposed in (Bahl and Padmanabhan 2000) to determine user location based on triangulation of signal strengths from different base stations. Signal strength probability distributions and clustering for infrastructure LANs was studied in (Youssef, Agrawala, and Shankar 2003). A two level framework for plan recognition in an indoor RF-based wireless network was provided in (Yin, Chai, and Yang 2004). At the first level, a dynamic Bayesian network was used to predict user actions from raw signals. The network used signal strength information at the base stations and actual physical locations on the map. N-gram model was then used at the second level to infer goals from actions.

Unsupervised clustering and classification of contexts obtained from multiple sensors has been studied in (Flanagan, Mäntyjarvi, and Himberg 2002), and fuzzy low level contextual information was segmented to obtain high level contexts in (Himberg et al. 2001). Recognition of audio contexts using Hidden Markov Model (HMM) was investigated in (Clarkson and Pentland 1998) where experiments involving classification of different sounds were said to give promising results. (Korpipaa et al. 2003) learnt context using a naive Bayes approach using data from audio, acceleration and light sensors. Context classes were user defined and low level sensor data was labeled to derive high level information.

Technology for enabling awareness (TEA) [1] investigated various aspects of context awareness for mobile devices. They made use of several sensors (accelerometer, microphones, touch and temperature sensors amongst others) to gather different types of information. The status of the cell phone was determined by the current context. Context profiles were pre-defined and Kohonen Self Organizing Maps (SOMs) were used to to cluster the context information from the sensors (Van Laerhoven and Cakmakci 2000). A probabilistic finite state machine architecture (similar to a Markov chain) was used for prediction.

ContextPhone [2] used location and usage data from multiple sources within a cell phone. They provided services like information and media sharing with annotations to form context, apart from activity logging. A framework for learning locations of user interest and route prediction was proposed in (Laasonen, Raento, and Toivonen 2004). Locations were determined by cells that were frequented by the user and the ones where the user spent the most time. Transitions between locations were noted. Locations were clustered and route predictions were based upon frequency and time factors. Route clustering was also performed in (Laasonen 2005), where path similarity was computed using the Jaccard measure and similar poaths were merged. Subsequent location prediction was performed using conditional probability and chain rule, with the probabilities conditioned upon time of day, weekday and frequency.

Reality mining (Eagle and Pentland 2006), based upon ContextPhone, took into account the proximity, time and location information to analyze the evolution of social networks. A Hidden Markov Model (HMM) based upon the hour of day as well as weekday/weekend criterion was used alongwith an Expectation Maximization algorithm to determine user behavior. High level location classes were used. A Gaussian mixture model was created to detect the proximity patterns between users and correlate the patterns with relationship types (obtained from user surveys). Since bluetooth devices are capable of device discovery, it enabled a de-

[1] http://www.teco.edu/tea

[2] http://www.cs.helsinki.fi/group/context

vice to passively scan for nearby devices and meet up with people sharing similar interests. Such an idea has been applied (using a similar Gaussian mixture model) for dating and match making services in Serendipity (Eagle and Pentland 2005).

A technique to detect mobile phone *cloning* fraud is proposed in (Fawcett and Provost 1997), where patterns of fraud are learned and adapted to the user call behavior. Anomaly detection has also been used to detect outliers in spatial data (Adam, Janeja, and Atluri 2004), where neighborhood relationships are modeled and outliers are identified. Suspicious large moving objects, such as ships, have been detected as anomalies (Li et al. 2007). Though it involves route modeling, it deals with additional attributes like speed and direction information generally not available on laptops and mobile phones. Even if a GPS (global positioning system) was available on these devices, an intruder will likely disable it to evade such systems.

Chapter 3

Data Cleaning

Traditional host based anomaly detection techniques create models of normal behavioral patterns and then look for deviations in test data. Such techniques perform supervised learning that require *clean* or labeled training data to build models of normal behavior, which is hard to obtain. Any amount of data collected cannot be guaranteed to be free of malice. The data sets available are generated in constrained environments. Other evaluations use synthetic data sets. Such data sets are not representative of actual application behavior, which contains many irregularities. The need for a system to filter audit data and produce a *clean* data set motivates our current research.

Unsupervised learning is an extensively researched topic in network anomaly detection (Portnoy 2000; Eskin et al. 2002; Chan, Mahoney, and Arshad 2003; Lazarevic et al. 2003). Network traffic comprises continuous and discrete attributes which can be considered along different dimensions of a feature space. Distance and density based algorithms can then be applied on this feature space to detect outliers. Due to the lack of a similar feature space, not much work has been done using unsupervised learning techniques in host based systems.

From the modeling/detection point of view, all the above mentioned approaches for host-

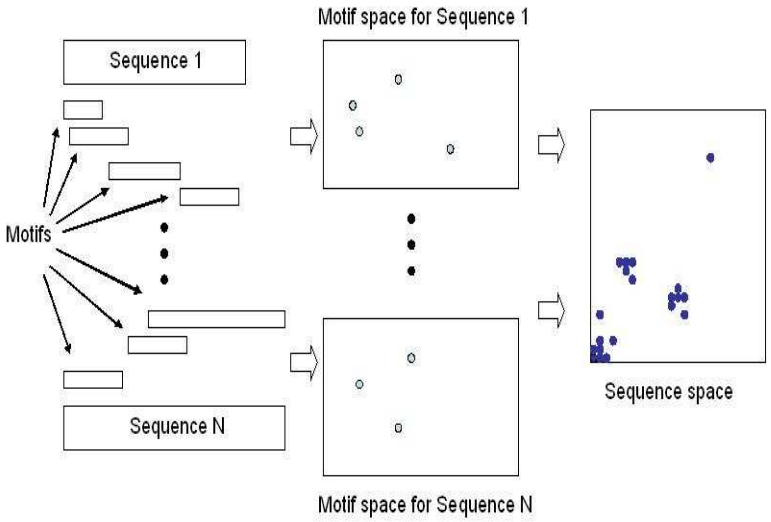

Figure 3.1: MORPHEUS approach

based systems use system call sequences. Parameter effectiveness for window based techniques has been studied in (Tan and Maxion 2002). Given some knowledge about the system being used, attackers can devise some methodologies to evade such intrusion detection systems. Wagner and Soto (Wagner and Soto 2002) modeled a malicious sequence by adding *no-ops* (system calls having no effect) to compromise an IDS based upon the sequence of system calls. Such attacks would be detected if the system call arguments are also taken into consideration, and this provides the motivation for our work.

In this chapter, we present a technique for data cleaning. The main steps are depicted in Fig. 3.1. First, we represent system call sequence based on certain patterns called motifs and their positions within the sequence (Tandon, Mitra, and Chan 2004). This representation is called the motif space. Further, we represent sequences in a single feature space called sequence space and refine the data set offline by purging anomalies using an unsupervised learning technique on the feature space (Tandon, Chan, and Mitra 2004; 2006).

3.1 Definitions

System call sequence Let Σ be a finite set of all distinct system calls. A system call sequence (*SCS*) s is defined as a finite sequence of system calls and is represented as (c_1 c_2 c_3 ... c_n), where $c_i \in \Sigma, 1 \leq i \leq n$.

After processing the audit data into process executions, system call sequences are obtained as finite length strings. Each system call is then mapped to a unique symbol using a translation table. Thereafter, they are ranked by utilizing prior knowledge as to how susceptible the system call is to malicious usage. A ranking scheme similar to the one proposed by Bernaschi et al. (Bernaschi, Gabrielli, and Mancini 2001) was used to classify system calls on the basis of their threat levels.

Motif A *motif* is defined as a subsequence of length greater than p if it appears more than k times, for positive integers p and k, within the finite set $S = \{s_1, s_2, \ldots, s_m\}$ comprising m *SCS*s. Motif discovery has been an active area of research in bioinformatics, where interesting patterns in amino and nucleic acid sequences are studied. Since motifs provide a higher level of abstraction than individual system calls, they are important in modeling system call sequences.

3.2 Motif extraction

Two sets of motifs are extracted via *auto-match* and *cross-match*, explained next.

3.2.1 Auto-match

The set of motifs obtained through auto-match comprise frequently occurring patterns within each sequence. For our experiments, we considered any pattern at least 2 characters long, occurring more than once as frequent. While the set S of *SCS*s is the input to this algorithm,

a set of unique motifs $M = \{m_1, m_2, \ldots, m_q\}$ is the output. It may happen that a shorter subsequence is subsumed by a longer one. We prune the smaller motif only if it is not more frequent than a larger motif that subsumes it. More formally, a motif m_i extracted using auto-match (1) has length ≥ 2, (2) has frequency ≥ 2, and (3) if there exists a motif $m_j \in M$ in a sequence $s_k \in S$ such that m_i is a subsequence of m_j but occurs independently in SCS s_k.

To illustrate this idea, consider the following synthetic sequence

$$acggcggfgjcggfgjxyz \tag{3.1}$$

Note that in this sequence we have a motif *cgg* with frequency 3, and another motif *cggf* with frequency 2, which is longer and sometimes subsumes the shorter motif but not always. We consider them as two different motifs since the frequency of the shorter motif was higher than the longer one. The frequently occurring subsequences (with their respective frequency) are *cg*(3), *gg*(3), *gf*(2), *fg*(2), *gj*(2), *cgg*(3), *cggf*(2), *ggfg*(2), *gfgj*(2), *cggfg*(2), *ggfgj*(2), *cggfgj*(2). The longest pattern *cggfgj* subsumes all the smaller subsequences except *cg*, *gg* and *cgg* since they are more frequent than the longer pattern, implying independent occurrence. But *cg* and *gg* are subsumed by *cgg*, since they all have the same frequency. Thus, the final set of motifs M=$\{$*cgg*, *cggfgj*$\}$.

3.2.2 Cross-match

Apart from frequently occurring patterns, we are also interested in patterns which do not occur frequently but are present in more than one *SCS*. These motifs could be instrumental in modeling an intrusion detection system since they reflect common behavioral patterns across sequences (benign as well as intrusive). We performed pair-wise cross-match between different sequences to obtain these. In other words, a motif m_i extracted using cross-match (1) has length ≥ 2, (2) appears in at least a pair of sequences $s_k, s_l \in S$, and (3) is maximal,

i.e., there does not exist a motif $m_j \in M(j \neq i)$ such that $m_j \subseteq s_k, s_l$ and $m_i \subset m_j$. Let us consider the following pair of synthetic sequences:

$$acgfgjcgfgjxyzcg \tag{3.2}$$

$$cgfgjpqrxyzpqr \tag{3.3}$$

Using cross-match between the example sequences in Eqs. 3.2- 3.3, we get the motifs *cgfgj* and *xyz*, since these are the maximal common subsequences across the two given sequences.

A simple method for comparing amino acid and nucleotide sequences called the $MatrixMethod$ is described by Gibbs and McIntyre (Gibbs and McIntyre 1970). A matrix is formed with one sequence written across and the other in the downward position on the left of the matrix. Any common element was marked with a dot and a series of dots along a diagonal gave a common subsequence between the two sequences. Using a technique similar to the Matrix Method, motifs are extracted which occur across sequences but may not be frequent within a single sequence itself.

Motifs obtained for a sequence (auto-match) or pairs of sequences (cross-match) are added to the motif database. Redundant motifs are removed. Motifs are then ordered based upon the likelihood of being involved in an attack. The ranking for individual system calls is used here and motifs are ordered using dictionary sort. The motifs are then assigned a unique id based upon their position within the ordered motif database.

3.3 Motif Space: motif-based representation of a sequence

After collecting all the motifs that exist in the set S of sequences in the motif database M, we represent each sequence in terms of the motifs occurring within it. For each sequence $s_i \in S$, we list all the motifs occurring within it along with their starting positions within the sequence.

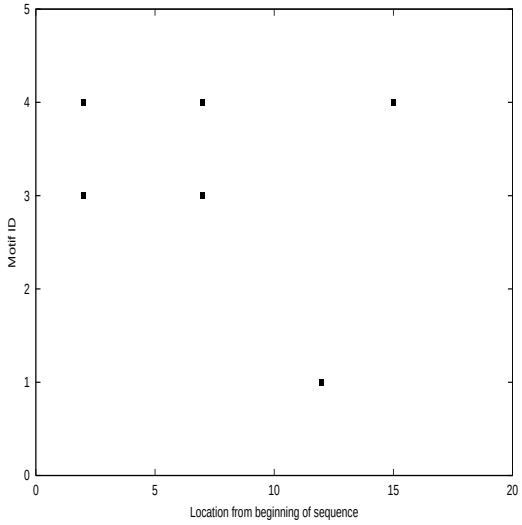

Figure 3.2: Motif-oriented representation for sequence in Eq. 3.2

This creates a two-dimensional representation for each *SCS* s_i, where the X-axis is the distance along the sequence from its beginning, and the Y-axis is the motif ID of those motifs present in s_i. A sequence can thus be visualized as a scatter plot of the motifs present in the sequence. Fig. 3.2 depicts such a representation for the synthetic sequence in Eq. 3.2, where the motifs *cg*, *cgfgj* and *xyz* are represented at the positions of occurrence within the respective sequence. A total of 4 unique motifs (*cg*, *cgfgj*, *pqr* and *xyz*), obtained from auto-match and cross-match of sequences in Eqs. 3.2- 3.3, are assumed in the motif database for the plot in Fig. 3.2. At the end of this phase, our system stores each *SCS* as a list of all motifs present within along with their spatial positions from the beginning of the sequence.

All the *SCS*s are modeled based upon the contained motifs. Malicious activity results in alterations in the *SCS* which is reflected by the variations in the motifs and their spatial positions. Plotting all the *SCS*s (based upon their motif-based representations) in a single feature space could reflect the similarity/dissimilarity between them.

3.4 Sequence Space: representing multiple sequences

After creating a motif-based representation for each sequence, all the test sequences S are plotted in a feature space called the *sequence space*. In this representation we measure the distance between pairs of *SCS*s along each of the two axes (motifs and their locations). Utilizing one (arbitrarily chosen) *SCS* from the set S as a reference sequence s_1, we measure $(d_x,\ d_y)$ distances for all *SCS*s. Thus, the sequences are represented as points in this *2D* sequence space, where the sequence s_1 is at the origin (reference point) on this plot. Let s_2 be any other sequence in S whose relative position with respect to s_1 is to be computed. Let x_{1_i} (x_{2_i}) be the position of the i^{th} motif in s_1 (s_2). Inspired by the symmetric Mahalanobis distance (Mahalanobis 1930), the distance is computed as follows:

$$d_x = \frac{\frac{\sum_{i=1}^{n_1}(x_{1_i}-\bar{x}_2)}{\sigma_{x_2}} + \frac{\sum_{j=1}^{n_2}(x_{2_j}-\bar{x}_1)}{\sigma_{x_1}}}{n_1+n_2}\ ;\ d_y = \frac{\frac{\sum_{i=1}^{n_1}(y_{1_i}-\bar{y}_2)}{\sigma_{y_2}} + \frac{\sum_{j=1}^{n_2}(y_{2_j}-\bar{y}_1)}{\sigma_{y_1}}}{n_1+n_2} \tag{3.4}$$

where s_1 has n_1 motif occurrences and s_2 has n_2 motif occurrences, $(d_x,\ d_y)$ is the position of s_2 w.r.t. s_1, and $(\bar{x},\ \bar{y})$ is the mean and $(\sigma_x,\ \sigma_y)$ is the standard deviation along the x and y axes. Using this metric, we try to calculate the variation in motifs and their locations in the two sequences.

After computing $(d_x,\ d_y)$ for all sequences in S with respect to the reference sequence (s_1), we plot them in the sequence space, as represented by the two plots in Fig. 3.3. The origin represents the reference sequence. It is important to note that the position of another sequence (calculated using Eq. 3.4) with respect to the randomly selected reference sequence can be negative (in X and/or Y direction). In that case the sequence space will get extended to other quadrants as well, as in Fig. 3.3b.

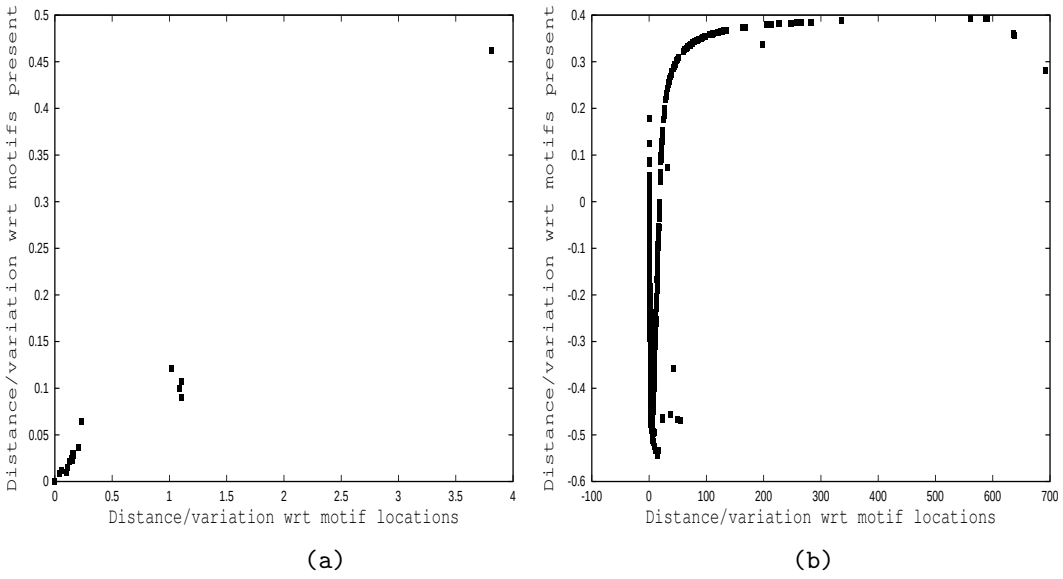

Figure 3.3: Sequence space for two applications (a) ftpd and (b) lpr

3.5 Unsupervised training with Local Outlier Factor (LOF)

Similar sequences are expected to cluster together in the sequence space. Malicious activity is known to produce irregular sequence of events. These anomalies would correspond to spurious points (global outliers) or local outliers in the scatter plot. In Fig. 3.3a, the point on the top-right corner of the plot is isolated from the rest of the points, making it anomalous. In this section we will concentrate on outlier detection, which has been a well researched topic in databases and knowledge discovery (Knorr and Ng 1998; Breunig et al. 2000; Ramaswamy, Rastogi, and Kyuseok 2000; Aggarwal and Yu 2001). It is important to note that an outlier algorithm with our representation is inappropriate for online detection since it requires complete knowledge of all process sequences.

LOF (Breunig et al. 2000) is a density-based outlier finding algorithm which defines a local neighborhood, using which a degree of outlierness is assigned to every object. The number of neighbors ($MinPts$) is an input parameter to the algorithm. A reachability density is

calculated for every object which is the inverse of the average reachability distance of the object from its nearest neighbors. Finally, a local outlier factor (LOF) is associated with every object by comparing its reachability density with each of its neighbors. A local outlier is one whose neighbors have a high reachability density as compared to that object. For each point this algorithm gives a degree to which that point is an outlier as compared to its neighbors (anomaly score). Our system computes the anomaly scores for all the *SCS*s (represented as points in sequence space). All the points for which the score is greater than a threshold are considered anomalous and removed.

We made some modifications to the original LOF algorithm to suit our needs. In the original paper (Breunig et al. 2000), all the points are considered to be unique and there are no duplicates. In our case, there are many instances when the sequences are exactly the same (representative of identical application behavior). The corresponding points would thus have the same spatial coordinates within the sequence space. Density is the basis of our system and hence we cannot ignore duplicates. Also, a human expert would be required to analyze the sequence space and suggest a reasonable value of $MinPts$. But the LOF values increase and decrease non-monotonically (Breunig et al. 2000), making the automated selection of $MinPts$ highly desirable. We present some heuristics to automate the LOF and threshold parameters, making it a parameter-free technique.

3.5.1 Automating the parameters

MinPts

To select $MinPts$, we use clustering to identify the larger neighborhoods. Then, we scrutinize each cluster and approximate the number of neighbors in an average neighborhood. We use the L-Method (Salvador and Chan 2004) to predict the number of clusters in the representation. This is done by creating a *number of clusters vs. merge distance* graph obtained from merging

one data point at a time in the sequence space. Starting with all N points in the sequence space, the two closest points are merged to form a cluster. At each step, a data point with minimum distance to another cluster or data point is merged. At the final step, all points are merged into the same cluster. The graph obtained has three distinct areas: a horizontal region (points/clusters close to each other merged), a vertical region (far away points/clusters merged), and a curved region in between. The number of clusters is represented by the knee of this curve, which is the intersection of a pair of lines fitted across the points in the graph that minimizes the root mean square error. Further details can be obtained from (Salvador and Chan 2004).

Assume k clusters are obtained in a given sequence space using L-Method (with each cluster containing at least two points). Let α_i be the actual number of points in cluster i, $1 \leq i \leq k$. Let ρ_i be the maximum pair-wise distance between any two points in cluster i; and τ_i is the average (pair-wise) distances between two points in cluster i. Let β_i be the expected number of points in cluster i. Its value can be computed by dividing the area of the bounding box for the cluster with the average area occupied by the bounding box of any two points in the cluster (for simplicity we assume square shaped clusters). Therefore, we get

$$\beta_i = \left(\frac{\rho_i}{\tau_i}\right)^2 \tag{3.5}$$

This gives us the expected number of points within the cluster. But the actual number of points is a_i. Thus, we equally distribute the excess points among all the points constituting the cluster. This gives us an approximate value for $MinPts$ (number of *close* neighbors) of the cluster i $(= \gamma_i)$

$$\gamma_i = \left\lceil \frac{\alpha_i - \beta_i}{\beta_i} \right\rceil \tag{3.6}$$

After obtaining $MinPts$ for all k clusters, we compute a weighted mean over all clusters to

obtain the average number of $MinPts$ for the entire sequence space.

$$MinPts = \left\lceil \frac{\sum_{i=1}^{k} \gamma_i \alpha_i}{\sum_{i=1}^{k} \alpha_i} \right\rceil \tag{3.7}$$

Only clusters with at least two points are used in this computation. But this approach gives a reasonable value for the average number of $MinPts$ in a sequence space if all the points are unique. In case of duplicates, Eq. 3.5 is affected since the maximum distance still remains the same whereas the average value is suppressed due to the presence of points with same spatial coordinates. If there are q points corresponding to a coordinate (x, y), then each of the q points is bound to have at least (q-1) $MinPts$.

Let p be the number of frequent data points (i.e. $frequency \geq 2$) in cluster i. Let ψ_j be the frequency of a data point j in cluster i. In other words, it is the number of times that the same instance occurs in the data. We compute γ' the same way as Eq. 3.6, where γ' is the $MinPts$ value for cluster i assuming unique points (no multiple instance of the same data point) in the sequence space.

$$\gamma_i' = \left\lceil \frac{\alpha_i - \beta_i}{\beta_i} \right\rceil \tag{3.8}$$

This value is then modified to accommodate the frequently occurring points (corresponding to sequences sharing the same spatial positions in the sequence space). We compute a weighted mean to obtain an appropriate value of $MinPts$ in cluster i as follows:

$$\gamma_i = \left\lceil \frac{\gamma_i' \alpha_i + \sum_{j=1}^{p} \psi_j(\psi_j - 1)}{\alpha_i + \sum_{j=1}^{p} \psi_j} \right\rceil \tag{3.9}$$

Average $MinPts$ for the entire plot can then be computed using Eq. 3.7.

Threshold

LOF only assigns a local outlier factor for a point in the sequence space which corresponds to its anomaly score. If the score is above a user specified threshold, then it is considered as anomalous and hence filtered from the data set. If the threshold is too low, there is a

risk of filtering a lot of points, many of which may depict normal application behavior. On the contrary, if the threshold is too high, some of the data points corresponding to actual intrusions (but close to many other data points on the sequence space) may not get filtered. We compute the threshold automatically by ordering the LOF scores and plotting them in increasing order (with each data point along the X-axis and the anomaly/LOF score along the Y-axis). Since the normal points are assumed in abundance, their LOF scores are ideally 1. We are interested in the scores after the first steep rise of this plot, since these correspond to outliers. Ignoring all the scores below the first steep rise (corresponding to normal sequences), the cut-off value can be computed as the median of all the scores thereafter. This heuristic gives a reasonable threshold value for the various applications in our data sets.

3.6 Training and online detection

The filtered data set obtained above provides clean data as training input to an online anomaly detection system like stide and LERAD. stide (Sequence TIme-Delay Embedding) [35] memorizes all contiguous sequences of predetermined, fixed length (n-grams) during training. This is done by using a sliding window and adding all unique sequences to the database. During test phase, an anomaly count is associated with n-gram mismatches and is defined as the number of mismatches in a temporally local region for a sequence. A threshold is set for the anomaly score above which a sequence is flagged anomalous, indicating a possible attack.

LERAD (LEarning Rules for Anomaly Detection) (Mahoney and Chan 2003) is a randomized algorithm that learns rules for the normal data set. With every rule a probability is assigned for encountering a novel value of the attribute in the consequent when the conditions in the antecedent are true. A non-stationary model is assumed for LERAD frequency is made irrelevant and only the last occurrence of an event is assumed important. The anomaly scoring function uses the probability and time since last anomaly of the rule violated by the

test input.

3.7 Experimental Evaluation

Our goal is to determine if our proposed representation can be used with an unsupervised learning algorithm (namely LOF) to detect and purge out anomalies, creating a clean training set for online detection systems. We would also like to note the change in performance after using our filtering scheme.

3.7.1 Data sets and preprocessing

We evaluated our techniques on seven applications obtained from three different data sets:

1. The DARPA intrusion detection evaluation data set was developed at the MIT-Lincoln Labs (Lippmann et al. 2000). We used the Solaris data from the Basic Security Module (BSM) audit logs. For our experiments, we selected the ftpd, ps, fdformat and eject applications to obtain a good range in the number of sequences and the number of system calls. These applications also have a good mix of different attack types (Kendell 1999). The ftpd application comprises of R2L (guessftp, ftpwrite) and DoS (warez, warezclient) attacks. On the other hand, ps, eject and fdformat are all U2R attacks.

2. Two applications (lpr and login) from the University of New Mexico (UNM) data sets (Warrender, Forrest, and Pearlmutter 1999) were also used. The lpr application comprised of 2703 normal traces running lpr collected from 77 hosts running SUNOS 4.1.4 at the MIT Artificial Intelligence Lab. Another 1001 traces correspond to the execution of the lprcp attack script. Older versions of lpr use only 1000 different names for printer queue files. The attack takes advantage of the fact that the old files are not removed from the queue before they can be used again. The attack works as follows: a symbolic

link is placed to the victim file at the beginning. All the intermediary traces increment the counter and the intruder overwrites the target file in the last trace. Traces from the login application were obtained from a Linux machine running kernel 2.0.35. A homegrown Trojan program was used for the attack traces.

3. We also used system call sequence logs corresponding to Microsoft excel macros in execution used by researchers at Florida Institute of Technology (FIT) (Whittaker and De-Vivanco 2002) and University of Tennessee at Knoxville (UTK) (Mazeroff et al. 2003). 36 normal traces correspond to statistical, chemistry and cost estimation related Excel macros. Two malicious traces modify the registry settings and execute some other application. Such a behavior is exhibited by the ILOVEYOU worm which opens the web browser to a specified website and executes a program, modifying registry keys and corrupting user files. This worm results in a distributed denial of service (DDoS) attack.

3.7.2 Outlier (anomaly) detection in sequence space

Our system creates a sequence space and plots all sequences as points with respect to other sequences. We claim that the malicious sequences are reflected as outliers in the sequence space. It is therefore imperative for us to evaluate if the outliers in the sequence space correspond to actual attacks. The underlying assumption is that the bulk of the data set constitutes of normal SCSs. We assume that the similar nature of normal behavior will cause them to cluster together. Outliers to these clusters would be the anomalies resulting from possible intrusions.

For the MIT-Lincoln lab data set, week 3 comprises of clean data while weeks 4 and 5 data has attacks. We are also given the timestamp for the occurrence of the attacks. After dividing the data into different applications and their processes, we combine the data for the 3 weeks together (on a per application basis) and feed it to our system. This gives a good mix

Table 3.1: Automated *MinPts* computation.

Application	eject	fdformat	ftpd	ps	lpr	login	excel
Number of sequences	21	19	91	341	3704	16	38
Average sequence length	66.43	57.63	184.93	66.45	835.73	730.81	2862.87
MinPts	3	2	10	71	6	3	2

of normal application behavior and some sequences resulting from intrusions. We also use an aggregation of all traces for the other data sets (UNM and FIT-UTK) on a per application basis for similar reasons.

We created a sequence space for each application. Fig. 3.3a represents the sequence space for the ftpd application from the DARPA evaluation data set, whereas Fig. 3.3b corresponds to the lpr data set from UNM. The X-axis on the plots is the distance due to the motif separation amongst sequences and Y-axis corresponds to the distance with respect to the motifs present in the sequence. Similar sequences tend to cluster together while anomalous sequences are represented as outliers.

We used the sequence space to detect local outliers using LOF on all the datasets. LOF takes MinPts-nearest neighbors (the number of points comprising the neighborhood of a point) as an input parameter and the results are very sensitive to this parameter selection. For our experiments, we varied this parameter value as a percentage of the entire population. We also used the MinPts value that we computed using our automated technique. These values are listed in Table 3.1. After computing the LOF or anomaly scores, we ranked them in descending order. All the sequences with scores greater than the threshold were considered anomalies and evaluated for detections and false alarms.

The results from the experiments, depicted in Table 3.2, indicate that none of the MinPts values were ideal to detect all the attacks. The two parameter values 15% and 20% seem

Table 3.2: True positives and false positives at varied LOF MinPts values.

Application	**Total Attacks**	**Number of different attacks detected (with false alarm count) for different values of LOF MinPts (% of total population)**				
		5%	10%	15%	20%	Automated (from Table 3.1)
eject	2	1 (1)	2 (1)	2 (0)	2 (0)	2 (0)
fdformat	3	3 (0)	3 (0)	3 (0)	3 (0)	3 (0)
ftpd	6	0 (6)	0 (11)	6 (6)	6 (1)	0 (11)
ps	4	0 (6)	4 (1)	4 (1)	4 (2)	4 (49)
lpr	1	0 (123)	1 (193)	1 (198)	1 (157)	1 (97)
login	1	0 (1)	0 (2)	1 (2)	1 (2)	1 (2)
excel	2	2 (0)	0 (3)	0 (0)	0 (0)	2 (0)
Total	**19**	**6 (137)**	**10 (211)**	**17 (207)**	**17 (162)**	**13 (159)**

to have the maximum number of detections (17 each, out of 19 total attacks). The only attacks missed were the ones in the excel application where a reasonable value of MinPts is best suggested as 5%. Our methodology for MinPts calculation was successful in computing the correct number for the parameter and hence successfully detected the attack sequence as outlier (for which the 15% and 20% values failed). The automated LOF parameter detected all the attacks except the ones in the ftpd application. The reason for such a behavior can be better understood from Fig. 3.3a. There are 2 main clusters in the sequence space one close to the origin and the other towards the center of the plot. The total number of points is 91 (80 in the large cluster, 10 in the smaller one, and one spurious point far away on the top-right corner of the plot). The MinPts value obtained by using our heuristic is 10, which seems to be an appropriate value. Inability of LOF to detect the anomalies in this representation is attributed to the fact that all the 10 points in the smaller cluster correspond to 6 different attacks. Therefore, the anomaly scores for all these points are very low. This implies that the concept of local outliers is not sufficient to capture such anomalous data points. Thus, we need to adopt a global view to find anomalous clusters as well, which can be incorporated in our sequence space. This would also be beneficial in detecting flooding attacks, which would typically correspond to high density points/clusters in the sequence space. Other than the ftpd application, the automated technique successfully detected all other attacks. This suggests that the MinPts values computed using our heuristic are generally reasonable.

As can be observed from Table 3.2, the number of false alarms is high for the lpr application (in the range 2.6-5.3%), which constitutes of over 3700 sequences and approximately 3.1 million system calls. The data was collected over 77 different hosts and represents high variance in application behavior. Though we were able to capture the lpr attack invoked by the lprcp attack script, we also detected other behavioral anomalies which do not correspond to attacks. Our goal here is to retain generic application behavior and shun anomalies. Peculiar

(but normal) sequences would also be deemed anomalous since they are not representative of the general way in which the application functions, as in this case. Since program faults and system crashes are not representative of normal behavior, purging these anomalies is justifiable.

Our representation scheme also subsumes the ideas presented in (Warrender, Forrest, and Pearlmutter 1999; Wespi, Dacier, and Debar 1999; 2000; Jiang, Hua, and Sheu 2002). The underlying assumption is that similar sequences would appear together in the sequence space. An attack modifies the course of events. This results in (a) either the absence of a motif, or (b) altered spatial positions of motifs within the sequence due to repetition of a motif, or (c) the presence of an entirely new motif. All these instances affect the spatial relationships amongst the different motifs within the sequence. Ultimately, this affects the distance of the malicious sequence with respect to the reference sequence, resulting in an outlier being plotted on the sequence space. It is this drift within the sequence space that the outlier detection algorithm is able to capture as an anomaly. Since the reference sequence is picked randomly, it may so happen that the reference sequence is the attack sequence itself. This does not affect our system since our distance metric is symmetric and the point is still classified as an outlier.

3.7.3 Effects of filtering the data

We reemphasize that our ultimate goal is to obtain clean data for other intrusion detection systems to train on. Thus, it is important to study how well our system can clean the training data and what effect does it have on the performance of an online detection system (in terms of true detections as well as false alarm generation).

Only the MIT-Lincoln Labs and FIT-UTK data sets were used for this set of experiments since they contained sufficient attacks to be used in both adulterated training and test data sets. The lpr and login data sets from UNM comprised of only a single attack. We have

already demonstrated in the previous section that our technique could filter them as spurious outliers. But a single attack is not sufficient for this set of experiments, as we would like to have attacks in both training and test data. Therefore, we did not involve those two applications for this experiment. We combined the clean week 3 data with the mixed week 4 data of the MIT-Lincoln lab data set to obtain an unlabeled data set. We use this to train stide and LERAD. We then tested on week 5 data (containing attacks with known timestamps). Subsequently, we used MORPHEUS to filter out spurious data points (and hence SCSs) from the combined data set. This marks the end of phase 5 of our system. The sixth and final phase is next, which uses the refined data set for training stide and LERAD. Week 5 data is used for testing purposes. As per the 1999 DARPA evaluation criteria, a system is considered to have successfully detected an attack if it generates an alarm within 60 seconds of the occurrence of the attack. We follow the same criterion for our evaluation. For the FIT-UTK Microsoft Excel data set, we randomly picked 33 traces (including one attack) for training and remaining 5 traces for testing purposes.

The parameter selection for our experiments was as follows: For stide, we used a window size of 6. A locality frame of 20 is used, that is the anomaly count keeps track of the number of mismatches in a temporally local region comprising 20 system calls. All the parameter values used are suggested to give best results in (Warrender, Forrest, and Pearlmutter 1999); parameter sensitivity is studied in (Tan and Maxion 2002). For LERAD, each tuple comprised of the system call, its return value and error status besides other arguments. In all cases, alarms are accumulated for the applications and then evaluated for the number of true detections and false positives.

Fig. 3.4 depicts the number of attacks detected by stide and LERAD for the 5 applications under study. It is observed that in both cases, the IDS was able to detect more attacks in ftpd and ps applications after data filtering by MORPHEUS while there was no change in the

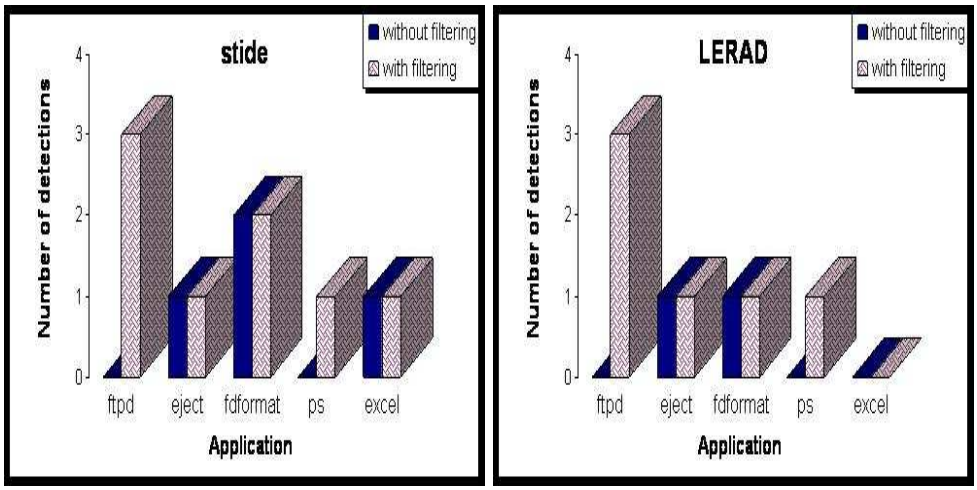

Figure 3.4: Comparison of attack detections with and without filtering, for (i) stide and (ii) LERAD.

performance for the other three applications (eject, fdformat and excel). This is because the training data also contained some attacks. For the ftpd and ps applications, the attacks in the test data were similar to the ones seen in the adulterated training data set, and were hence missed by both the IDSs. When filtered using MORPHEUS, the attack SCSs were purged and hence detections were possible in the test phase. For the applications eject and fdformat, the attacks in the adulterated training and testing data sets were different in character. Hence both the systems detected them irrespective of the filtering procedure. For excel, stide was able to detect the worm in both cases due to similar reasons. LERAD was not able to capture the malicious sequence due to incomplete argument information.

No false alarms were generated in any case in stide except the excel application, where one false positive was produced in each case. For LERAD, 1 false alarm was generated for the ps application with and without filtering. No other false positives were obtained. Overall, the results indicate that the filtering process was instrumental in increasing the number of detections without increasing the number of false alarms.

3.8 Summary

Most of the traditional host based IDSs require a clean training data set which is difficult to obtain. Our system, called MORPHEUS, addresses and attempts to solve this problem of data filtering. We present a motif-based representation for system call sequences (SCSs) based upon their spatial positions within the sequence. We also propose a novel representation of sequences called sequence space using a distance metric between the motif-based representations. We also exhibited the efficacy of this feature space to filter anomalies by integrating it with an existing unsupervised learning algorithm (called LOF) for outlier detection. Experiments were performed on different applications which varied in size, operating system (SUNOS, Solaris, Linux and Windows), and environment (simulated and live/real). Results indicate that our system can successfully detect the vulnerability-based anomalies. The generation of false alarms is caused by the irregularities in the data set and the results are sensitive to the parameter selection for the outlier algorithm. We proposed heuristics to automate the parameters to MORPHEUS MinPts (a parameter to LOF) and threshold for raising alarms, thereby making our system parameter-free. Our automatically computed parameter was generally able to detect the attacks producing the least false alarms in the most irregular real data set. After filtering the anomalous points, the clean training data set was used by an online detection system resulting in higher detection rates, implying that MORPHEUS effectively purged the anomalies to create a better training data set.

An attacker might devise a clever technique to evade typical sequence-based anomaly detection systems. Wagner and Soto (Wagner and Soto 2002) presented one such idea wherein they were successful in modeling a malicious sequence by adding null operators to make it consistent with the sequence of system calls. The sequence based techniques dealing with short sub-string patterns can be bypassed by spreading the attack over longer duration (or longer sub-sequences). MORPHEUS uses variable length motifs and also takes the relative

positions of the motifs for anomaly detection, and is better equipped and more robust against such evasions. In essence, our system models sequences at two different levels at the individual motif level and also at the level of spatial relationship between motifs within the audit sequence. The latter level adds to the security of the system and would make it even harder for the attacker to evade the system, since he has to now not only use the normal audit event patterns, but also place those event-sequences/motifs within the respective sequence at proper relative positions.

Chapter 4

Enriched Representations

Most of the traditional host-based anomaly detection systems focus on system call sequences, the assumption being that a malicious activity results in an abnormal (novel) sequence of system calls. Recent research has shown that sequence-based systems can be compromised by conducting mimicry attacks. Such attacks are possible by inserting dummy system calls with invalid arguments such that they form a legitimate sequence of events.

A drawback of sequence-based approaches lies in their non-utilization of other key attributes, namely system call arguments. The efficacy of such systems might be improved upon if a richer set of attributes (return value, error status and other arguments) associated with a system call is used to create the model. In this chapter we present a host-based anomaly detection system that is based upon system call arguments (Tandon and Chan 2003; 2005; 2006). We learn the important attributes using a variant of a rule learning algorithm called LERAD. We also present various argument-based representations and compare their performance with some of the well-known sequence-based techniques.

4.1 Learning Rules for Anomaly Detection

Algorithms for finding association rules, such as Apriori,(Agrawal, Imielinski, and Swami 1993) generate a large number of rules. This incurs a large overhead and may not be appropriate for online detection. We would like to have a minimal set of rules describing the normal training data. LERAD(Mahoney and Chan 2003) is a conditional rule-learning algorithm that forms a small set of rules. LERAD learns rules of the form:

$$A = a \wedge B = b \wedge \ldots \Rightarrow X \in \{x_1, x_2, \ldots\} \tag{4.1}$$

where A, B, and X are attributes and a, b, x_1, x_2 are values for the corresponding attributes. The learned rules represent the patterns present in the normal training data. The set $\{x_1, x_2, \ldots\}$ in the consequent constitutes all unique values of X when the antecedent occurs in the training data.

During the detection phase, records (or tuples) that match the antecedent but not the consequent of a rule are considered anomalous and an anomaly score is associated with every rule violation. The degree of anomaly is based on a probability estimation of novel (zero frequency) events.(Witten and Bell 1991) For each rule, from the training data, the probability, p, of observing a value not in the consequent is estimated by:

$$p = \frac{r}{n} \tag{4.2}$$

where r is the cardinality of the set, $\{x_1, x_2, \ldots\}$, in the consequent and n is the number of records (tuples) that satisfy the rule during training. Since p estimates the probability of a novel event, the larger p is, the less anomalous a novel event is. Hence, during detection, when a novel event is observed, the degree of anomaly (anomaly score) is estimated by:

$$AnomalyScore = \frac{1}{p} = \frac{n}{r} \tag{4.3}$$

A non-stationary model is assumed for LERAD only the last occurrence of an event is assumed important. Since novel events are bursty in conjunction with attacks, a factor t is introduced which is the time interval since the last novel (anomalous) attribute value. If a novel event occurred recently (small value of t), a novel event is more likely to occur at the present moment. Hence, the anomaly score is measured by t/p. Since a record can deviate from the consequent of more than one rule, the total anomaly score of a record is aggregated over all the rules violated by the tuple to combine the effect from violation of multiple rules:

$$TotalAnomalyScore = \sum \frac{t}{p} = \sum \frac{tn}{r} \tag{4.4}$$

The more the violations, more significant the anomaly is, and the higher the anomaly score should be. An alarm is raised if the total anomaly score is above a threshold.

The rule generation phase of LERAD comprises of 4 main steps:

1. Generate initial rule set: Training samples are picked up at random from a random subset S of training examples. Candidate rules (as depicted in Eq. 4.1) are generated from these samples.

2. Coverage test: The rule set is filtered by removing rules that do not cover/describe all the training examples in S. Rules with lower rate of anomalies (lower r/n) are kept.

3. Update rule set beyond S: Extend the rules over the remaining training data by adding values for the attribute in the consequent when the antecedent is true.

4. Validate the rule set: Rules are removed if they are violated by any tuple in the validation set.

Since system call is the key (pivotal) attribute in a host based system, we modified LERAD such that the rules were forced to have a system call as a condition in the antecedent. The only exception we made was the generation of rules with no antecedent.

4.2 System call and argument based representations

We conjecture that even though augmentation of attributes to system call sequences may tend to make the hypotheses space more complex, it will also enable an algorithm to learn the hypotheses more accurately. We now present the different representations for LERAD.

4.2.1 Sequence of system calls: S-LERAD

Using sequence of system calls is a very popular approach for anomaly detection. We used a window of fixed length 6 (as this is claimed to give best results in stide and t-stide (Forrest et al. 1996; Warrender, Forrest, and Pearlmutter 1999)) and fed these sequences of six system call tokens as input tuples to LERAD. This representation is selected to explore whether LERAD would be able to capture the correlations among system calls in a sequence. Also, this experiment would assist us in comparing results by using the same algorithm for system call sequences as well as their arguments. A sample rule learned in a particular run of S-LERAD is:

$$(s_0 = close) \wedge (s_1 = mmap) \wedge (s_5 = open) \Rightarrow s_2 \in \{munmap\}[\tfrac{1}{p} = \tfrac{n}{r} = \tfrac{455}{1}]$$

This rule is analogous to encountering *close* as the first system call (represented as s_0), followed by *mmap* and *munmap*, and *open* as the sixth system call (s_5) in a window of size 6 sliding across the audit trail. Each rule is associated with an n/r value. The number 455 in the numerator refers to the number of training instances that comply with the rule (n in Eq. 4.3). The number 1 in the denominator implies that there exists just one distinct value of the consequent (*munmap* in this case) when all the conditions in the premise hold true (r in Eq. 4.3).

4.2.2 Argument based model: A-LERAD

We propose that argument and other key attribute information is integral to modeling a good host-based anomaly detection system. We extracted arguments, return value and error status of system calls from the audit logs and examined the effects of learning rules based upon system calls along with these attributes. Any value for the other arguments (given the system call) that was never encountered in the training period for a long time would raise an alarm. A typical argument based rule is

$$(s_0 = close) \Rightarrow a_1 \in \{0x2, 0x3, 0x4, 0x5, 0x6\}[\tfrac{1}{p} = \tfrac{n}{r} = \tfrac{500}{5}]$$

where a_1 is the first argument for the first system call ($close$).

We performed experiments on the training data to measure the maximum number of attributes (MAX) for every unique system call. We did not use the test data for these experiments so that we do not get any information about it before our model is built. Since LERAD accepts the same (fixed) number of attributes for every tuple, we had to insert a NULL value for an attribute that was absent in a particular system call. The order of the attributes within the tuple was made system call dependent. As we had modified LERAD to form rules based upon the system calls, there is consistency amongst the attributes for any specific system across all models. By including all attributes we utilized the maximum amount of information possible.

It can be argued that inclusion of NULL values in a rule may result in the formation of many not-so-important rules, thereby making the rule set large and incurring high time and space overhead. But it is also important to note that even though NULL values do not seem to be adding any useful information, they could still help to detect anomalies. Consider the rule

$$(s_0 = munmap) \wedge (a_1 = 0x10) \Rightarrow a_3 \in \{NULL\}[\tfrac{1}{p} = \tfrac{n}{r} = \tfrac{2000}{1}]$$

The rule has only one acceptable value for the third argument when the system call (s_0) is *munmap* and the first argument is 0x10. The rule has a large coverage (= 2000 tuples) in the training data (assuming 2000 tuples is large for a given system call with respect to entire data set). Thus, one would never expect to see any other value for the third argument when the antecedent is true. However, an intruder could craft a mimicry attack by introducing arbitrary argument values (without realizing what a valid value for that argument should be), resulting in a novel value. In such a situation we could say with high confidence that it is a highly unexpected event and hence depicts anomalous behavior. Even though the NULL value for the argument might not determine the nature of the attack, it could be decisive in detecting the anomaly introduced due to a slight carelessness on the part of the intruder.

4.2.3 Merging system call sequence and argument information of the current system call: M-LERAD

The first representation we discussed is based upon sequence of system calls; the second one takes into consideration other relevant attributes, whose efficacy we claim in this paper; so fusing the two to study the effects was an obvious choice. Merging is accomplished by adding more attributes in each tuple before input to LERAD. Each tuple now comprises of the system call, MAX number of attributes for the current system call, and the previous five system calls. The n/r values obtained from the all rules violated are aggregated into an anomaly score, which is then used to generate an alarm based upon the threshold. A sample rule for M-LERAD is of the form

$$(s_0 = close) \wedge (s_5 = open) \Rightarrow a_1 \in \{0x4, 0x5\}[\tfrac{1}{p} = \tfrac{n}{r} = \tfrac{107}{2}]$$

where s_0 and s_5 correspond to the system calls at the extremities of the sliding window and a_1 is the first argument for the current system call (i.e. *close*).

4.2.4 Merging system call sequence and argument information for all system calls in the sequence: M*-LERAD

All the proposed variants, namely S-LERAD, A-LERAD and M-LERAD, consider a sequence of 6 system calls and/or take into the arguments for the current system call. We propose another variant called multiple argument LERAD (M*-LERAD) in addition to using the system call sequence and the arguments for the current system call, the tuples now also comprise the arguments for all system calls within the fixed length sequence of size 6. Each tuple now comprises of the current system call, MAX attributes for the current system call, 5 previous system calls and MAX attributes for each of those system calls. An actual rule formed by M*-LERAD is

$$(s_0 = close) \wedge (a_{5,2} = 4) \wedge (s_4 = mmap) \Rightarrow a_{3,4} \in \{0xe000\}[\tfrac{1}{p} = \tfrac{n}{r} = \tfrac{117}{1}]$$

In the rule above, s_0 and s_4 are the first and fifth system calls respectively in the current window, $a_{5,2}$ is the second argument for the last system call and $a_{3,4}$ is the fourth argument for the fourth system call in the sliding window.

4.3 Experimental Evaluation

The goal is to study if our variant of LERAD with feature spaces comprising system calls and their arguments can detect attack-based anomalies.

4.3.1 Data sets and experimental procedure

We used the following data sets for our experiments:

1. The 1999 DARPA intrusion detection evaluation data set: Developed at the MIT Lincoln Lab(Lippmann et al. 2000), we selected the BSM logs from Solaris host tracing system

calls that contains 33 attacks. The attack taxonomy(Kendell 1999) is briefly explained here.

- Probes or scan attacks are attempts by hackers to collect information prior to an attack. Examples include illegalsniffer, ipsweep, mscan, portscan amongst others.
- DoS (Denial of Service) attacks are the ones in which a host or a network service is disrupted. For example, arppoison, selfping, dosnuke and crashiis are all DoS attacks.
- R2L (Remote to Local) are those attacks wherein an unauthorized user gains access to a system. Examples of R2L attacks are guest, dict, ftpwrite, ppmacro, sshtrojan and framespoof.
- U2R (User to Root) / Data attacks are those in which a local user is able to execute non-privileged commands, which only a super user can execute. Examples are eject, fdformat, ffbconfig, perl, ps and xterm.

The following applications were chosen: *ftpd*, *telnetd*, *sendmail*, *tcsh*, *login*, *ps*, *eject*, *fdformat*, *sh*, *quota* and *ufsdump*, due to their varied sizes (about 1500 to over 1 million system calls). We expected to find a good mix of benign and malicious behavior in these applications. Training was performed on week 3 data and testing on weeks 4 and 5. An attack is considered to be detected if an alarm is raised within 60 seconds of its occurrence (same as the DARPA evaluation).

2. *lpr*, *login* and *ps* applications from the University of New Mexico (UNM): The *lpr* application comprised of 2703 normal traces from hosts running SUNOS 4.1.4. Another 1001 traces result from the execution of the *lprcp* attack script. Traces from the *login* and *ps* applications were obtained from Linux machines running kernel 2.0.35. Trojan programs were used for the attack traces.

3. Microsoft *excel* macros executions (FIT-UTK data): Normal *excel* macro executions are logged in 36 distinct traces. 2 malicious traces modify registry settings and execute some other application. Such a behavior is exhibited by the ILOVEYOU worm which opens the web browser to a specified website and executes a program, modifying registry keys and corrupting user files, resulting in a distributed denial of service (DDoS) attack.

The input tuples for S-LERAD were 6 contiguous system calls; for A-LERAD they were system calls with their return value, error status and arguments; The inputs for M-LERAD were sequences of system calls with arguments of the current system call; whereas in M*-LERAD, they were system call sequences with arguments for all the 6 system calls. For tide, the inputs were all the pairs of system calls within a window of fixed size 6; stide and t-stide comprised all contiguous sequences of length 6. For all the techniques, alarms were merged in decreasing order of the anomaly scores and evaluated at varied false alarm rates.

4.3.2 Comparison of sequence base methods

t-stide is supposed to give best results among the traditional sequence-based techniques(Warrender, Forrest, and Pearlmutter 1999). As the UNM and FIT-UTK data sets do not have complete argument information to evaluate LERAD variants that involve arguments, we compared the performance of S-LERAD and t-stide on these data sets. The ROC curves from this set of experiments are displayed in Fig. 4.1. The X-axis in the plots corresponds to various false alarm rates and the Y-axis represents the percentage of attacks detected. The drawback of anomaly detection lies in the generation of false alarms. Typically, a human expert (administrator) has to deal with all the alarms and the cost of wrongly flagging an alarm can be quite high. Our goal is thus to detect as many attacks while minimizing the generation of false alarms. We have therefore limited our ROCs to a maximum of 1% false alarms (0.01 on the X-axis). Results from Fig. 4.1 show that both the techniques were able to detect all the

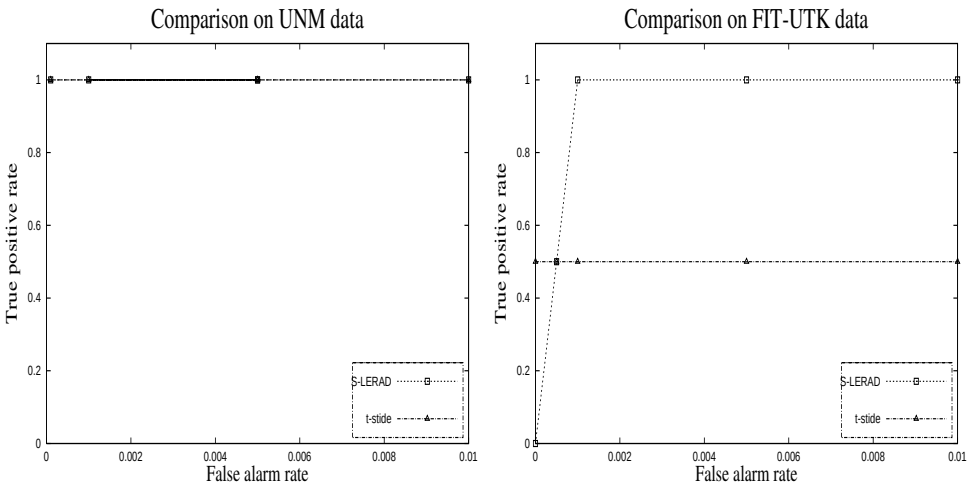

Figure 4.1: ROCs for UNM And FIT-UTK data.

attacks for the UNM data without generating any false alarm. However, for the FIT-UTK data set, there was a difference in the performance. t-stide was initially able to detect more attacks, but S-LERAD was able to detect all the attacks at 0.1% false alarm rate.

We also performed experiments on the DARPA BSM data sets to evaluate all the techniques. Fig. 4.2 illustrates the total attacks detected (Y-axis) at different false alarms rates (X-axis). The original DARPA evaluation (Lippmann et al. 2000) used a threshold of 10 false alarms per day. We used the same evaluation criterion for consistency. The tolerance limit of allowed false alarms may vary from one organization to the other. A government organization dealing with sensitive data would not want to miss any anomaly, keeping the threshold high. On the other hand, a small company may not have the infrastructure to sustain the cost associated with many false alarms per day, thereby preferring a lower threshold but at the risk of missing some intrusions. In order to encompass the viability for a broader range of organizations, we also evaluated the techniques at 0 and 5 false alarms per day. At zero false alarms, tide, stide and t-stide detected the most attacks, suggesting that maximum deviations in temporal sequences are true representations of actual attacks. But as the threshold is relaxed, S-LERAD outperformed all the 3 sequence-based techniques. This can be attributed

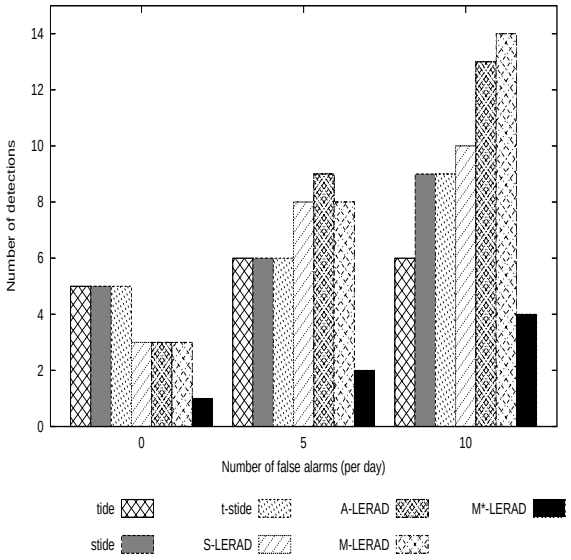

Figure 4.2: Number of detections at different false alarm rates for the BSM data.

to the fact that S-LERAD is able to generalize well and learns the important correlations between the system calls within the given window size.

For completeness, we also present the ROC curves for various techniques in Fig. 4.3. As can be observed from the figure, S-LERAD performs better at false alarm rates less than 0.25%. But t-stide detects more attacks at higher false alarm rates. We reiterate that our goal is to increase the number of attack detections, minimizing the false alarms at the same time. We also list the area under the curve (upto 1% false alarms) in Table 4.1, where higher area implies better performance (Flach 2004). Results in the table show that overall t-stide performs better than S-LERAD in the range of 0 to 1% false alarm rates.

There are two noteworthy points about the evaluation. Firstly, it is interesting to note that the DARPA evaluation had a stricter requirement since 10 false alarms per day (Fig. 4.2) corresponds to even less than 0.1% false alarms on the ROC curve (Fig. 4.3). Another point that deserves mention is that as per the DARPA evaluation, an attack is considered detected only if an alarm is raised within one minute of its occurrence. Thus, an alarm after 60 seconds

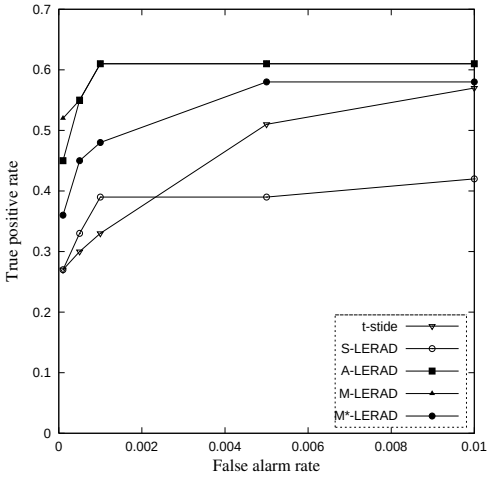

Figure 4.3: ROC for BSM data.

Table 4.1: Area under curve up to 1% false alarm rates.

Technique	*Area* ($\times 10^{-4}$)
t-stide	46.79
S-LERAD	39.12
A-LERAD	60.25
M-LERAD	60.46
M*-LERAD	54.51

would be considered as a false positive even though it may have resulted due to some unusual activity by the intruder.

4.3.3 Comparison of system call sequence and argument based methods

For the DARPA BSM data set, A-LERAD fared better than S-LERAD and the other sequence-based techniques when evaluated in terms of both absolute number (Fig. 4.2) and the percentage (Fig. 4.3) of false alarms, suggesting that argument information is more useful than sequence information. Using arguments could also make a system robust against mimicry attacks which evade sequence-based systems. It can also be seen from Fig. 4.3 that the A-LERAD curve closely follows the curve for M-LERAD. This implies that the sequence information is redundant; it does not add substantial information to what is already gathered from arguments. M*-LERAD performed the worst among all the techniques, as can be seen in Fig. 4.2. The reason for such a performance is that M*-LERAD generated alarms for both sequence and argument based anomalies. An anomalous argument in one system call raised an alarm in six different tuples, leading to a higher false alarm rate. As the alarm threshold was relaxed, the detection rate improved (Fig. 4.3). Table 4.1 also shows that A-LERAD and M-LERAD perform the best under 1% false alarm rates, followed by M*-LERAD and S-LERAD.

Generally, the better performance of LERAD variants can be attributed to its anomaly scoring function. It associates a probabilistic score with every rule. Instead of a binary (present/absent) value (as in the case of stide and t-stide), this probability value is used to compute the degree of anomalousness. It also incorporates a parameter for the time elapsed since a novel value was seen for an attribute. The advantage is twofold:

1. it assists in detecting long term anomalies;

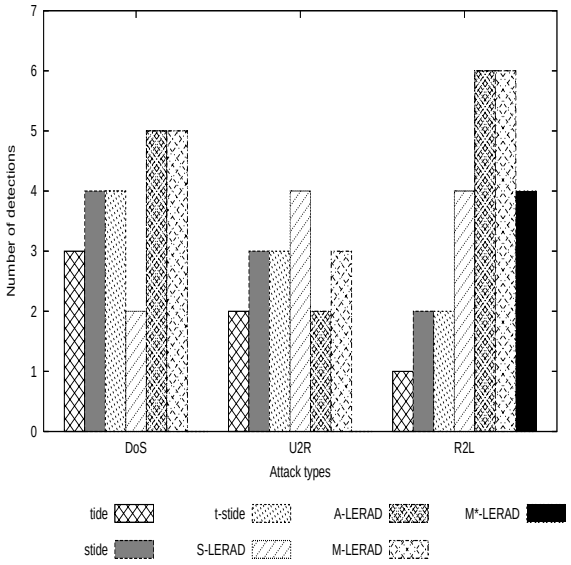

Figure 4.4: Types of attacks detected at 10 false alarms per day for the BSM data.

2. suppresses the generation of multiple alarms for novel attribute values in a sudden burst of data.

Figure 4.4 plots the different types of attacks (mentioned in Sec. 4.3.1) detected at 10 false alarms per day (criterion used in the 1999 DARPA evaluation). Different attack types (DoS, U2R and R2L) are represented along the X-axis and the Y-axis denoted the total attacks detected in each attack category. M-LERAD was able to detect the largest number of attacks 5 DoS, 3 U2R and 6 R2L attacks A-LERAD followed closely with 5 DoS, 2 U2R and 6 R2L attacks. An interesting observation is that the sequence-based techniques generally detected the U2R attacks whereas the R2L and DoS attacks were better detected by the argument-based techniques. Our techniques were able to detect some poorly detected attacks(Lippmann et al. 2000), *warezclient* being one of them. Our models also detected stealthy *ps* attacks.

Table 4.2: Attacks detected by A-LERAD and A^C-LERAD.

False Alarms per day	*Number of attacks detected*	
	A-LERAD	A^C-LERAD
5	10	9
10	13	11
20	17	16

4.3.4 NULL attribute values

As described in Section 4.2.2, we inserted NULL values for attributes not present for a given system call. The expectation is that LERAD would form a rule if the NULL attribute value is characteristic for a system call in an application. Experiments were performed to see if NULL attributes help in detecting anomalies or if they formed meaningless rules. We added a constraint that the NULL values could not be added to the attribute values in the rules and called this variant A^C-LERAD (A-LERAD with constraint). Table 4.2 compares the number of attacks detected by A-LERAD and A^C-LERAD by varying the false alarm rate. Results indicate that A-LERAD was able to detect more attacks than the constrained counterpart, suggesting that rules with NULL valued attributes are beneficial to the detection of anomalies corresponding to attacks.

To analyse why A-LERAD was able to perform better, two metrics were devised to characterize the rule set:

1. Rule Ratio (RR): the ratio of the number of rules formed by A^C-LERAD to the number of rules formed by A-LERAD. This ratio conveys which technique formed a smaller rule set, resulting in a lower run-time overhead.

2. Coverage Ratio (CR): Coverage is the count of the tuples in the training set that are covered by the rule set. Since the coverage is descriptive of the number of tuples it

Table 4.3: Comparison of rule ratio and coverage ratio.

Application	Rule Ratio (RR)	Coverage Ratio (CR)
ftpd	0.35	0.51
telnetd	0.33	0.33
tcsh	0.44	0.48
sendmail	0.52	0.56
quota	0.50	0.53
login	0.36	0.38
sh	0.29	0.27
eject	0.62	0.56
ps	0.50	0.55
ufsdump	0.50	0.49

describes, we would want a high value for coverage. Coverage ratio is the ratio of the coverage of A^C-LERAD to the coverage of A-LERAD.

RR should ideally be low since a small rule set is desired. The coverage ratio should be close to 1, indicating that a comparable number of tuples are covered by A^C-LERAD and A-LERAD. Table 4.3 shows the results for various applications in the BSM log. For example, the size of the rule set for A^C-LERAD was 35% of the size of the rule set of A-LERAD for *ftpd*. This implies fewer rules for A^C-LERAD and hence a lower run-time overhead during the detection phase. However, the coverage ratio was 51%. The anomaly score is higher for rules with higher coverage (Eq. 4.3). This assists A-LERAD to assign higher scores to tuples that violate rules and hence detect more attacks than A^C-LERAD. Overall results indicate that even though discarding NULL attribute values results in a drastic reduction in the rule set, they also cause smaller coverage amongst the training tuples. Another interesting observation

from Table 4.3 was that a decrease in the size of the rule set generally yields a similar decrease in the coverage.

4.3.5 Analysis of anomalies - attack detections and false alarms

An anomaly is a deviation from normalcy and, by definition, does not necessarily identify the nature of an attack. Anomaly detection serves as an early warning system; humans need to investigate if an anomaly actually corresponds to a malicious activity. Table 4.4 list the anomalous attributes that resulted in the detection of the attack. It is interesting to note that the anomalies that led to the attacks detected by argument-based variants of LERAD, in many cases, do not represent the true nature of the attacks. Instead, it may be representative of behavioral patterns resulting from the execution of some other program after the intruder successfully gained access to the host. For example, an instance of *guest* attack is detected by A-LERAD not by observing attempts by the hacker trying to gain access, but by encountering novel arguments to the *ioctl* system call which was executed by the hacker trying to perform a control function on a particular device. A stealthy *ps* attack was detected by our system when the intruder tried to change owner using a novel group id.

Even if the anomaly is related to the attack itself, it may reflect very little information about the attack. Our system is able to learn only a partial signature of the attack. *guessftp* is detected by a bad password for an illegitimate user trying to gain access. However, the attacker could have made interspersed attempts to evade the system. Attacks were also detected by capturing errors committed by the intruder, possibly to evade the IDS. *ftpwrite* is a vulnerability that exploits a configuration error wherein a remote ftp user is able to successfully create and add files and gain access to the system. An instance of this attack is detected by monitoring the subsequent actions of the intruder, wherein he attempts to set the audit state using an invalid preselection mask. This anomaly would go unnoticed in a system

Table 4.4: Anomalous arguments for attacks detected by A-LERAD.

Attack	Attribute anomalies
ftpwrite	set audit without ownership, invalid file descriptor, invalid mmap return value, invalid device-dependent request code
guesstelnet	novel ioctl return value
ps	change group ownership with anomalous group id, invalid device, inappropriate ioctl for device
guessftp	fail file open on remote system
warez	invalid device-dependent request code, change ownership of file without privileges
guest	inappropriate ioctl for device
warezclient	not owner to set audit state
syslogd	novel ioctl argument

monitoring only system calls.

We re-emphasize that our goal is to detect anomalies, the underlying assumption being that anomalies generally correspond to attacks. Since not all anomalous events are malicious, we expect false alarms to be generated. Table 4.5 lists the attributes responsible for the generation of alarms and whether these resulted in actual detections or not. It is observed that some anomalies were part of benign application behavior. At other instances, the anomalous value for the same attribute was responsible for detecting actual malicious execution of processes. As an example, many attacks were detected by observing novel arguments for the *ioctl* system call, but many false alarms were also generated by this attribute. Even though not all novel values correspond to any illegitimate activity, argument-based anomalies were instrumental in detecting the attacks.

Table 4.5: Top anomalous attributes for A-LERAD.

Attribute causing false alarm	Whether some attack was detected by the same attribute
ioctl argument	Yes
ioctl return value	Yes
setegid mask	Yes
open return value	No
open error status	No
fcntl error status	No
setpgrp return value	No

4.3.6 Storage and computational overheads

Compared to sequence-based methods, our techniques extract and utilize more information (system call arguments and other attributes), making it imperative to study the feasibility of our techniques for online usage. For t-stide, all contiguous system call sequences of length 6 are stored during training. For A-LERAD, system call sequences and other attributes are stored. In both the cases, space complexity is of the order of O(n), where n is the total number of system calls, though the A-LERAD requirement is more by a constant factor k since it stores additional argument information.

During detection, A-LERAD uses only a small set of rules (in the range 10-25 for the applications used in our experiments). t-stide, on the other hand, still requires the entire database of fixed length sequences during testing, which incur larger space overhead during detection. We conducted experiments on *tcsh* application in the BSM data set, which comprises of over 2 million system calls in training and has over 7 million system calls in test data. The rules formed by A-LERAD require around 1 KB space, apart from a mapping table

Table 4.6: Computational overhead for one week training and two weeks testing.

Application	*Total training time (seconds)*		*Testing time per sample (milliseconds)*	
	tstide	ALERAD	tstide	ALERAD
ftpd	0.19	0.90	0.04	0.16
telnetd	0.96	7.12	0.02	0.17
tcsh	6.32	29.56	0.13	0.17
sendmail	2.73	14.79	0.05	0.17
quota	0.20	3.04	0.01	0.14
login	2.41	15.12	0.04	0.17
sh	0.21	2.98	0.03	0.16
ufsdump	6.76	30.04	0.01	0.14

to map strings and integers. The memory requirements for storing a system call sequence database for t-stide were over 5 KB plus a mapping table between strings and integers. The results suggest that A-LERAD has better memory requirements during the detection phase. We reiterate that the training can be done offline. Once the rules are generated, A-LERAD can be used to do online testing with lower memory requirements.

The time overhead incurred by A-LERAD and t-stide in our experiments is given in Table 4.6. The CPU times have been obtained on a Sun Ultra 5 workstation with 256 MB RAM and 400 MHz processor speed. It can be inferred from the results that A-LERAD is slower than t-stide. During training, t-stide is a much simpler algorithm and processes less data than A-LERAD for building a model and hence t-stide has a much shorter training time. During detection, t-stide just needs to check if a sequence is present in the database, which can be efficiently implemented with a hash table. On the other hand, A-LERAD needs to

check if a record matches any of the learned rules. Also, A-LERAD has to process additional argument information. Run-time performance of A-LERAD can be improved with more efficient rule matching algorithm. Also, t-stide will incur significantly larger time overhead when the stored sequences exceed the memory capacity and disk accesses become unavoidable A-LERAD does not encounter this problem as easily as t-stide since it will still use a small set of rules. Moreover, the run-time overhead of A-LERAD is below a couple of hundred microseconds per sample, which is reasonable for practical purposes.

4.4 Summary

Sequence based anomaly detection systems are known to suffer from high false alarm rates. In this paper, we portrayed the efficacy of incorporating system call argument information and used a rule-learning algorithm to model a host-based anomaly detection system. We performed experiments on data sets from varied operating systems and applications. We empirically demonstrated that our sequence-based variant (S-LERAD) detects more attacks at lower false alarm rates. It was thus able to generalize better than the prevalent sequence based techniques, which rely on pure memorization. Results also show that our argument-based model, A-LERAD, detected more attacks than all the sequence-based techniques, stressing on the importance of system call arguments for modeling host based systems.

Merging argument and sequence information creates a richer model for anomaly detection, as illustrated by the empirical results of M-LERAD. M*-LERAD detected lesser number of attacks at lower false alarm rates since every anomalous attribute results in alarms being raised in 6 successive tuples, leading to either multiple detections of the same attack (counted as a single detection) or multiple false alarms (all separate entities). Results also indicated that sequence-based methods help detect U2R attacks whereas R2L and DoS attacks were better detected by argument-based models. Our argument-based techniques detected different

types of anomalies. Some anomalies did not represent the true nature of the attack. Some attacks were detected by subsequent anomalous user behavior, like trying to change group ownership. Some other anomalies were detected by learning only a portion of the attack, while some were detected by capturing intruder errors.

Though our techniques incur higher time overhead due to the complexity of our techniques (since more information is processed) as compared to t-stide, they build more succinct models that incur much less space overhead - our techniques aim to generalize from the training data, rather than pure memorization. Moreover, under 200 microseconds per sample (during testing phase) is reasonable for online systems, even though it is significantly longer than t-stide.

Chapter 5

Increasing coverage to improve detection of anomalies

Rules for normal behavior can be hard coded by a human expert, a tedious task that incurs significant effort and cost; or automatically learned from normal data using machine learning. One such technique, called LERAD (LEarning Rules for Anomaly Detection)(Mahoney and Chan 2003), efficiently learns a succinct set of comprehensible rules and detects attacks unknown to the algorithm. To reduce false alarms, these rules are validated on normal *held-out* data and all violated rules are discarded. However, these rules were selected initially to cover a relatively large number of training examples and their elimination could possibly lead to missed detections. Outright rejection of such rules (called *Pruning* in this paper) reduces the coverage. This chapter presents two methods to improve rule coverage (Tandon and Chan 2007) – we can either lessen the belief in the violated rule instead of eliminating it (*Weighting*), or backtrack to find rules that cover the training examples that should be covered (*Replacement*). In *Weighting*, rules are associated with weights estimating rule belief.

A conformed rule increases our belief in it and hence its weight is increased. On the other hand, weight is decreased upon rule violation symbolizing decrease in trust. In *Replacement*, coverage is increased by including candidate rules that cover attribute values which have lost coverage due to pruned rules. Additionally, new rules are learned from attribute values that are not covered. We conjecture that increasing coverage over training data would increase the number of attack detections. Thus, we also present a third technique, called *Hybrid*, that chooses between *Weighting* and *Replacement*, the one that has higher coverage on training data.

In Section 2.2.1, we describe two aspects of rule quality: predictiveness and belief. For rule learning algorithms, many studies demonstrate the efficacy of using weights (*predictiveness* and/or *belief*) over not using weights as well as pruning over not pruning. However, we are not aware of studies in comparing using weights and pruning, particularly in anomaly detection. In this chapter, we study how rule weighting compares to pruning in a rule learning algorithm for anomaly detection. Also, for the LERAD algorithm, two instances are randomly chosen and rules are generated using matching attribute values, making it a data driven approach. Hypotheses search is from general to specific, and rules are updated by adding values over entire training data. LERAD differs from AQ15 in that it allows different attributes in the rule consequent, whereas AQ15 learns classes based on a single attribute. Also, AQ15 generates all rules that cover a data instance. In contrast, LERAD randomly choses matching attributes, thus generating a subset of the rules and making the algorithm more efficient. In this chapter, we also present *Replacement* variant for LERAD that revisits candidate rules to replace rules discarded during validation. Since rules generated previously are checked against the validation set for false alarms, this step is hypotheses driven. In addition, *Replacement* takes into account attribute values not covered and learns rules based on those target values, making this step a data driven strategy and the technique a mixture of data and hypotheses

driven methods.

5.1 Rule Pruning in LERAD

LEarning Rules for Anomaly Detection (LERAD) (Mahoney and Chan 2003) is an efficient randomized algorithm that forms conditional rules of the form:

$$a_1 = v_{11} \bigwedge a_2 = v_{23} \bigwedge ... => a_c \in \{v_{c1}, v_{c2}, ...\} \quad [p] \tag{5.1}$$

where a_i is the i^{th} attribute and v_{ij} is the j^{th} value for a_i. LERAD adopts a probabilistic framework and estimates $P(C|A)$, where A is the antecedent and C is the consequent of the rule $A \Rightarrow C$. During training, a set of rules R that "minimally" describes the training data are generated and their $p = P(\neg C|A)$ is estimated, where C, though expected, is not observed when A is observed. An estimate for novel events from data compression (Witten and Bell 1991) is used:

$$p = P(NovelEvent) = \frac{r}{n}. \tag{5.2}$$

where n is the total number of observed events and r is the number of unique observed events. A sample network anomaly rule for LERAD is:

$$SrcIp = 128.1.2.3 \wedge DestIp = 128.4.5.6 \Rightarrow DestPort \in \{21, 25, 80\}[p = \frac{r}{n} = \frac{3}{100}] \tag{5.3}$$

A data instance is a feature vector comprised of all feature attribute values, and multiple data instances form a data set. For example, a network data set may be composed of data instances represented by the feature vector $< SrcPort, DestPort, SrcIp, DestIp >$. The rule of Eq. 5.3, which claims three distinct destination ports (21-FTP, 25-SMTP, 80-HTTP) given the source and destination IP addresses, is satisfied by 100 data instances. One or more such rule(s) forms the rule set. A synthetic network data set and rule set is presented in Table 5.1, with the semantics of data instance and rule as explained above.

Table 5.1: Example data set and rule set.

d_i	$SrcPort$	$DestPort$	$SrcIp$	$DestIp$
d_1	80	80	128.1.2.3	128.4.5.6
d_2	80	80	128.1.2.3	128.4.5.6
d_3	80	25	128.3.2.1	128.4.5.6

r_1: $* \Rightarrow DestPort \in \{25, 80\}[p = 2/3]$

r_2: $SrcIp = 128.1.2.3 \Rightarrow DestIp \in \{128.4.5.6\}[p = 1/2]$

r_3: $DestIp = 128.4.5.6 \Rightarrow SrcPort \in \{80\}[p = 1/3]$

Definition 1. Let i be a data instance and j be an attribute. $cover_{ij}$ is 1 if there exists a rule r (Eq. 5.1) in the rule set such that instance i satisfies the condition(s) in the antecedent of rule r and the value of attribute j in instance i is a member of the set of values of the same attribute j in the consequent of rule r. That is, $cover_{ij}$ indicates if the value of attribute j in instance i is covered by a rule in the rule set. More formally:

$$cover_{ij} = \begin{cases} 1, & \text{if } j^{th} \text{ attribute value is in consequent} \\ & \text{of a rule satisfied by } i^{th} \text{ instance} \\ 0, & \text{otherwise} \end{cases} \tag{5.4}$$

In Table 5.1, rule r_1 is applicable for all three data instances, and all $DestPort$ values are contained in the consequent. Hence, $cover_{12} = cover_{22} = cover_{31}$=1. The attribute values not covered by the rule set are $SrcIp$ of data instances $d_1 - d_3$, each of which has a cover value of 0. Similarly, $cover_{34}$=0, since $DestIp$ of data instance d_3 is not covered by any rule in the rule set.

Definition 2. For the entire data set, *coverage* is defined as the fraction of attribute values covered by the rule set. Let N be the total number of instances, and M the number of

Input: sample set (D_s), training set (D_t), and validation set (D_v)

Output: LERAD rule set R

1. generate candidate rules from D_s and evaluate them
2. perform coverage test - select a "minimal" set of candidate rules that covers D_s:
   (a) sort candidate rules in increasing order of probability of being violated
   (b) discard rules that do not cover any attribute values in D_s
3. train the selected candidate rules on D_t
4. eliminate the rules that cause false alarms on D_v

Figure 5.1: Main steps of LERAD algorithm

distinct attributes, then *coverage* is formally defined as:

$$coverage = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} cover_{ij} \tag{5.5}$$

The example in Table 5.1 has three data instances ($d_1 - d_3$) and four attributes. Eight attribute values are covered by the 3 rules in the rule set, resulting in a *coverage* value of 0.67.

5.1.1 Training Candidate Rules and Coverage Test

For describing the LERAD algorithm, we use the following notation. Let D be the entire data set, and D_T be the training set with normal behavior and D_E be the evaluation (test) data set with normal behavior as well as attacks such that $D_T \cup D_E = D$ and $D_T \cap D_E = \emptyset$. Training data is further partitioned into subsets D_t (training data set) and D_v (validation *held-out* data set) respectively such that $D_t \cup D_v = D_T$, $D_t \cap D_v = \emptyset$, and $|D_t| > |D_v|$. Also, let R be the rule set learned after training.

The LERAD algorithm consists of four main steps as illustrated in Fig. 5.1. Step 1 intends

Table 5.2: Example training data subset $D_s = \{d_i\}$ for $i = 1..3$ and rules r_k ($k = 1..3$) generated from D_s. Consequent attribute values in data instances are marked by (r_k) in coverage test.

d_i	$SrcPort$	$DestPort$	$SrcIp$	$DestIp$
d_1	80	80 (r_2)	128.1.2.3	128.4.5.6
d_2	80	80 (r_2)	128.1.2.3	128.4.5.6
d_3	80	25 (r_1)	128.3.2.1	128.4.5.6

r_1: $* \Rightarrow DestPort \in \{25, 80\}[p = 2/3]$

r_2: $SrcIp = 128.1.2.3 \Rightarrow DestPort \in \{80\}[p = 1/2]$

r_3: $SrcIp = 128.1.2.3 \bigwedge DestIp = 128.4.5.6 \Rightarrow DestPort \in \{80\}[p = 1/2]$

to generate and evaluate candidate rules from a small data sample D_s (such that $|D_s| \ll |D_t|$), which allows efficient training. Step 2 selects a small set of predictive rules that sufficiently describe D_s. This allows learned models to be small. This step, called *coverage test*, is based on two heuristics. First, rules with lower $p = P(\neg C|A)$ are preferred. Second, a rule can cover multiple instances in D_s, but an instance does not need to be covered by more than one rule. Hence, rules are sorted based on p and evaluated in ascending order (Step 2a). For each rule, instances covered by the rule are marked. If a rule cannot mark any remaining unmarked instances, it is removed. That is, rules with lower p are retained and rules that do not contribute to covering instances not covered by previous rules with lower p values are discarded (Step 2b). More specifically, since rules can have different attributes in the consequent, the attribute of an instance, not the entire instance, is marked. Hence, a rule is removed only if it cannot mark any unmarked attribute of any instance.

Table 5.2 contains example data and candidate rules to help illustrate the algorithm. Rules r_2 and r_3 in the table have a lower p (1/2) than r_1, so r_2 or r_3 is evaluated first. Rule r_2

is arbitrarily picked before r_3, and marks *DestPort* of d_1 and d_2. Then r_3 cannot mark any attribute and is removed. Finally, r_1 marks *DestPort* of d_3. At this point, rules r_2 and r_1 are selected. This procedure guarantees at least one attribute of all instances in D_s are marked by a subset of candidate rules. Also, rules that are subsumed by more general rules are automatically removed due to a higher p.

The selected rules are then updated using the much larger set D_t (Step 3 in Fig. 5.1) by updating the consequent and p. If the antecedent of a rule matches an instance in D_t and the consequent of the rule does not contain the corresponding value, the value is added to the consequent. p is updated by updating the number of instances that match the antecedent in D_t (n) and values in the consequent (r). The validation set D_v is used in Step 4, and is described next in context of *Pruning*.

5.1.2 Validating Rules

To reduce overfitting the training data, machine learning algorithms use a separate *held-out* data to validate the trained model. LERAD uses validation set D_v for the rules learned from D_t. For each rule $r_k \in R$ and instance $d \in D_v$, one of three cases apply:

1. The rule is *conformed* when all conditions in the antecedant as well as the consequent are satisfied by the instance. For example, instance [SrcIp = 128.1.2.3, DestIp = 128.4.5.6, DestPort = 80] conforms to the rule in Eq. 5.3.

2. The rule is *violated* if the antecedant holds true but the consequent does not. The rule in Eq. 5.3 is violated by the instance [SrcIp = 128.1.2.3, DestIp = 128.4.5.6, DestPort = 23].

3. The rule is not applicable for the instance if any condition in the antecedant is not satisfied, an example instance being [SrcIp = 128.1.5.7, DestIp = 128.4.5.6, DestPort =

21].

A conformed rule (case 1) is not updated but the associated p value is modified. Given instance [SrcIp = 128.1.2.3, DestIp = 128.4.5.6, DestPort = 80], the rule in Eq. 5.3 has new p value of 3/101 upon conformance. For rule violation (case 2), the rule is eliminated from the rule set (Step 4 in Fig. 5.1) since D_v is normal and each anomaly is a false alarm. Inapplicable rules (case 3) are left unchanged along with their p values. This version of LERAD is referred to as *Pruning* for the remainder of the paper.

5.1.3 Scoring Anomalies

During the monitoring stage, LERAD uses the learned rules to assign an anomaly score to each data instance. During detection, given a data instance d, an anomaly score is generated if d violates any of the rules. Let $R' \subset R$ be the set of rules that d violates. The anomaly score is calculated as:

$$AnomalyScore(d) = \sum_{r_k \in R'} \frac{1}{p_k}, \tag{5.6}$$

where r_k is a rule in R' and p_k is the p value of rule r_k representing its *predictiveness*. The reciprocal of p_k reflects a surprise factor that is large when anomaly has a low likelihood (small p_k). Intuitively, we are less surprised if we have observed a novel value in a more recent past. Let t_k be the duration since the last novel value was observed in the consequent of rule r_k. A non-stationary model is proposed and each violated rule r_k assigns a score:

$$Score_k = \frac{t_k}{p_k}. \tag{5.7}$$

Total anomaly score is accumulated over all violated rules:

$$AnomalyScore(d) = \sum_{r_k \in R'} \frac{t_k}{p_k}. \tag{5.8}$$

The t_k factor also accommodates the "bursty" nature of network traffic (Paxson and Floyd 1995), so that multiple successive anomalies generate a single high scoring alarm.

We claim that *Pruning* reduces rule coverage, resulting in lower accuracy. We propose two solutions to increase coverage: retain pruned rules with lower rule belief using *Weighting*, or revisit candidate rules to replace pruned rules, called *Replacement*. These solutions are discussed in the next two sections.

5.2 Rule Weighting

LERAD performs a coverage test to minimize the number of rules (Step 2 in Fig. 5.1). Thus each selected rule covers a relatively large number of examples in the training set D_t. But removing a rule that causes false alarms also removes coverage on a relative large number of training examples, which can lead to missed detections. Thus, there is a trade off between decreasing false alarms and increasing missed detections.

We propose associating a weight with each rule in the rule set to symbolize rule belief. Violated rules are penalized by reducing their weights, whereas conformed rules are rewarded with increase in their respective weights. A sample rule using our method is of the form:

$$SrcIp = 128.1.2.3 \bigwedge DestIp = 128.4.5.6 \Rightarrow DestPort \in \{21, 25, 80\}[p = 3/100, w = 1.0] \tag{5.9}$$

The semantics of this rule is similar to the rule in Eq. 5.3, but a new w value is introduced for the rule weight to represent belief in the rule. p and w are distinct and independent entities — p is the probability of not seeing a value in the consequent when the conditions in the antecedant hold true (i.e. probability of the rule being violated) and corresponds to *predictiveness* from Section 2.2.1; weight w, on the other hand, approximates the *belief* of the entire rule.

Input: sample set (D_s), training set (D_t), and validation set (D_v)

Output: LERAD rule set R

1. generate candidate rules from D_s and evaluate them
2. coverage test - select a "minimal" set of candidate rules that covers D_s:
   (a) sort candidate rules in increasing order of probability of being violated
   (b) discard rules that do not cover any attribute values in D_s
3. train the selected candidate rules on D_t and assign weight of unity
4. validate rules on D_v
   (a) increase weight on rule conformance (increase rule belief)
   (b) decrease weight on rule violation (reduce rule belief)

Figure 5.2: Rule Weighting in LERAD

5.2.1 Validating Rules

The training is similar to *Pruning*, as seen in Fig. 5.2. The main difference lies in the validation step (Step 4 in Fig. 5.2). Instead of making a binary decision of retaining or eliminating a rule, we keep a rule but update its associated weight. The rule consequent and p value may also be updated. For conformed rules (case 1 in Sec. 5.1.2), p is updated similar to *Pruning* but has an additional w value. Rule violation (case 2 in Sec. 5.1.2) results in updating the rule as well as probability p. For the instance [SrcIp = 128.1.2.3, DestIp = 128.4.5.6, DestPort = 23], the rule in Eq. 5.9 is modified as:

$$SrcIp = 128.1.2.3 \bigwedge DestIp = 128.4.5.6 \Rightarrow DestPort \in \{21, 23, 25, 80\}[p = 4/101, w] \tag{5.10}$$

How weights are updated for conformed and violated rules (case 1 and 2 respectively from Sec. 5.1.2) is discussed next. No action is taken for inapplicable rules (case 3).

5.2.2 Weighting Strategies

We propose associating a weight to each rule $r \in R$, where weights symbolize rule *belief*. Violated rules are penalized by reducing their weights, whereas conformed rules are rewarded with increase in their respective weights. Next, we present three weighting schemes used in our experiments.

Winnow-Specialist-based Weight Update

Winnow is an incremental weight updating algorithm for voting experts (Littlestone 1988), which correspond to rules in our case. Our first weighting strategy is similar to the Winnow specialist variant of (Blum 1997). Initially all rule weights are assigned a value 1, signifying equality of *belief* across the rule set. For any data instance $d \in D_v$, a rule $r \in R$ must either hold good or be inapplicable (in which case it abstains from voting). Any rule violation in D_v corresponds to a false alarm (since D_v comprises of non-attack data) and reduces trust in the culprit rule. If a rule formed during training is not useful, it is likely to be violated many times. Such rules are penalized by multiplicative decay of their weight. On the other hand, if a rule is conformed by a data instance $d \in D_v$ when other rule(s) were violated, it stresses upon validity of the rule and increases trust. Since the rule formed during training is expected to hold true in validation as well, we increase its weight by a small fraction. The intent is to levy a heavy penalty by decreasing the weight by a factor α when the rule is violated, but increase the weights by factor β for a conformed rule.

The strategy to update weights is formally defined by the following weight update function:

$$w_k = \begin{cases} w_k \times \alpha, & \text{if } r_k \in R \text{ is violated} \\ w_k(1+\beta), & \text{if } r_k \in R \text{ is conformed but} \\ & r_j \in R \text{ is violated } (j \neq k) \end{cases} \tag{5.11}$$

where α, $\beta \in \Re$, $0 \leq \alpha < 1$ and $0 \leq \beta \leq 1$. Assuming $\alpha = 0.5$ and inital weight 1, the weight is equal to 0.5 the first time the rule is violated. It is reduced to 0.25 upon second violation and so on. On the other hand, weight is updated as 1.5, 2.25, 3.375 ($\beta = 0.5$) for the first three conformances respectively, when there was atleast one rule violation for the same data instance. Theoretical bounds for the parameters have been presented in (Littlestone 1988; Blum 1997). It can be noted that *Pruning* is a special case of this weighting strategy, with $\alpha = \beta = 0$.

Equal Reward Apportioning

This is a variant of the Winnow-Specialist-based approach explained above. One can observe from Eq. 5.11 that the weights for correct rules are incremented by a constant factor β. This results in varied weight increments across conforming rules. For example, given $\alpha = \beta = 0.5$, current weights 1.0 and 0.5 of two conforming rules r_1 and r_2 are updated as 1.5 and 0.75 respectively. The Winnow-Specialist-based scheme thus favors rules with already higher weights by increasing their weights even more, resulting in potential imbalance. Moreover, the amount of weight increase is independent of whether a high or low belief rule was violated.

The *Equal Reward Apportioning* scheme adopts an impartial approach towards all conforming rules, irrespective of their current weights. This weighting scheme aggregates the total weight reduction due to violation of rules, and rewards the conforming rules by equally distributing the consolidated weight mass amongst them. For each instance in $d \in D_v$, the total penalty TP is computed as:

$$TP = \sum_{r_k \in R_v} (1 - \alpha) w_k, \tag{5.12}$$

where $R_v \subseteq R$ is the set of rules violated by d and $\alpha \in \Re$ ($0 \leq \alpha < 1$). Let $R_c \subseteq R$ be the set

of conformed rules. The weights are updated as follows:

$$w_k = \begin{cases} w_k \times \alpha, & \text{if } r_k \in R \text{ is violated} \\ \\ w_k + \frac{TP}{|R_c|}, & \text{if } r_k \in R \text{ is conformed} \end{cases} \tag{5.13}$$

The amount of weight increase for conforming rules is thus dependent on the amount of weight decreased for violated rules. Following the example above, if the violated rule r_3 has weight 0.6, weights for conformed rules r_1 and r_2 are incremented by the same amount (0.15), resulting in weights 1.15 and 0.65 respectively. On the other hand, if a higher trust rule is violated, say rule r_4 with weight 1.0, it provides greater boost to the conforming rules r_1 and r_2 by incrementing their weights by 0.25 each.

Weight of Evidence

Weight of evidence is defined as the measure of evidence provided by an observation in favor of a target attribute value as opposed to other values for the same target attribute. This measure is based on information theory and has been applied in classification tasks based on event associations (Wang and Wong 2003). Mathematically, it is the difference in the mutual information when the target attribute Y takes a certain value y and when it doesn't, given some observed value x for the attribute X:

$$W(Y = y/Y \neq y \mid X = x) = I(Y = y; X = x) - I(Y \neq y; X = x), \tag{5.14}$$

where $I(a; b)$ is the mutual information of a and b and is computed as:

$$I(a; b) = P(a, b) log \frac{P(a, b)}{P(a)P(b)}. \tag{5.15}$$

We cannot apply Eq. 5.14 directly to our problem since we are not trying to predict a single target value. Rather, we want to measure the gain provided by an observation for the target value to be from a finite set of values. The weight of evidence for the k^{th} rule is reformulated

as:

$$w_k(Y \in \{y_1, y_2, \ldots, y_n\} / Y \notin \{y_1, y_2, \ldots, y_n\} \mid \underline{\mathrm{X}}) \tag{5.16}$$

$$= I(Y \in \{y_1, y_2, \ldots, y_n\}; \underline{\mathrm{X}}) - I(Y \notin \{y_1, y_2, \ldots, y_n\}; \underline{\mathrm{X}})$$

where $\{y_1, y_2, \ldots, y_n\}$ is the set of values for the target attribute Y of the rule r_k; and $\underline{X}$ corresponds to the conditions in the antecedent.

We used this scheme to associate weights with the rules in the rule set. The weight is computed for each rule $r \in R$ based on the evidence in D_v. Contrary to the previous two incremental weighting techniques, this involves batch weighting where evidence is consolidated from D_v as a whole. Moreover, weight of evidence can be positive, negative or zero. A positive value reflects high trust in the rule whereas a negative or zero value implies otherwise. Only rules with positive weights are kept and the remaining may be eliminated. One can also scale the values by a linear shift of the axis such that all weights are positive. Now the high belief (positive weight of evidence) rules have high positive weights, whereas the low (negative/zero weight of evidence) trust rules have low positive weights. Due to its simplicity and intuitiveness, we used the former approach for our experiments.

5.2.3 Scoring Anomalies

Each rule assigns an anomaly score to a test instance $d \in D_T$, a higher score implying more critical aberration. Different algorithms adopt different scoring schemes, the simplest being incrementing the anomaly score by unity. The anomaly score for the test instance is aggregated over all the rules in the rule set. A rule may abstain from assigning a score if it is not applicable (i.e. the antecedent does not hold true). We incorporate the weight representing rule trust to compute the anomaly score:

$$AnomalyScore(d) = \sum_{r_k \in R'} (w_k \times Score_k), \tag{5.17}$$

where $Score_k$ is due to violation of rule r_k and w_k is the weight of the violated rule. Thus, each rule assigns an anomaly score proportional to its weight, and all the scores are aggregated to compute the total anomaly score. Modified anomaly score for LERAD follows from Eqs. 5.7, 5.17:

$$AnomalyScore(d) = \sum_{r_k \in R'} \frac{w_k t_k}{p_k}. \tag{5.18}$$

Thus, anomaly score in *Weighting* incorporates both the *predictiveness* and *belief* aspects of rule quality.

5.3 Rule Replacement

Weighting introduces the additional aspect of rule belief and increases rule coverage over training data by retaining previously pruned rules. Alternatively, one can revisit rules rejected during coverage test (Step 2 in Fig. 5.1). In this section, we present the *Replacement* technique that substitutes pruned rules with new rules to increase coverage over training data. *Replacement* increases coverage in two ways:

- by considering rules with lower predictiveness that were rejected during coverage test, and
- by generating new candidate rules from instances in the training data set with attribute values not covered.

Since rules generated previously are checked against the validation set for false alarms, the former approach is hypotheses driven. The latter approach takes into account attribute values not covered and learns rules based on those target values, making it a data driven strategy and the replacement technique a mixture of data and hypotheses driven methods.

The main steps of *Replacement* are presented in Fig. 5.3. Steps 1-4 are same as *Pruning* with the exception of Step 2b, where rules that do not increase coverage are retained in a

Input: sample set (D_s), training set (D_t), and validation set (D_v)

Output: LERAD rule set R

1. generate candidate rules from D_s and evaluate them
2. select a "minimal" set of candidate rules that covers D_s (i.e. coverage test):
   (a) sort candidate rules in increasing order of probability of being violated
   (b) R_{pool} = rules that do not cover any consequent attribute values
   (c) select remaining rules as candidate rules
3. train the selected candidate rules on D_t
4. eliminate the rules that cause false alarms on D_v
5. while (rule violations in $D_v \wedge R_{pool} \neq \emptyset \wedge$ candidate rules $\in R_{pool} \wedge$ coverage < 1.0)
   (a) if (candidate rule increases coverage in D_t)
   add candidate rule to R and remove it from R_{pool}
   (b) validate the rule set on D_v and eliminate rules violated
6. generate new candidate rules from D_u, which are instances in D_s with attribute values not covered
7. select a "minimal" set of candidate rules that covers D_u
8. eliminate violating rules on D_v and add remaining rules to R

Figure 5.3: Rule Replacement in LERAD

Table 5.3: Example training data subset $D_u = d_i$ $i = 4, 5$, representing attribute values not covered for $DestPort$.

d_i	$SrcPort$	$DestPort$	$SrcIp$	$DestIp$
d_4	25	80	128.1.2.3	128.7.8.9
d_5	25	80	128.1.2.3	128.0.3.5

rule pool R_{pool}, instead of eliminating them. The validation phase (Step 4) may prune rules with high predictiveness, resulting in significant loss of coverage. Rules from R_{pool} can then be re-evaluated to increase coverage over training data (Step 5). Rules that increase the coverage are added to the rule set in Step 5a. It can be noted that such rules will have lower predictivess than the rules in Rule Pruning. Step 5b validating the new rules against D_v. Rules causing false alarms are eliminated, and remaining rules from R_{pool} are considered in the subsequent iteration. This loop terminates when no rules remain the pool, or all attribute values are covered, or the entire rule set conforms to the validation data set. At the end of each iteration, coverage increases or remains the same. To maximize coverage, all candidate rules are considered in each iteration, except those already in the rule set or pruned in validation in previous iterations. Additionally, new candidate rules are learned from instances with attribute values that are not covered (Step 6). Only data instances with not yet covered values are used in this step to generate new candidate rules, which are constrained to have those values in the rule consequent. Generalizations and rules with higher predictiveness are preferred to keep the rule set small (Step 7). Step 8 involves pruning rules that cause false alarms on the validation data set. Anomalies are scored as shown in Eq. 5.8 for the *Pruning* strategy.

Consider the synthetic data and rules from Table 5.2. Coverage test removes r_3 ($SrcIp =$ 128.1.2.3 $\bigwedge DestIp$ = 128.4.5.6 $\Rightarrow$ $DestPort \in \{80\}[p = 1/2]$), which is now added to the

rule pool R_{pool}. Assume rule r_2 ($SrcIp = 128.1.2.3 \Rightarrow DestPort \in \{80\}[p = 1/2]$) is pruned during validation step. This reduces the *cover* value of attribute $DestPort$ from three to one instance. *Replacement* allows rules from the pool to substitute for pruned rules in order to increase coverage. Thus, r_3 is added to the rule set, increasing the attribute $Cover$ back to three instances in Table 5.2. In addition, assume a couple more data instances of Table 5.3 that lose coverage due to pruned rules r_2. Note that r_3 does not apply to these data instances. New rules are generated from these samples (Step 6 in Fig. 5.3) by constraining the rule consequent to include $DestPort$ values of d_4 and d_5 and generating the antecedent through matching attributes such that the rule is satisfied by these instances. An example rule that covers the two data instances is:

$$SrcIp = 128.1.2.3 \bigwedge SrcPort = 25 \Rightarrow DestPort \in \{80\}[p = 1/2]. \tag{5.19}$$

If the new candidate rule conforms to D_v, it is added to the final rule set R.

5.4 Hybrid Approach

In the previous sections, we presented *Weighting* and *Replacement* for increasing rule coverage. In this section, we present the *Hybrid* approach that chooses among the two techniques based on which one has higher coverage on the training data. *Weighting* rule set comprises of rules with high predictiveness but can have low belief. *Replacement* may constitute many rules with low predictiveness. Combining the two approaches can result in rules with low predictiveness and low belief, and thus avoided. Thus, *Hybrid* picks one or the other based on coverage.

The main steps of the hybrid approach are presented in Fig. 5.4. Rule sets R_w and R_r are generated in Steps 1 and 2 using algorithms of Figs. 5.2 and 5.3 respectively. Let c_w be the coverage of weighting rule set and c_i be the coverage of i^{th} iteration in *Replacement* (c_r after all iterations). Thus, coverage for pruning is c_1. It can be noted that $c_w \geq c_1$

Input: sample set (D_s), training set (D_t), and validation set (D_v)

Output: LERAD rule set R

1. generate rule set R_w using *Weighting* steps of Fig. 5.2
2. generate rule set R_r using *Replacement* steps of Fig. 5.3
3. compute c_w = attribute value coverage of R_w over D_t
4. compute c_r = attribute value coverage of R_r over D_t
5. if $(c_w \geq c_r)$

   $R \leftarrow R_w$

   else

   $R \leftarrow R_r$

Figure 5.4: Hybrid approach in LERAD

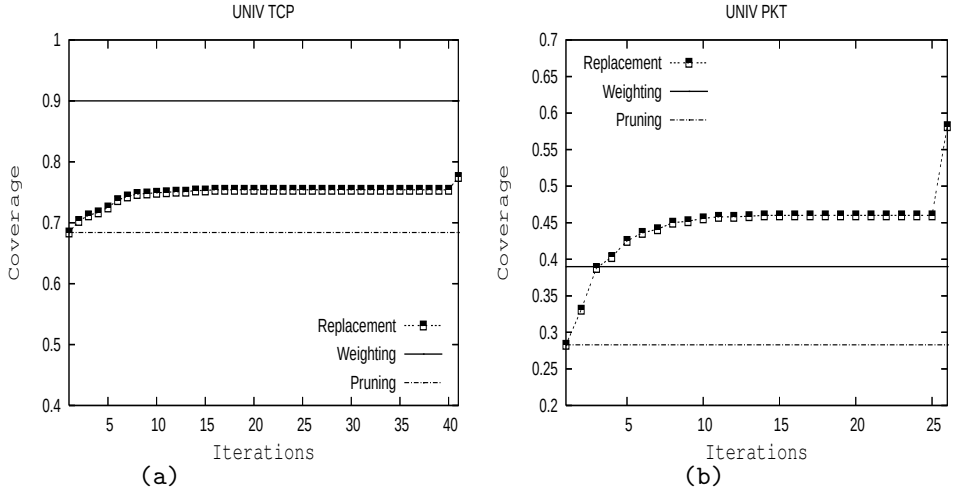

Figure 5.5: Coverage comparsion for *Pruning*, *Weighting* and *Replacement* for (a) UNIV TCP and (b) UNIV PKT.

and $c_{i+1} \geq c_i$. But the relation between c_w and c_n depends on the data set, as depicted in Figs. 5.5a-b. Fig. 5.5a shows coverage over all iterations for UNIV TCP data set (data set details presented in Section 5.5.1). Rule weighting has higher coverage than *Replacement*. On the other hand, *Replacement* betters the coverage of *Weighting* in the fourth iteration for UNIV PKT data set, as evident from Fig. 5.5b. Assuming higher coverage leads to higher accuracy, the *Hybrid* approach selects the technique with the higher coverage. Anomalies are scored using Eq. 5.8 for *Replacement* or Eq. 5.18 if *Weighting* is selected. For *Replacement*, a jump in coverage can be noted during the last iteration in Fig. 5.5. This increase is due to new candidate rules learned from attribute values that were perviously not covered (Steps 6-8 in Fig. 5.5).

5.5 Empirical Evaluation

In this section, we evaluate and compare the *Pruning*, *Weighting*, *Replacement*, and *Hybrid* schemes for anomaly detection.

5.5.1 Experimental Data

In addition to the three data sets described in Chapter 4, we evaluated the techniques on the following two data sets:

1. The DARPA/Lincoln Laboratory intrusion detection evaluation network data set (IDEVAL) (Lippmann et al. 2000) contains 201 labeled instances of 58 attacks. Since one day of inside traffic is missing, and there are one queso and four snmpget attacks against the router which are not visible from inside the local network, the total number of detectable attacks is 185. Refer (Kendell 1999) for attack taxonomy.

2. Over 600 hours of traffic collected on a university departmental server (UNIV) over 10 weeks, comprising of six labeled attacks - port/security scan from inside the firewall, an external HTTP proxy scan, an external DNS version probe, Nimda HTTP worm, Code Red II HTTP worm, and the Scalper worm. The port/security scan has two parts; first an attempt to retrieve the password file by a cgi-bin/htsearch exploit, followed by a port scan, with open ports probed further to test for vulnerabilities.

5.5.2 Experimental Procedures

We considered three attribute sets for each of the two network data sets: reassembled TCP streams (TCP) which reads attributes of the inbound side of unsolicited (client to server) reassembled TCP sessions; inbound client IP packets (PKT) which uses the first 32 pairs of bytes in each IP packet as attributes. The data sets will hereafter be referred to as IDEVAL TCP, IDEVAL PKT, UNIV TCP, and UNIV PKT respectively. For the IDEVAL data, we performed training on week 3, which contains no attacks, and testing on weeks 4 and 5. For UNIV data, we tested on weeks 2 through 10, using the previous week as training. By chance, there are no known attacks in week 1. However, there are generally attacks in the training data which could mask detections in the test data.

For host based data sets, we used system calls and related attributes to create application-based models, consisting of return value, error status and other arguments. Only BSM data set had complete argument information. For the UNM and FIT-UTK datasets, the sliding window of contiguous system calls was used, with a window size of 6, as this is claimed to give best results (Warrender, Forrest, and Pearlmutter 1999).

5.5.3 Evaluation Criteria

We evaluate and provide comparison for accuracy of models, computational and storage overheads.

Accuracy. For IDEVAL data set (both network and host), an attack is counted as detected if one or more alarms identifies the target address within 60 seconds of any portion of the attack (same as the 1999 DARPA evaluation criterion). Any other alarm is a false alarm. For the UNIV network traffic, we use the criterion that the technique must exactly identify at least one of the packets or TCP sessions involved in the attack. For the UNM and FIT-UTK host data sets, flagging an anomaly anywhere within the attack trace was used to be consistent with previous evaluations.

A Receiver Operating Characteristic (ROC) curve is an effective representation for model evaluation. We use ROC curves for studying the trend in percentage of attacks detected at different false alarm rates. We also list the areas under the ROC curve, where higher area implies better performance (Flach 2004). The area under the curve is normalized for the false alarm rate. Since the drawback of anomaly detection is the generation of false alarms, we focus on small false alarm rates (up to 1%).

Storage and Computational Overhead. To evaluate the viability of our technique for online usage, we measure its space and computational requirements. The storage overhead includes the size of the stored model, i.e. rules learned. We also measure the CPU time during the training and testing phases to determine the effectiveness of the techniques.

5.5.4 Accuracy of Weighting, Replacement and Hybrid

The ROC curves for the *Pruning*, *Weighting*, *Replacement* and *Hybrid* variants of LERAD are presented in Fig. 5.6. The respective areas under ROC curve are listed in Tables 5.4, 5.5 - values greater than that of *Pruning* are highlighted. The values in the table are not the Y

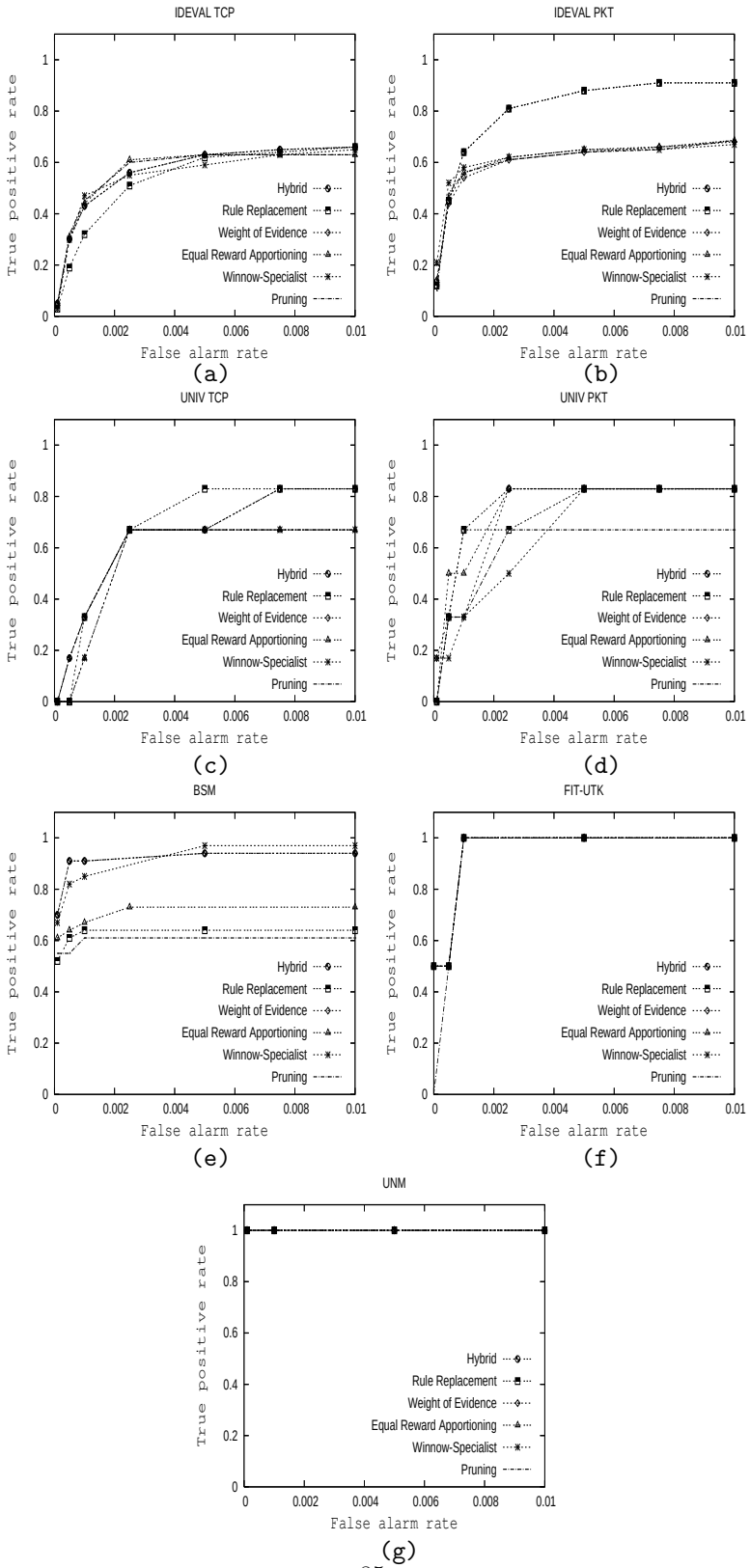

Figure 5.6: ROC (upto 1% false alarm rates) for *Pruning*, *Weighting*, *Replacement* and *Hybrid*.

axis (detection rate) on the ROC curve, but represent the percentage of the maximum area under the curve upto the respective false alarm rate. The random detector has the same false alarm rate and true positive rate for any threshold (x=y line for ROC). Since *Weight of Evidence* had the highest accuracy in weighting, it was used in conjunction with *Replacement* for *Hybrid*.

Figs. 5.6a-b present the ROC curves for the IDEVAL network data. Fig. 5.6a suggests that all techniques generally detect same number of attacks for IDEVAL TCP. Even their area under ROC curves are close to each other. But looking at the actual number of detections at 1% false alarm rate, *Weighting* and *Hybrid* variants detected 7 new attacks for IDEVAL TCP; whereas six extra attacks were detected by *Replacement*. For IDEVAL PKT, *Replacement* and *Hybrid* improved the AUC of *Pruning* by 33%.

For the UNIV data set (Figs. 5.6c-d), *Weight of Evidence* generally outperformed *Pruning* in all cases. *Replacement* had higher AUC than *Pruning* and all weighted variants at 1% false alarm rate. *Hybrid* selected *Weighting* for UNIV TCP and *Replacement* for UNIV PKT because of higher coverage on the resepective data sets. *Weighting*, *Replacement* and *Hybrid* schemes detected one more attack than *Pruning* for both UNIV data sets - the *Code Red II worm* was detected using tcp streams whereas the packet data detected the *DNS version probe*. An interesting observation for all LERAD variants on IDEVAL and UNIV network data sets was that PKT data detected more attacks than TCP data for all false alarm rates $\leq 1\%$. Evaluating the attacks detected by TCP and PKT, we saw a significant overlap between the two with some attacks being detected by only one of the attribute set.

The ROC curves for the host datasets are presented in Figs. 5.6e-g. For the BSM data, *Weight of Evidence* detected most attacks at 0.1% false alarm rate (approx. 50% more attacks than *Pruning*) whereas *Winnow-Specialist* had maximum area under curve at 1% false alarm rate, detecting 60% additional attacks than *Pruning*. *Replacement* did marginally better than

Table 5.4: Area under ROC curve (in %) upto 0.1% false alarm rate. Results better than *Pruning* are in bold-face. Random detector has area = 0.05% (at 0.1% false alarm rate).

Data set	**0.1% False Alarm Rate**					
	Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*	*Replacement*	*Hybrid*
IDEVAL TCP	27.2	26.2	26.5	25.8	17.5	25.8
IDEVAL PKT	38.6	**44.2**	**38.9**	37.5	**39.9**	**39.9**
UNIV TCP	15.9	4.3	4.3	15.9	8.3	15.9
UNIV PKT	23.1	21.0	**35.0**	**28.2**	**31.6**	**31.6**
BSM	56.5	**78.3**	**63.9**	**84.7**	**59.1**	**84.7**
FIT-UTK	50.0	**95.0**	**95.0**	**95.0**	**95.0**	**95.0**
UNM	100.0	100.0	100.0	100.0	100.0	100.0
Number of times better/tie/worse than Pruning	—	3/1/3	4/1/2	3/2/2	4/1/2	4/2/1
Average improvement (%) over Pruning [excluding UNM]	—	9.6	13.3	25.7	8.6	29.2

Table 5.5: Area under ROC curve (in %) upto 1% false alarm rate. Results better than *Pruning* are in bold-face. Random detector has area = 0.5% (at 1% false alarm rate).

Data set	**1% False Alarm Rate**					
	Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*	*Replacement*	*Hybrid*
IDEVAL TCP	57.5	55.8	57.5	57.3	54.1	57.3
IDEVAL PKT	61.1	**62.1**	**61.8**	61.0	**81.1**	**81.1**
UNIV TCP	59.3	57.0	57.0	**65.3**	**68.6**	**65.3**
UNIV PKT	60.1	**66.5**	**75.7**	**73.8**	**76.7**	**76.7**
BSM	60.6	**92.7**	**71.6**	**92.5**	**63.5**	**92.5**
FIT-UTK	62.5	**96.3**	**96.3**	**96.3**	**96.3**	**96.3**
UNM	100.0	100.0	100.0	100.0	100.0	100.0
Number of times better/tie/worse than Pruning	—	4/1/2	4/2/1	4/1/2	5/1/1	5/1/1
Average improvement (%) over Pruning [excluding UNM]	—	18.8	15.9	23.3	21.5	29.0

Pruning, with only one additional attack detection. *Hybrid* had same accuracy as *Weighting* and a total of 12 new attacks were detected (at 1% false alarm rate), including *fdformat*, *ffbconfig*, *guest*, *syslogd*, *httptunnel*, 4 distinct *secret* attacks, *portsweep*, *eject* and *selfping* exploits. On the FIT-UTK data, *Weighting*, *Replacement* and *Hybrid* had greater area under ROC curve than *Pruning* at 0.1% and 1% false alarm rates, though all techniques successfully captured the 2 malicious macro executions at 1% false alarm rate. Accuracy was the same for the UNM data set, where all techniques detected 3 attacks.

The second last row of Tables 5.4- 5.5 lists the number of times the respective strategy did better, same and worse than *Pruning*. Results indicate that *Weighting*, *Replacement* and *Hybrid* generally have greater accuracy than *Pruning*. This suggests that the rules discarded by LERAD might be effective in detecting attack based anomalies. At 0.1% false alarm rate, *Hybrid* outperformed *Pruning* four times and was worse once. *Equal Reward Apportioning* and *Replacement* also had higher AUC on four data sets but lower AUC on 2 data sets. At 1% false alarm rate, *Equal Reward Apportioning*, *Replacement* and *Hybrid* had same or better accuracy than *Pruning* on six occassions, and were worse only once; whereas *Weight of Evidence* and *Winnow-specialist* had higher AUC five times and lower AUC for two data sets. Overall, *Hybrid* had better accuracy than *Pruning* on most data sets at both 0.1% and 1% false alarm rates.

Though the second last row of the table gives us the number of times our techniques better, same or worse than *Pruning*, it fails to capture the magnitude of gain or loss in accuracy. This is captured in the last row of the tables, which denotes the average percentage of improvement in the AUC over *Pruning*, and is defined as:

$$\frac{1}{|datasets|} \sum_{|datasets|} \left(\frac{AUC_x - AUC_{Pruning}}{AUC_{Pruning}}\right) \times 100, \tag{5.20}$$

where $x \in \{Weighting, Replacement, Hybrid\}$. The UNM data set is excluded from calculating the average improvement in Tables 5.4 and 5.5 because improvement over *Pruning*'s

Table 5.6: Coverage comparison of *Weighting* and *Replacement*

Data set	**Weighting**	**Replacement**
IDEVAL TCP	**0.91**	0.86
IDEVAL PKT	0.43	**0.64**
UNIV TCP	**0.90**	0.77
UNIV PKT	0.39	**0.58**
BSM	**0.87**	0.85
FIT-UTK	**0.87**	0.85
UNM	0.88	0.88

100% AUC is not possible and all the methods have 100% AUC (no loss in accuracy). Among the weighted variants, *Weight of Evidence* had the best accuracy among all weighting techniques - an improvement of 25.7% and 23.3% over *Pruning* at 0.1% and 1% false alarm rate respectively. *Replacement* had a gain of 8.6% at 0.1% false alarm rate, compared to an improvement of 21.5% at 1% false alarm rate. But *Hybrid* performed the best, with the most average improvement in accuracy of about 29%.

5.5.5 Coverage vs. Accuracy

We measured the coverage of *Weighting* and *Replacement* techniques on the various data sets. The results are compiled in Table 5.6, where bold values are better. Results show that *Weighting* has higher or same coverage except both PKT data sets (IDEVAL and UNIV), where *Replacement* had higher coverage.

The *Hybrid* approach selects *Weighting* or *Replacement* based on coverage on training data. It assumes higher coverage yields higher accuracy (AUC). Is this assumption supported? Which data sets have higher accuracy with higher coverage at 0.1% and 1% false alarm rates,

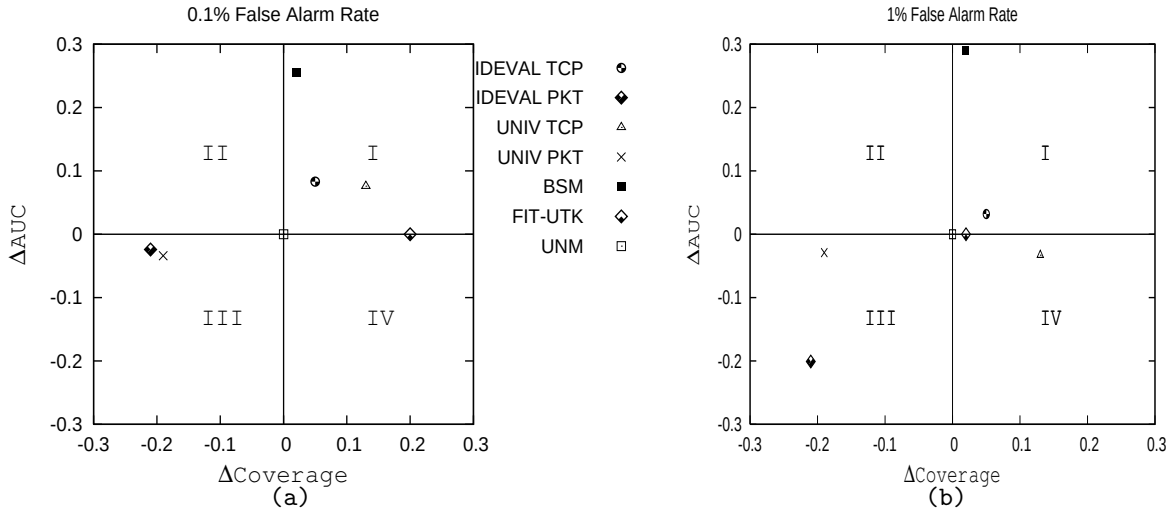

Figure 5.7: Accuracy vs. Coverage at (a) 0.1% and (b) 1% false alarm rates. X-axis represents difference in coverage and Y-axis is the difference in AUC for *Weighting* and *Replacement*.

and how does that explain the performance of *Hybrid*? Results are depicted in Fig. 5.7. The X-axis, denoted as $\Delta Coverage$, represents the difference in coverage of *Weighting* and *Replacement*:

$$\Delta Coverage = Coverage_{weighting} - Coverage_{replacement} \tag{5.21}$$

The Y-axis (ΔAUC) represents the difference in accuracy for the two techniques:

$$\Delta AUC = AUC_{weighting} - AUC_{replacement} \tag{5.22}$$

The correlation between accuracy and coverage is positive if increasing (decreasing) coverage increases (decreases) accuracy. The correlation is negative when increased (decreased) coverage leads to decreased (increased) accuracy. In Figs. 5.7a-b, positive correlations are represented by data points in quadrants I and III, whereas negative correlations are denoted by points on quadrants II and IV. Both techniques were identical for the UNM data set, hence marked as origin in Figs. 5.7a-b. IDEVAL TCP and BSM data sets are represented by data points in Quadrant I, where rule weighting had higher coverage and accuracy than *Replace-*

ment. *Replacement* did better (in terms of accuracy) than *Weighting* with increased coverage for IDEVAL PKT and UNIV PKT, and are represented by data points in Quadrant III. The only data set showing a negative correlation was UNIV TCP at 1% false alarm rate Fig. 5.7b. This is due to the fact that rule weighting had higher coverage than *Replacement* (0.90 vs. 0.77 - Table 5.6), but lower accuracy. Table 5.4 depicts accuracy of 15.9% for *Weighting* and 8.3% for *replacement* at 0.1% false alarm rate, and 65.3% and 68.6% respectively at 1% false alarm rate (Table 5.5). *Hybrid* picks the less accurate model for UNIV TCP at 1% false alarm rate. For the data set, *Weighting* has higher coverage over *Replacement*, but *Hybrid* has AUC of *Replacement* and not *Weighting*, as seen in Tables 5.4- 5.6. But for all other cases, the higher coverage selection by *Hybrid* yields higher accuracy. In our experiments, there is only one instance of negative correlation between coverage and accuracy, indicating that increased coverage generally increases accuracy. This supports our motivation for *Hybrid*, that for a given data set, algorithm with higher coverage yields higher accuracy.

But how does coverage affect accuracy across data sets, for the same algorithm? From Tables 5.4- 5.6 and Figs 5.7a-b, note that for *Hybrid*, the two TCP data sets have the highest coverage ($\geq$ 90%), but they have the least accuracy. This can be explained by the fundamental aspects that affect accuracy: (i) data representation, i.e. the features used; (ii) knowledge/model representation; and (iii) the learning algorithm, which finds the best model to fit the data. Since the knowledge/model representation (LERAD rules) and the learning algorithm (LERAD) are the same, the data representation attributes to lower AUC for TCP. Results indicate that TCP header doesn't model network data as well as PKT for intrusion detection. Since the exploits are not detectable at the TCP header level, any increase in coverage does not affect the accuracy. Thus, data sets with higher coverage might not have higher accuracy for a given algorithm. Our claim for *Hybrid* yielding higher accuracy with higher coverage is only applicable across techniques on the same data set.

Table 5.7: New attacks detected by weighting schemes at 1% false alarm rate.

Factor contributing to attack detection	**Data set: attack(s) detected**
Conformed rule(s) with increased rule *belief*	IDEVAL TCP: yaga, sechole
Violated rule(s) with reduced rule *belief*	IDEVAL TCP: arppoison, syslogd, perl, crashiis, secret UNIV TCP: codered UNIV PKT: bindver BSM: fdformat, ffbconfig,guest, syslogd, httptunnel, secret, portsweep, eject, selfping

5.5.6 Additional Attacks Detected by Weighting and Replacement beyond Pruning

Both *Weighting* and *Replacement* inherently differ on how the rule set coverage is increased over the training data. We conjecture that increased coverage can result in higher attack detections. In this section, we analyze additional attack detections and study if the additional rules are main contributors to new attack detections.

The increase in detections for all weighted variants (*Winnow-specialist*, *Equal Reward Apportioning* and *Weight of Evidence*) is caused by an increase in the anomaly score, which could result from: (a) increased belief for conformed rules, and/or (b) scores from rules discarded by *Pruning* but retained (with reduced belief) by the weighted variants. We analyzed the attacks detected using the 3 weighting schemes that were missed by *Pruning* at 1% false alarm rate. The results are listed in Table 5.7. Most of the new attacks detected are due to rules that were eliminated by *Pruning*, and support our claim for retaining the rules but reducing their belief in *Weighting*. The *Perl* attack was detected in IDEVAL TCP due to an anomalous

payload attribute that was part of the exploit. *Syslogd* is a Denial of Service attack that was flagged due to an invalid source whereas *crashiis* involved an unusual request. The *Code Red II* HTTP requests for /default.ida (GET /default.ida?NNNN...) in the UNIV TCP data set are captured by anomaly in the application payload. *fdformat* and *ffbconfig* vulnerabilities are buffer overflow attacks that are detected by encountering unusual arguments in the BSM data. The *syslogd* exploit violated a rule due to syslog segmentation fault.

Two attacks were detected by increasing the weight of existing rules: *yaga* is detected by long duration times due to the TCP connection not being closed after crashing and rebooting the target; whereas the *sechole* exploit is detected by an anomaly in the application payload. Rule weighting also reinforced the detection of attacks already detected by *Pruning*. This was attributed to large rule weights for some rules, resulting in further increase of the anomaly score. Also, there were multiple alarms for the same attack due to violation of rules introduced by the weighted variants but absent in *Pruning*.

Replacement substitutes violated rules with low predictiveness rules. Additional rules are also generated from attribute values previously not covered, as shown in Steps 6-8 of Fig. 5.3. The increase in attack detections for *Replacement* is thus attributed to (a) candidate rules replacing pruned rules, and (b) new candidate rules learned from attribute values that were not covered. We analyzed the attacks detected by the replacement scheme that were missed by *Pruning* at 1% false alarm rate. Table 5.8 lists all the new attacks detected in each of the two categories. As seen from the table, existing candidate rules that replace pruned rules are able to detect most new attacks.

5.5.7 Computational and Storage Overhead

Besides using different weight updation formulae, another distinction between the three weighting schemes discussed in Section 5.2.2 is the number of rules retained. *Winnow Special-*

Table 5.8: New attacks detected by *Replacement* at 1% false alarm rate.

Factor contributing to attack detection	**Data set: attack(s) detected**
Existing candidate rule(s) from rule pool	IDEVAL TCP: ppmacro, xsnoop, fdformat, xterm IDEVAL PKT: netcat_breakin, warez, sshtrojan, warezclient, portsweep, phf, tcpreset, sendmail, ipsweep, eject, processtable, perl, crashiis, apache2, guest, anypw, xterm, guest, snmpget, back, yaga, sqlattack, syslogd, guesspop UNIV TCP: codered UNIV PKT: bindver BSM: fdformat
New rule(s) from attribute values not covered earlier	IDEVAL TCP: selfping IDEVAL PKT: mailbomb, casesen, insidesniffer, dosnuke, xterm, perl, ncftp, selfping, loadmodule, ppmacro

Table 5.9: Computational overhead: training phase.

Data set	**No. of instances**	**Total training time (seconds)**					
		Pruning	*Winnow*	*Equal Reward*	*Wt. of Evidence*	*Replace*	*Hybrid*
IDEVAL TCP	35452	2.06	2.52	2.18	2.56	15.44	18.19
IDEVAL PKT	280281	3.98	6.27	7.86	12.60	496.89	514.92
UNIV TCP	141162	8.69	9.57	9.16	10.56	53.98	62.39
UNIV PKT	1305873	18.15	23.48	23.33	45.25	203.79	241.19
BSM	1261252	90.55	107.15	107.81	101.27	475.89	562.89
FIT-UTK	94759	0.91	0.95	0.98	1.27	1.08	1.45
UNM	3128	0.05	0.04	0.04	0.06	0.06	0.10

ist and *Equal Reward Apportioning* schemes suggest keeping all the rules that were previously discarded by *Pruning*, whereas *Weight of Evidence* keeps a subset thereof. *Replacement* reintroduces candidate rules and generates new ones from data not covered by initial candidate rules. This may result in larger rule sets and increased execution times for *Weighting*, *Replacement* and *Hybrid* techniques. To check the viability of the techniques for online usage, we studied the overhead involved with all the above techniques, both in terms of storage (size of rule set) and the CPU times for training and testing. Experiments were performed on a SUN Ultra 60 workstation with 450 MHz clock speed and 512 MB RAM.

The time requirements for training are listed in Table 5.9. Among the weighting methods, most notable difference existed for IDEVAL PKT data set, where *Equal Reward Apportioning* was twice and *Weight of Evidence* took thrice the time than *Pruning*. Compared to *Pruning* and *Weighting*, *Replacement* has significantly higher times, due to the high number of

Table 5.10: Storage requirements: size of rule set.

Data set	**Number of rules**					
	Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*	*Replacement*	*Hybrid*
IDEVAL TCP	52	71	71	71	160	71
IDEVAL PKT	100	108	108	106	554	554
UNIV TCP	45	88	88	88	94	88
UNIV PKT	48	80	80	75	293	293
BSM	155	176	176	176	359	176
FIT-UTK	11	12	12	12	12	12
UNM	36	36	36	36	36	36

iterations for the algorithm. Additional rules are also generated from data without rule set coverage. *Hybrid* is even worse, since it needs to learn both weighting and replacement models before selecting one based on higher coverage. In the worst case, the time taken by *Weighting* was approx. 45 μsec/instance for *Weight of Evidence* vs. 14 μsec/instance for *Pruning* in the case of IDEVAL PKT; *Replacement* took 1773 μsec/instance for the same data set. Since training can be performed offline, higher training times are acceptable. The time requirements for training can be reduced further by terminating the loop (Step 5 in Fig. 5.3) early, when there is no increase in coverage even though there might be rule violations in validation phase.

Storage of the model is determined by the number of rules in the rule set. Table 5.10 lists the number of rules generated for the various data sets. For all data sets except FIT-UTK and UNM, the number of rules is based on per week of data. Amongst the network data sets, the least overhead was obtained for IDEVAL PKT where the increase was roughly

6-8% for various weighting strategies. UNIV TCP presented the maximum overhead, where the number of rules almost doubled for all weighted schemes, and six times for *Replacement*. *Hybrid* selects *Weighting* or *Replacement* based on higher coverage on training data. Thus, the rule set would be same as *Weighting* or *Replacement*, depending upon the technique selected. *Weighting* is selected for all data sets except IDEVAL PKT and UNIV PKT, resulting in a smaller rule set for *Hybrid* than *Replacement*. Considering the large amount of data used during training (1-9 weeks) and the number of attributes involved, the size of the weighted and replacement rule sets formed is fairly reasonable. For *Weighting*, we could additionally limit the rule set size by eliminating a rule which has been violated a certain number of times or with weight below a threshold. For *Replacement*, rule set size can be reduced by terminating iterations earlier and/or ignoring new rules below a certain *predictiveness* threshold. Host data sets displayed lower storage overhead. For the BSM data, the weighted rule set was over 13% larger than *Pruning*. *Replacement* produced 359 rules compared to 155 for pruning. Since 11 different applications were modeled in BSM data, this corresponds to an average of 16 rules/application for weighting and 33 rules/application for *Replacement*, which is small for one week of training data. Number of rules was same for UNM data, whereas the weighted and replacement rule set size exceeded by one rule for FIT-UTK data set.

The time taken during test phase is also dependent on the rule set size. The more the rules, the higher is the number of sanity checks to be made for each test instance. Typically, the time taken should be low for online detection. The results obtained from our experiments are presented in Table 5.11. Due to larger rule sets, *Weighting*, *Replacement* and *Hybrid* schemes have longer execution times, making them computationally more expensive than *Pruning*. The maximum overhead for *Weighting* was 5.99 μsec/instance on UNIV TCP data set, and 19.32 μsec/instance for *Replacement* on IDEVAL PKT. Thus, the overhead is only a fraction of a millisecond per instance, reasonable for an online system.

Table 5.11: Computational overhead: testing phase.

Data set	**No. of instances**	**Total testing time (seconds)**					
		Pruning	*Winnow*	*Equal Reward*	*Wt. of Evidence*	*Replace*	*Hybrid*
IDEVAL TCP	178099	7.72	8.76	8.26	8.65	9.56	8.74
IDEVAL PKT	534763	3.02	3.90	3.68	3.20	13.35	13.22
UNIV TCP	143403	7.64	8.50	8.21	8.41	8.97	8.65
UNIV PKT	1310493	7.70	8.64	8.21	8.18	16.61	16.37
BSM	1889680	113.93	121.34	121.63	120.97	187.31	120.66
FIT-UTK	13745	0.10	0.10	0.10	0.09	0.10	0.10
UNM	7283	0.06	0.06	0.06	0.06	0.06	0.06

5.6 Summary

Machine learning research has been pursued to learn anomaly rules for intrusion detection. LERAD is one such algorithm that can characterize normal behavior in logical rules by finding associations among nominal attributes. It forms a small set of "easy to comprehend" rules that characterize the data. The algorithm is very efficient and effective in capturing anomaly based attacks. A separate *held-out* data is used to validate the rules. Any violations result in the rule being eliminated. We conjecture that discarding rules with possibly high coverage can lead to missed detections. In this paper we propose three techniques to increase rule coverage: *Weighting*, *Replacement* and *Hybrid*.

Weighting retains violated rules in the rule set and associates a belief value with each rule. Weights are representative of rule *belief* in our strategy. A conformed rule increases rule trust and hence the weight is increased. On the other hand, weight is decreased upon rule

violation. Three weighting schemes are presented - *Winnow-specialist-based weighting*, *Equal Reward Apportioning* and *Weight of Evidence*. *Replacement* collects rules ignored in coverage test in a rule pool. These rules are reevaluated to replace pruned rules and increase coverage. The steps of validation, pruning, and replacement are repeated until certain exit criterion is met. Furthermore, new rules are learned from remaining attribute values that were not covered. We also present *Hybrid* technique that selects between *Weighting* and *Replacement* based on higher coverage on training data.

We evaluated *Pruning*, *Weighting*, *Replacement* and *Hybrid* LERAD variants on various network and host data sets. Empirical results show that weighted and replacement rules detect more attack-based anomalies than pruning at less than 1% false alarm rates. The weighted strategies accounted for 7 more attack detections for IDEVAL TCP data set, whereas *Replacement* detected 6 extra attacks than *Pruning*. For *Weighting*, the most significant improvement was in the case of BSM data, where detected 12 new attacks (60% more than *Pruning*) were detected. *Replacement* performed best on IDEVAL PKT data set, where it detected 32% more attacks than pruning. At 0.1% false alarm rate, *Equal Reward Apportioning* outperformed *Pruning* in 5 data sets and generally performed the best. *Replacement* had the best accuracy on our data sets at 1% false alarm rate, where it did better than *Pruning* on seven data sets. Generally, all proposed techniques were better than *Pruning* in terms of AUC as well as number of attack detections at 1% false alarm rate.

We studied the effect of coverage on accuracy. Results indicate that increased coverage generally resulted in better accuracy. That is the reason why *Hybrid* did better than *Weighting* and *Replacement*, as shown in Fig. 5.7. We also analyzed the new attack detected by *Weighting* and *Replacement* based LERAD variants. For *Weighting*, these were attributed to high anomaly scores resulting from (a) violations of rules discarded by *Pruning* but retained by weighted variants with reduced belief; and (b) increased belief for existing rules due to the

weight update functions. The former factor contributed to most new attack anomalies. For *Replacement*, detections are attributed to (a) candidate rules replacing pruned rules, and (b) new candidate rules learned from attribute values that are not covered. Our analysis shows that most of the new attack anomalies are detected by the first factor.

We also computed overheads incurred due to *Weighting* and *Replacement*. Training times are generally higher for *Replacement* as it involves multiple iterations, the worst in our experiments being 1773 μsec/instance. For *Weighting*, it was 30 μsec/instance. But training can be performed offline. Since previously discarded rules are retained (for *Weighting*) and additional rules are added (for *Replacement*), rule sets tend to be larger. This has direct effect on the test phase - the larger the rule set, the higher is the testing overhead. But this overhead is minimal – 6 μsec/instance for *Weighting* and 19 μsec/instance for *Replacement*.

Chapter 6

Detecting Suspicious Behavior for Mobile Hosts

Mobile devices like cellular phones, smartphones and Pocket PCs are rapidly gaining popularity worldwide, featuring increased storage capacity, greater application support, and reduced price. Smartphones and PDAs are increasingly used by employees to connect to corporate networks, retrieve and store important data. Data stored on a device is often not encrypted and no security mechanism is adopted by the user. Losing a phone or having it stolen is currently the biggest risk that a mobile consumer faces (Dedo 2004; Goode 2006). Over 55 million mobile phones were estimated lost worldwide in 2006 alone and projections for the total number over the subsequent five years exceed 500 million handsets (Goode 2006). A lost device may contain personal and confidential company data that can be accessed by an unauthorized user. The device can be misused, leading to identity theft, data leakage (e.g. social security numbers of employees or customers), impersonation, and high service charges to the subscriber. Although the user can notify the loss to the network operator and have

the device disabled or data wiped out,

Wireless networks enable users to connect to a network without any physical connection. 802.11 wireless local area networks (WLANs) allow mobile hosts to join a network without being inside a building, hence bypassing traditional physical security. Publicly available tools like NetStumbler (Netstumbler 2004) can locate available WLANs while driving around in the neighborhood, called war driving (Flickenger 2003). Security experts have identified many WLAN attacks; they include ARP poisoning, MAC/IP spoofing, man-in-the-middle, session hijacking, and replaying (Peikair and Fogie 2003; Welch and Lathrop 2003). These attacks generally try to redirect traffic and masquerade as legitimate users with stolen identity.

This chapter focuses on security issues for mobile hosts, such as ones mentioned above. The goal is an automated system to detect suspicious behavior. Mobile hosts have an inherent property of movement across multiple locations. Considering the mobile nature of the problem, we propose STAD (Spatio Temporal Anomaly Detection). STAD monitors user location at different time intervals and learns a probabilistic model. We assume that the device is used by a single user, and will interchangeably use the terms *user* and *device*. Our goal is to learn mobility patterns for a user, who would generally be at the same location at a given time. For example, an employee is typically at work from 9 a.m. to 6 p.m. during weekdays, and at home most of the remaining time. Even the weekends may generate potential patterns like weekly kids' soccer games, and grocery shopping. A machine learning algorithm could model these contexts and make predictions to determine any anomalies. A lost or stolen phone, or an unauthorized WLAN user, might result in contextual anomalies for the given model, either in terms of location, or time, or both.

We present the problem as one dealing with sparse data, a well researched topic in natural language processing that has been applied to language modeling, text compression and information retrieval problems. In addition to the anomaly detectors, we present a Kullback-

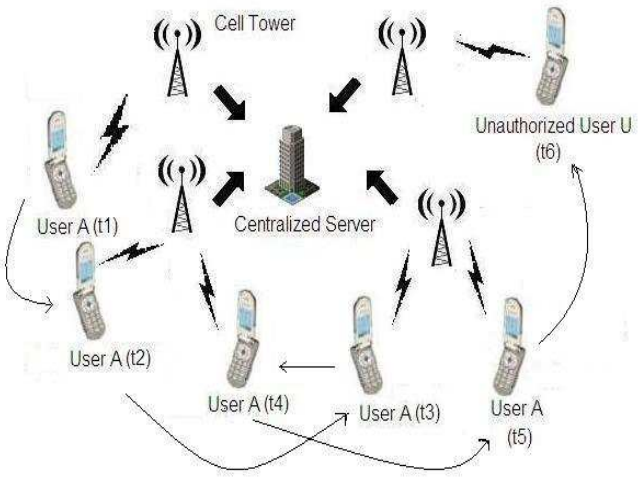

Figure 6.1: Mobile Anomaly Detection: For authorized user A, mobile context is learned over a period of time $(t1 - t3)$. Subsequent locations (at times $t4 - t6$) are validated against the model. An unauthorized user U is likely to be inconsistent with the spatio-temporal model for user A.

Leibler divergence based agglomorative hierarchical clustering to merge day profiles for reduced storage. We evaluate and compare STAD with Markov chains on real WLAN and mobile phone data sets. Results show that our proposed schemes are effective in capturing spatial temporal anomalies introduced by an unauthorized user, and incur minimal computational overhead for online usage.

6.1 Detecting spatio temporal anomalies for mobile hosts

This section introduces the framework for detecting abnormalities attributed to unauthorized users. We describe how probabilities are learned and anomalies are scored. The scenario is represented in Fig. 6.1. For an authorized user A, cell tower communicates with a mobile device and captures its location. The information is sent to a centralized server, where

a model is learned for the device location over different time intervals. The device may communicate with multiple cell towers, but all contextual information is routed to the server for model learning. Though cell ID data lacks the physical topology and proximity of the actual locations, it is a reasonable location estimate and easy to extract. In Fig. 6.1, the model is created for locations at time instances $t1 - t3$. The learned model is then used to maintain conformity for device (time $t4 - t6$ in the figure). Now consider the possibility of device theft by unauthorized user U. Any subsequent usage (Fig. 6.1 time instance $t6$) by U would most likely be inconsistent with the model in terms of location and time, raising an alarm. The device can then be locked by the network operator using a PIN that can only be unlocked by the operator or the authentic user via a challenge-response mechanism. Recent work (Li et al. 2007) deals with detecting suspicious large moving objects, such as ships. Though it involves route modeling, it deals with additional attributes like speed and direction information generally not available on laptops and mobile phones. Even if a GPS (global positioning system) was available on these devices, an intruder will likely disable it to evade such systems.

We propose tracking the frequency of the mobile phone at various locations within a fixed time interval. The frequency is then normalized across all possible locations and probability approximated at each of those locations during the time period. The probability of a mobile phone m at location l during time interval t is estimated by:

$$P_m^t(l) = \frac{freq_m^t(l)}{\sum_l freq_m^t(l)} \tag{6.1}$$

P_m^t is called the **spatial probability distribution** of m at time t. To reduce the data size and complexity and ease computation, we suggest using time intervals. For example, an interval size (δ) of 10 minutes results in 144 intervals per day (η). This creates a profile for a single day. Thus, for any day of week d, a profile consists of η spatial probability distributions

(Eq. 6.1) denoted formally as

$$Profile_m^d = (P_m^{d,1}, P_m^{d,2}, \cdots, P_m^{d,\eta}) \tag{6.2}$$

where $d \in D = \{Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday\}$.

During the monitoring (test) phase, we use the learned profile (*spatial probability distributions*) to estimate the likelihood of each data record in the test set. To include some state information of where the mobile phone was previously, we consider a time window W previous to the current time instance to estimate the probability of current location l_c at current time instant t_c. Time window W is measured by number of minutes in this paper and is a parameter to our algorithm. Let w be the number of time instances (data records) in W (minutes). Let t_{c-w+1} to t_c be the w instances in time window W. We denote $P_m^{d,W}(l_{c-w+1}, l_{c-w+2}, \cdots, l_c)$ as $P_m^{d,W}(l_W)$, and approximate it using probability chain rule as:

$$\begin{aligned} P_m^{d,W}(l_W) = \quad & P_m^{d,t_c}(l_c|l_{c-1}, \cdots, l_{c-w+1}) \times \\ & P_m^{d,t_{c-1}}(l_{c-1}|l_{c-2}, \cdots, l_{c-w+1}) \times \\ & \cdots \times P_m^{d,t_{c-w+1}}(l_{c-w+1}) \end{aligned} \tag{6.3}$$

The probability of a sequence of states is thus denoted as the product of probabilities of a state conditioned upon the previous states in the sequence. Storing all such probability values imposes an overhead and also increases the computational complexity.

6.1.1 Naive Approach

For simplicity and because the independence assumption of the Naive Bayes classifier generally seems to work well (Domingos and Pazzani 1997), we assume independence between subsequent locations, resulting in $P_m^{d,t_c}(l_c|l_{c-1}, \cdots, l_{c-w+1}) = P_m^{d,t_c}(l_c)$. The likelihood of mobile phone m over the time window W is thus approximated as the product of the marginal

probabilities:

$$P_m^{d,W}(l_W) = \prod_{i=c-w+1}^{c} P_m^{d,t_i}(l_i) \tag{6.4}$$

To avoid the underflow in multiplication, we use log likelihood instead:

$$log(P_m^{d,W}(l_W)) = \sum_{i=c-w+1}^{c} log(P_m^{d,t_i}(l_i)) \tag{6.5}$$

For anomaly detection systems, an anomaly score denotes the degree of abnormality for the test data instance. An anomaly score can be calculated for m and location l_c using the negative log likelihood of aggregated spatial probability distribution over a window W:

$$\begin{aligned} AnomalyScore_m^{d,W}(l_W) &= -log(P_m^{d,W}(l_W)) \\ &= -\sum_{i=c-w+1}^{c} log(P_m^{d,t_i}(l_i)) \end{aligned} \tag{6.6}$$

The lower the likelihood of a location given the current context, the higher is the anomaly score.

The independence assumption of the Naive approach is not always valid, since to get to a specific location one typically traverses a fixed set of locations. The assumption is relaxed with Markov Chains, described next.

6.1.2 Markov Chain

In Markov Chain, the current state depends only on the previous state. This technique involves a probability transition matrix comprising of single step transition probabilities for all observed states. The spatial probability distribution of Eq. 6.1 is modified as

$$P_m^t(l_j|l_k) = \frac{P_m^t(l_j,l_k)}{P_m^t(l_k)} = \frac{freq_m^t(l_j,l_k)}{\sum_{k'} freq_m^t(l_k,l_{k'})} \tag{6.7}$$

For the Markov Chain, $P_m^{d,t_c}(l_c|l_{c-1},\cdots,l_{c-w+1}) = P_m^{d,t_c}(l_c|l_{c-1})$. The probability estimate for the sequence of traversed states of Eq. 6.3 is now revised as

$$P_m^{d,W}(l_W) = \quad P_m^{d,t_c}(l_c|l_{c-1}) \times P_m^{d,t_{c-1}}(l_{c-1}|l_{c-2}) \times \cdots \times P_m^{d,t_{c-w+1}}(l_{c-w+1}) \tag{6.8}$$

Log likelihood is used to prevent underflow and the modified anomaly score is the negative log likelihood of aggregated spatial probability distribution:

$$\begin{aligned} AnomalyScore_m^{d,W}(l_W) &= -log(P_m^{d,W}(l_W)) \\ &= -logP_m^{d,t_{c-w+1}}(l_{c-w+1}) - \sum_{i=c-w+2}^{c} log(P_m^{d,t_i}(l_i|l_{i-1})) \end{aligned} \tag{6.9}$$

The Naive approach in Sec. 6.1.1 can be considered as zero-order Markov Chain, whereas this section describes first-order Markov Chain. We limit ourselves to lower order Markov Chains for ease of computation. Higher order Markov Chains can also be considered with higher time and space complexity.

6.1.3 STAD

A mobile phone could be within the proximity of multiple cell towers, as shown in Fig. 6.2. The signal strength may vary due to distance and undergo attenuation due to obstructions such as buildings. The mobile phone may transmit to the cell tower with the maximum signal strength ($C3$ in the figure). Similarly, a 802.11 based laptop may be in the vicinity of multiple access points, and may connect to one with the highest signal strength from the current orientation, or the optimal access point for load balancing. Thus, a physical location may correspond to multiple cell towers or access points. Though we expect to aggregate information for connectivity with all cell towers corresponding to a single location over a period of time, the frequency information may not always be an accurate estimate. Both the approaches discussed so far can flag valid but low frequency events as anomalous, resulting in higher false alarms. Next, we present an approach called STAD (Spatial Temporal Anomaly Detection) to alleviate false alarms due to low probability events. STAD only considers novel events in calculating the anomaly score. That is, no matter how frequent/likely an event has been observed, it is considered normal and has no contribution to the anomaly score. For

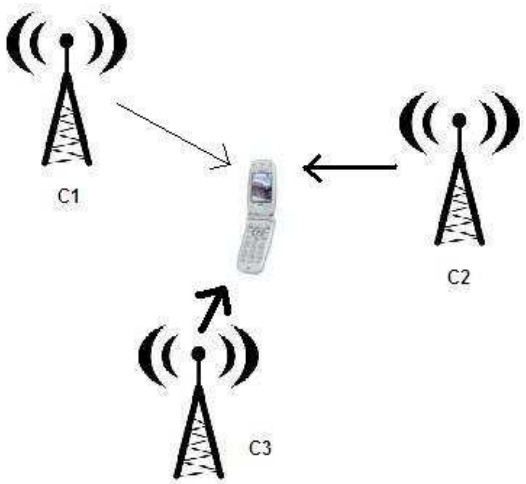

Figure 6.2: A mobile device may transmit to a subset of cell towers in proximity, resulting in inaccurate probability estimates.

example, an employee heading to work uses an alternate route when there is traffic congestion on the regular route. A low occurrence frequency may still flag the event as anomalous using naive and Markov Chain approaches, resulting in a false positive. But it would be deemed normal in STAD. Thus, cell towers $C1$ and $C2$ in Fig. 6.2 would also correspond to valid locations despite low frequency.

For STADn (or n^{th} order STAD), only smoothed probability for novel event $P_m^{d,W}(l_i|l_{i-1}\cdots l_{i-n})$ is estimated. We investigate zero and first order STAD in this paper, called STAD0 and STAD1 respectively. STAD0 maintains the subsequent location independence assumption of Naive approach, whereas the assumption is relaxed in STAD1 and the current state depends on the previous state. STAD assigns negative logarithm of the novel probability estimate as the anomaly score for the current test instance. For STAD0, it is

$$Score0_i = \begin{cases} -log(P_m^{d,t_i}(l_i)), & \text{if } freq_m^{d,t_i}(l_i) = 0 \\ \\ 0, & \text{otherwise} \end{cases} \tag{6.10}$$

The anomaly score for l_c is aggregated over a time window W (l_W) of w instances to ignore spurious anomalies. For STAD0, it is computed as:

$$AnomalyScore_m^{d,W}(l_W) = \sum_{i=c-w+1}^{c} Score0_i \tag{6.11}$$

For STAD1, which assumes that the current state is dependent on the previous, the anomaly score for the current test instance is calculated as:

$$Score1_i = \begin{cases} -log(P_m^{d,t_i}(l_i|l_{i-1})), & \text{if } freq_m^{d,t_i}(l_i, l_{i-1}) = 0 \\ \\ 0, & \text{otherwise} \end{cases} \tag{6.12}$$

The anomaly score for l_c over window W (l_W) is calculated as:

$$AnomalyScore_m^{d,W}(l_W) = Score0_{c-w+1} + \sum_{i=c-w+2}^{c} Score1_i \tag{6.13}$$

An alarm is generated on exceeding a threshold.

6.1.4 AEMI (Augmented Expected Mutual Information)

Correlations between successive locations can also be learned using mutual information, and is computed as:

$$I(l_1, l_2) = p(l_1, l_2) log \frac{p(l_1, l_2)}{p(l_1)p(l_2)}. \tag{6.14}$$

Though mutual information considers the mutual dependence between the locations, it does not take into account their independent occurrence. Augmented Expected Mutual Information (AEMI) (Chan 1999) accounts for both the mutual dependence and independence of the random variables (locations in our case). It is calculated as:

$$AEMI(l_1, l_2) = I(l_1, l_2) - \sum_{(A=l_1, B=\bar{l_2}), (A=\bar{l_1}, B=l_2)} I(A, B). \tag{6.15}$$

The first term provides the supporting evidence for the contextual succession and dependence for l_1 and l_2, whereas subsequent terms subtract counter evidence. Higher AEMI values

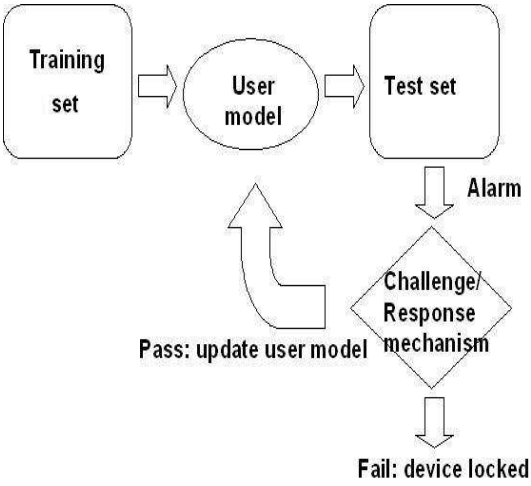

Figure 6.3: Adaptive user modeling for valid novel locations for authorized user.

indicate higher dependence between the random variables. We also experimented with a spatio-temporal anomaly detector based on joint probability of successive locations using AEMI as the correlation function. The anomaly score is aggregated over the window W as:

$$AnomalyScore_m^{d,W} = -\sum_{i=c-w+1}^{c-1} AEMI(l_i, l_{i+1}) \tag{6.16}$$

6.2 Adaptive Modeling

In addition to real-time monitoring, a desirable property of anomaly detectors is adaptability. A researcher attending a conference in a new city will constantly violate the learned model, flooding the device with false alarms and overwhelming the user. To suppress false alarms, it is imperative for systems to be adaptive. One such system is depicted in Fig. 6.3, which invokes a challenge/response mechanism for user alarm. Once authenticated (i.e. false alarm), the spatial day profile of Eq. 6.2 is updated with the novel location and frequency incremented for revised smoothed probability estimates. For STAD, as explained in Sec. 6.1.3, models are built using the training set only. For adaptive-STAD, models are udpated with instances from

the test set once they are verified in an online manner. That is, performance of adaptive-STAD is still based on unseen instances in the test set, but once the instances are seen and verified, they are incorporated into the model. This enables the system to adapt to concept drift and reduce subsequent false alarms. If user cannot verify authenticity, the device may be locked.

6.3 Smoothing probability values

Our techniques use probability estimate of novel events to score anomalies. Variance reduction techniques are required to compute the non-zero unobserved event probability estimate. In the event of a novel location, spatial probability distribution underestimates the probability of the new value by assigning it a value 0, resulting in an undefined anomaly score (Eqs. 6.6, 6.9, 6.11, 6.13). This problem of data sparseness is similar to the one in maximum likelihood estimator of a language model in natural language processing.

Let s be the number of times a device was present at a specific location l in a given time interval. Thus, $s = freq(l)$. We denote n as the total frequency count $= \sum_l freq(l)$; and r as the number of distinct locations for the device. Furthermore, let f_k be the number of distinct locations with frequency count equal to k at a given context. It can be observed that $\sum_k f_k = r$ and $\sum_k k f_k = n$.

Witten and Bell (Witten and Bell 1991) studied different schemes to deal with the *zero* frequency problem in adaptive statistical coding applied to text compression . They found the following estimate (due to (Moffat 1988)) to give the best results:

$$P(l) = \begin{cases} \frac{s}{n+r}, & \text{if } l \text{ is observed} \\ \\ \frac{r}{n+r}, & \text{otherwise} \end{cases} \tag{6.17}$$

This is referred to as *Method C* in the original paper. The rationale is to increase the

probability of novel events with the number of distinct observations. For example, given two 10-integer sequences $S_1 = \langle 1, 0, 0, 0, 0, 1, 1, 0, 0, 0 \rangle$ and $S_2 = \langle 1, 2, 1, 0, 5, 7, -8, 12, 0, 9 \rangle$, S_2 is more likely to encounter a novel subsequent value than S_1, since there is more randomness and variability in S_2 than S_1. In Eq. 6.17, note that the sum of all smoothed probability values (including novel events) is unity.

6.4 Spatio-temporal context clustering

Our anomaly detectors create a model for each day of the week. Typically, a high volume of data is tracked by such a system in real time. Thus, small models are desired. Merging similar day profiles together reduces the storage overhead. For example, an executive assistant with a relatively same schedule from Monday to Friday may result in a succint model for all weekdays. Similarly, a professor teaching same classes on Tuesday and Thursday can have a similar model for those two days.

Each individual day profile is an object during clustering, and each day has the same representation as the day profile in Eq. 6.2. That is, for η intervals per day and N locations per interval, each object has η $N \times N$ spatial distributions. The distance between a pair of spatial distributions is estimated using Kullback Leibler divergence, which is a measure of the relative entropy between two distributions. Kullback Leibler divergence KL between spatial distributions p_1 and p_2 is calculated as:

$$KL^t(p_1||p_2) = \sum_i p_1^t(i) log \frac{p_1^t(i)}{p_2^t(i)} \tag{6.18}$$

Since Kullback Leibler divergence is not symmetric $[KL^t(p_1||p_2) \neq KL^t(p_2||p_1)]$, it cannot be used as-is to measure distance for clustering similar spatial distributions. Thus, we define distance $\Delta^t(p_1, p_2)$ between p_1 and p_2 as:

$$\Delta^t(p_1, p_2) = KL^t(p_1||p_2) + KL^t(p_2||p_1) \tag{6.19}$$

The Δ value is normalized in the computation above. It can be noted that Δ is symmetric for a given distribution pair p_1 and p_2. The total distance between a pair of profiles (objects) $Profile_m^{d_1}$ and $Profile_m^{d_2}$ for days d_1 and d_2 respectively is computed using Euclidean distance:

$$D(Profile_m^{d_1}, Profile_m^{d_2}) = \sqrt{\sum_{i=1}^{\eta} \Delta^i(p_i^{d_1}, p_i^{d_2})^2} \tag{6.20}$$

Our clustering method is an agglomorative hierarchical approach (Jain and Dubes 1988). All day profiles are disjoint clusters input to Alg. 1. During each iteration, two most similar clusters (profiles) are merged (Step 3). Cluster similarity is measured using the distance function in Eq. 6.20. Distance between exactly same profiles is zero. Merging entails combining the spatial distributions for the profiles with the minimum distance (Step 4). Individual spatial distributions are then replaced by the merged profile. Associated cluster distances are recomputed in Step 5. The stopping criterion for the algorithm (in Step 2) can be the number of desired clusters k or a distance threshold θ for merging most similar clusters.

Input: $\bigcup_{d \in D} Profile_m^d$, as defined in Eq. 6.2
Output: $\bigcup_{d' \subseteq D} Profile_m^{d'}$
Compute pairwise cluster distances using Eq.6.20

while *(stopping criterion not met)* **do**

  Identify clusters with minimum pairwise distance

  Merge minimum distance profiles into $Profile_m^{d'}$

  Recompute pairwise cluster distances with merged cluster

end

Algorithm 1: Clustering day profiles

6.4.1 Space complexity analysis of learned model

Clustering merges similar contexts and creates a succint learned model, which is used to validate data during testing phase. For STAD0, the space complexity of the stored model is

$O(\eta CN)$, where η is the number of time intervals per day, C is the number of clusters, and N is the number of locations per interval. For STAD1, the space complexity is $O(\eta CN^2)$. But the probability matrix is generally sparse and can be implemented more efficiently. AEMI has the same complexity as STAD1, since all counter-evidence from Eq. 6.15 is computed on the fly from the joint probability matrix. In the worst case, $C = C_{max}$, which is the same as no clustering. In our case, $C_{max} = 7$ (number of days/week).

6.5 Empirical Evaluation

6.5.1 Experimental Data and Procedures

To evaluate and compare the spatio-temporal anomaly detection techniques, we used publicly available real WLAN and mobile phone data. The WLAN data set comprises of syslog records collected from mobile users at Dartmouth College campus between April 1, 2001 and June 30, 2004 (Kotz, Henderson, and Abyzov 2005). MAC address, access point name, and associated timestamp information was logged. We divided the data set based on the type of most frequented locale (building). The locale where a user spends the most time is deemed as her home locale. The six locales we used were ACA (academic), ADM (admin), ATH (athletic), LIB (library), RES (residential) and SOC (social). We maintain a distinction between *same* and *other* locale – users from *same* home locale are expected to be harder to distinguish from self, whereas users from *other* locale should be more dissimilar from self. We conducted experiments on data from 5344 Dartmouth campus wireless users spanning over the 3 year period.

Over 500,000 hours of mobile phone data was collected at the MIT campus (Eagle and Pentland 2006) using the Context Phone framework (Raento et al. 2005). 89 different users, ranging from freshmen to graduate students to faculty members at the university, were the

subjects in the experiments. We extracted the location and time data - location corresponds to the cell id (for cellular networks), and timestamp was broken into *day of the week* and *time of the day* features. A sampling rate of 1 second/instance was used in our experiments.

To quantify the efficacy of the anomaly detectors, we used location data for all devices with at least one week of training data. Disjoint training and test sets were created for each user. Since we do not have explicit labels for bad behavior, for each mobile device, we pretend that the behavior of unauthorized users is similar to the other mobile devices. That is, given a trained model for a mobile device, test data from randomly selected 25 users were used to approximate behavior of unauthorized users. For the Dartmouth data, we also had the distinction between *same* and *other* locale. Thus, experiments were performed with 25 users selected from each of the two categories to represent malicious behavior. The confusion matrix is presented in Table 6.1. A test data sample from device B is validated against all models. Any alarm against model A is a true positive and against model B is a false alarm. No alarm against model B is a true negative, but not flagging an anomaly against model A is considered an undetected malicious attack on the device.

Time interval δ (Sec. 6.1.1) is a parameter to our techniques. Coarse-grained values reduce sparseness, but tend to include multiple contexts. On the other hand, fine-grained values are focused on specific context but exacerbate the issue of data sparseness, thereby increasing the number of false alarms. Larger interval sizes also accommodate spurious anomalies better than smaller intervals. We present results with δ=60 minutes, though values from 15-60 minutes yielded similar results. The time window W (Eq. 6.4) is another parameter for our techniques. Small W value flags anomalies at an early stage thereby minimizing loss, hence we chose W=10 minutes. A conservative distance threshold θ=0.5 was used for clustering.

Table 6.1: Confusion matrix in the context of anomaly detection for mobile devices

Actual	**Prediction**	
	Unauthorized user	Authorized user
Unauthorized user	True Positive	False Negative
Authorized user	False Positive	True Negative

6.5.2 Evaluation Criteria

Generating an alarm for any event corresponding to a malicious user is trivial. We used a stricter evaluation that **all** events from an unauthorized user are evaluated separately, and correctly detecting malicious usage for a single event (high anomaly score) does not automatically count as success for all other events from the same user. Thus, each event interval is evaluated independently of the others, thereby making it harder for the anomaly detectors. Computer security techniques are typically evaluated using a Receiver Operator Characteristic (**ROC**) curve that plots the rate of correct anomalies detected (i.e. different device) alongwith the false alarm (i.e. same device) rate. The area under the ROC curve (**AUC**) is also calculated. A larger AUC value is desired as it is representative of the total percentage of true positives detected at the cost of varied false alarm rates. In addition to model accuracy, we compared the time requirements and storage overhead.

6.5.3 Comparison of accuracy

We evaluated and compared the five anomaly detection techniques – Naive, Markov Chain (MC), STAD0, STAD1 and AEMI – on the Dartmouth 802.11 WLAN and MIT mobile phone data sets. For WLAN data, we maintain a distinction between *same* and *other* locale, as discussed in Sec. 6.5.1. The ROC curves are presented in Fig. 6.4. The random detector has the same false alarm rate and true positive rate for any threshold (represented by the diagonal

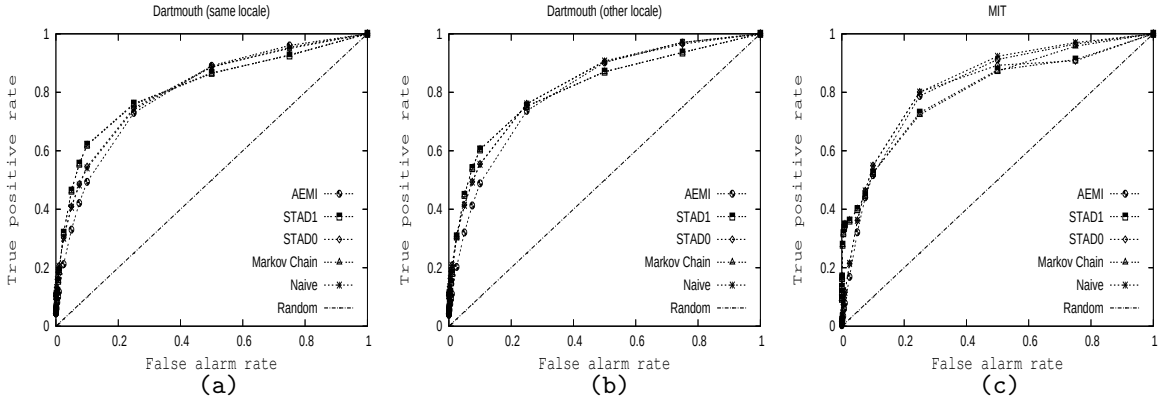

Figure 6.4: ROC curves for mobile anomaly detection.

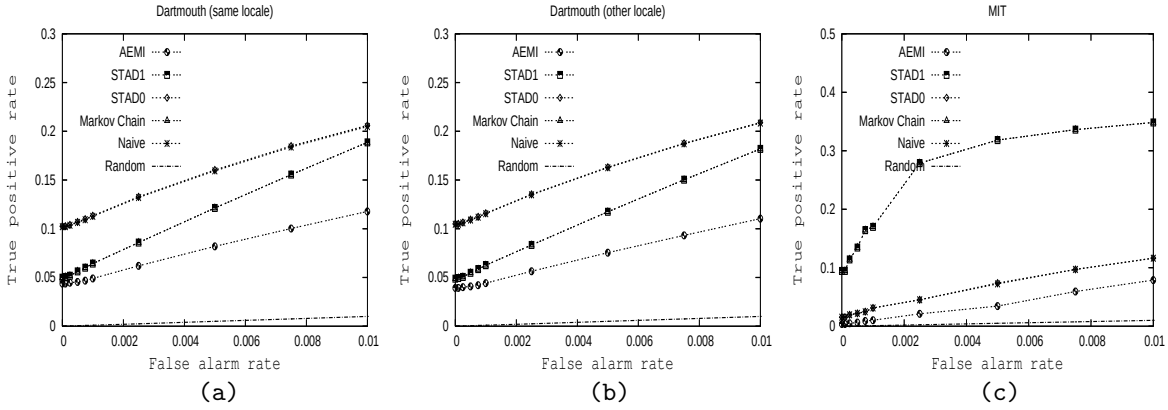

Figure 6.5: ROC curves for mobile anomaly detection with FAR ≤ 0.01.

$x = y$ in the ROC curve). The ROC curve for Dartmouth (same locale) in Fig. 6.4a indicates that STAD1 and Markov chain (MC) performed the best till 0.35 false alarm rate (FAR), after which AEMI had the highest accuracy. For Dartmouth (other locale) in Fig. 6.4b, STAD1 and MC had maximum detections at FAR $\leq$ 0.25, but were overtaken by STAD0, Naive and AEMI at higher FAR. A similar trend was noticed in the ROC for the MIT data set (Fig. 6.4c), where STAD1 does best below FAR=0.1, and thereafter surpassed by STAD0 and AEMI.

Since the drawback of anomaly detection is the generation of false alarms, we focus on small FAR. Hence we analyze the ROC curves up to 1% FAR in Figs. 6.5a-c. Accuracy, measured in terms of respective areas under the ROC curve (AUC), is listed in Table 6.2 for $FAR \in \{0.001, 0.01, 1\}$. AUC values in square brackets are with clustering. The random detector has AUC of 0.05×10^{-5}, 0.05×10^{-3} and 0.5 respectively. For Dartmouth (same locale), STAD1 and Markov perform best at FAR=1; whereas lower order techniques STAD0 and Naive had maximum detections at FAR=1 in Dartmouth (other locale) data. This result is expected, as users from same locale would be more similar and tracking transitions should distinguish users better. This is also consistent with Dartmouth (other locale), where STAD0 and Naive had maximum detections at FAR=1. But at FAR $\leq$ 0.01 (Figs. 6.5a,b), we noticed that STAD0 and Naive approach outperformed all other techniques irrespective of the locale. On the contrary, for MIT mobile phone data set, STAD1 and MC had highest accuracy at FAR=0.001,0.01.

In Sec. 6.4.1, we presented a theoretical analysis for reduced storage requirements with model clustering. It is thus imperative to study its effect on model accuracy. Table 6.2 lists the AUC values (within square brackets) for all methods with clustering. For Dartmouth data, the worst case AUC reduction was 2.5% for Naive with clustering at FAR=0.001. Remainder AUC values with clustering were same or better, with the highest increase of 10.8% for AEMI

Table 6.2: Area under ROC curve (AUC) upto various false alarm rates (FAR). AUC values with clustering is in square brackets. Bold values represent maximum AUC among all methods (without clustering) at given FAR.

Data set	Method	AUC		
		$\times 10^{-5}$	$\times 10^{-3}$	
		FAR=0.001	FAR=0.01	FAR=1
	Random	0.05	0.05	0.50
Dartmouth (same locale)	Naive	10.76 [10.48]	1.56 [1.56]	0.81 [0.81]
	MC	5.67 [6.10]	1.18 [1.25]	**0.82** [0.82]
	STAD0	**10.79** [10.61]	**1.57** [1.57]	0.81 [0.82]
	STAD1	5.67 [6.11]	1.18 [1.25]	**0.82** [0.82]
	AEMI	4.55 [4.61]	0.83 [0.92]	0.80 [0.80]
Dartmouth (other locale)	Naive	11.11 [10.58]	1.61 [1.60]	0.82 [0.82]
	MC	5.80 [5.97]	1.20 [1.20]	0.81 [0.81]
	STAD0	**11.32** [10.72]	**1.62** [1.60]	**0.83** [0.82]
	STAD1	5.80 [5.97]	1.20 [1.20]	0.81 [0.81]
	AEMI	4.30 [4.51]	0.74 [0.98]	0.81 [0.81]
MIT	Naive	2.20 [2.30]	0.71 [0.69]	**0.83** [0.83]
	MC	**13.50** [13.60]	**2.89** [2.86]	0.81 [0.81]
	STAD0	2.20 [2.30]	0.71 [0.69]	0.81 [0.81]
	STAD1	**13.50** [13.60]	**2.89** [2.86]	0.80 [0.80]
	AEMI	0.70 [0.70]	0.39 [0.38]	0.82 [0.82]

at FAR=0.01, followed by 7.7% increase for STAD1 at FAR=0.001. For MIT data, the maximum AUC reduction with clustering was 2.8% for Naive and STAD0 at FAR=0.01. But at lower FAR(=0.001), there was an AUC increase of 4.5%. In general, the percentage gain in accuracy was more with clustering as compared to no clustering.

The main questions posed by the results were: (i) why one set of techniques did better on Dartmouth data set and another set on the MIT data, and (ii) why AEMI performed the worst at $FAR \leq 1\%$ for all data sets. Answer for the first question can be explained by the inherent nature of mobility in the two data sets. Dartmouth data set as gathered from laptops, which typically have lower degree of mobility in comparison to the mobile phones of MIT data set. The mobile stations in WLAN are not as mobile as the users themselves; for instance, users do not carry their laptops to the restrooms, or a colleague's office for a quick chat. Mobile phones, on the other hand, are a wearable sensor that people generally carry with them all the time. Since the phone is more mobile than a laptop, STAD1 fared better than STAD0 on the MIT data set. Conversely, the higher AUC of STAD0 than STAD1 in Dartmouth data is attributed to limited mobility of the 802.11 devices such as laptops.

To understand the poor performance of AEMI at low FAR, we analyzed the false alarms further. For Dartmouth data, there were approx. 125% more novel joint probability values on an average compared to the marginal probability values that accounted for anomalies in test data. In hindsight, the behavior was expected. A device can be within range to multiple access points or cell towers at any instant. Thus, the frequency information may not be a true representation of the accessible access points or towers. The problem is exacerbated by extending it into higher order, where transitions between access points or cell towers are noted. Thus, training data for the example in Fig. 6.2 may only cover one transition ($C1 \rightarrow C2$), but the test data may comprise of other valid transitions, which would be wrongly flagged as anomalies. This resulted in a sparse joint probability space. The computation of AEMI

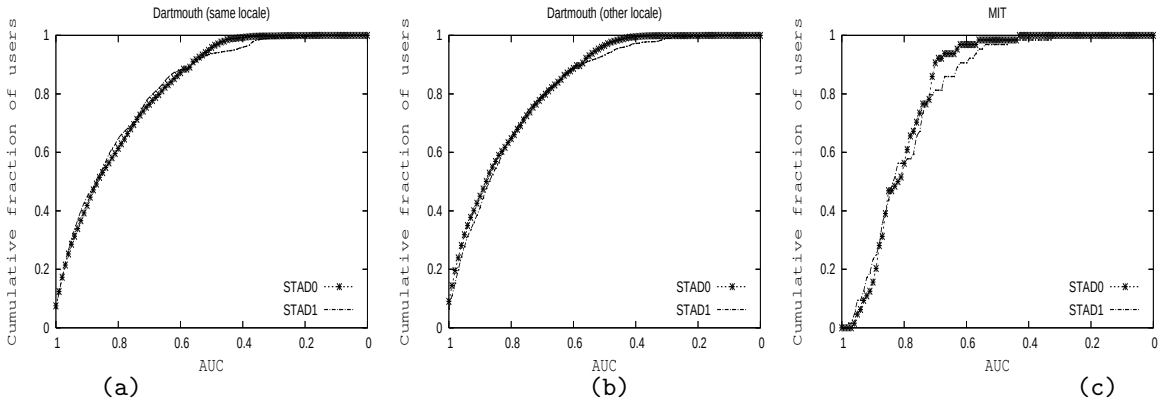

Figure 6.6: Cumulative fraction of mobile users for various AUC values.

comprises of three joint probability values (Eq. 6.15), resulting in poor performance.

Another observation from the ROC curves and AUC is the similarity in the accuracy of STAD0 with Naive method and STAD1 with MC. At less than 1% FAR, STAD0 demonstrated higher accuracy than Naive approach for Dartmouth (same locale) data, whereas performance was same in other two data sets. STAD1 and MC were at par with each other on all three data sets. These results support our claim for STAD that frequency of observed events may be ignored for computing the anomaly score. Further, for Dartmouth data, AUC for *o*ther locale was generally higher than *s*ame locale. This was expected since users from *s*ame locale, in contrast to *o*ther locale, are expected to be more similar to self and relatively harder to detect.

Different users behave differently. Some are more predictable than others. We studied the trend in the cumulative fraction of users above AUC value $\in$ [0,1]. Results are shown in Figs. 6.6a-c. Though the average AUC for all techniques at FAR=1 is in the range 0.80-0.83 (Table 6.2), the figure suggests appropriateness of STAD for some users with much higher AUC. For Dartmouth (same locale) data, approx. 42% users for STAD0 and 45% for STAD1 had AUC value exceeding 0.90. For Dartmouth (other locale), AUC $\geq$ 0.90 was achieved by 45% users with STAD0 and 41% of the users with STAD1. For MIT data, similar scores were

Table 6.3: AUC comparison for original and adaptive (ad-STAD) STAD.

Data set	Method	ad-STAD AUC [original STAD AUC]		
		$\times 10^{-5}$	$\times 10^{-3}$	
		FAR=0.001	FAR=0.01	FAR=1
Dartmouth (same locale)	STAD0	51.99[10.79]	6.94[1.57]	0.92[0.81]
	STAD1	4.29[5.67]	1.17[1.18]	0.87[0.82]
Dartmouth (other locale)	STAD0	98.37[11.32]	9.84[1.62]	0.98[0.83]
	STAD1	28.67[5.80]	12.00[1.20]	0.98[0.81]
MIT	STAD0	49.05[2.20]	7.35[0.71]	0.93[0.81]
	STAD1	32.90[13.50]	7.13[2.89]	0.93[0.80]

achieved by 16% and 24% users with STAD0 and STAD1 respectively. These results suggest applicability of our techniques to at least a significant percentage of users who demonstrate higher predictability in mobility patterns.

Adaptive STAD. An important aspect of security systems is adaptability. Similar to STAD, adaptive STAD (ad-STAD) validates unseen test data for model violations. Unlike STAD, it updates the model with locations that have already been seen and verified, as explained in Sec. 6.2. Table 6.3 lists and compares the AUC values of adaptive STAD (ad-STAD) with the original versions. Results show a significant reduction in the number of false alarms and increase in the number of detections in most data sets. The most significant improvement was for MIT data, which demonstrated a 22 fold increase in AUC for STAD0 at FAR=0.001. Generally, adaptive STAD0 performed the best with the highest AUC, indicating its effectiveness to adapt to concept drift and curtail false alarms.

Table 6.4: Average training and testing rates (μseconds/instance).

Data set	Technique	Training rate	Testing rate
Dartmouth	Naive	0.005	0.415
	MC	0.006	0.432
	STAD0	0.005	0.414
	STAD1	0.006	0.430
	AEMI	0.006	0.432
MIT	Naive	0.135	0.407
	MC	0.137	0.409
	STAD0	0.135	0.404
	STAD1	0.137	0.408
	AEMI	0.137	0.416

6.5.4 Time and space requirements

Time requirements for training and testing. For anomaly detection system to be effective, it should be able to detect misuse in real-time. For completeness, we computed time requirements for model creation (training) as well as model validation (testing) for the anomaly detection techniques. Experiments were performed on a 2GHz Pentium M processor, 1 GB RAM PC running Windows XP. The results are compiled in Table 6.4. Training was very fast, with fraction of a microsecond rates. In addition to these values, clustering added an average overhead of 27.68 μseconds/instance for Dartmouth data, and 77.36 μseconds/instance for MIT data set. Though training can be performed offline, but online testing is desired to minimize loss. For testing, STAD0 had minimum time requirements – 0.414 μseconds/instance for Dartmouth and 0.405 μseconds/instance for MIT data. This was expected as only novel events contribute towards scoring anomalies. Our techniques incur low

computational overhead, making them reasonable for an online system.

Reduced storage for trained model. We proposed context clustering using an agglomorative hierarchical approach with Kullback-Leibler-based distance metric in Sec. 6.4. Clustering collapses similar days into a single cluster, thereby reducing the space requirements. For Dartmouth data, clustering produced 5.94 clusters per user, i.e. a 15.14% reduction in model storage over approach with no clustering. For MIT data set, average number of clusters was 5.28, reducing storage by 24.57%.

6.6 Summary

To alleviate the problem of MAC spoofing and lost or stolen mobile devices, we presented an automated technique called STAD to detect spatial temporal anomalies for mobile devices. Our technique creates smoothed stochastic user models from training data and flags model violations on disjoint test data. We performed experiments on real WLAN and mobile phone data sets. Predictive accuracy was measured on unseen instances of the test set. STAD0 performs best when the degree of mobility is low, as in Dartmouth WLAN data. STAD1 had highest accuracy for higher degree of mobility, as one depicted in MIT mobile phone data. Over 40% users in Dartmouth data set and 16-24% in MIT data set had AUC ≥ 0.9, suggesting applicablity of our technique to a significant percentage of users. We used Kullback-Leibler divergence based agglomorative hierarchical clustering to merge profiles for reduced storage. In our experiments, there was a 15-25% reduction in the size of the learned model. The test time requirements for our technique was approx. 0.4 μsecond per instance, making it viable for online usage. We demonstrate the efficacy of adaptive-STAD to reduce false alarms and increase detections, with a 22-fold improvement for MIT data at FAR=0.001.

Chapter 7

Conclusions

Anomaly detection techniques complement signature-based methods for intrusion detection. Though capable of detecting novel attacks, they suffer from generation of false alarms. Network traffic as well as operating system events on a host can be monitored for anomaly detection. Network and host systems have different characteristics that are targeted to detecting different types of attacks. This dissertation presents several methods of improving the accuracy of host-based anomaly detection systems. Next, we summarize our findings from the various investigations.

7.1 Results and Contributions

Most of the traditional supervised learning based IDSs require a clean training data set which is difficult to obtain. Training data comprised of attacks results in an erroneous model for supervised anomaly detection, and all subsequent occurrence of the attack would be missed. It is thus important to purge all anomalies from the training data. In Chapter 3, we presented a motif-based representation for system call sequences based upon their spatial positions

within the sequence. Our system, called MORPHEUS, also created a single representation for all sequences called sequence space - using a distance metric between the motif-based representations. An unsupervised local outlier algorithm was used to purge this data void of all attacks and other anomalies. This offline procedure created a clean training data set. We also presented heuristics to automate parameter selection for the outlier detection algorithm. Our automated technique successfully detected local outliers, but failed to capture attack-based clusters. We also demonstrated the efficacy of data cleaning in conjunction with two anomaly detectors, namely t-stide and LERAD. With our technique, the number of detections increased by 100-300% for data sets used in our experiments.

Merging argument and sequence information creates a richer model for anomaly detection. This relates to the issue of feature extraction and utilization for better behavioral modeling in a host-based system. In Chapter 4, we portrayed the efficacy of incorporating system call argument information and used a rule-learning algorithm (LERAD) to model a host-based anomaly detection system. We proposed four variants: system calls within a fixed size window (S-LERAD), arguments for the current system call (A-LERAD), system calls within a fixed size sliding window and arguments for the current system call (M-LERAD), and all system calls and correpsonding arguments within a fixed window (M*-LERAD). Experiments on well known data sets indicated that our argument-based model, A-LERAD, detected more attacks than all the sequence-based techniques. Our sequence-based variant (S-LERAD) was also able to generalize better than the prevalent sequence based techniques, which rely on pure memorization. We also analyzed the alarms and attributed attack detections to partial attack signature, behavioral patterns of the intruder, or some peculiarity in the program execution.

LERAD is a supervised algorithm for anomaly detection that uses rule pruning to avoid overfitting. But pruning of learned model may adversely affect system accuracy. In Chapter 5, we argued that pruning may result in loss of coverage, thereby reducing the detection rate.

We presented three techniques to increase coverage - rule weighting, rule replacement and hybrid approach. We listed predictiveness and belief as two key aspects of rule quality. We proposed rule weighting wherein rules pruned earlier were retained. Additionally, rules with higher belief had higher weight. Rule replacement, on the other hand, replaced pruned rules with other candidates for improved coverage. The hybrid approach chose between weighting and replacement, depending on which technique had higher coverage. Generally, the hybrid approach detected most attacks at low false alarm rates. We studied the effect of coverage on accuracy and found that, for a given data set, higher coverage did correspond to higher accuracy in most cases.

In addition to the security threats for static hosts like desktop computers, there are key security issues targeted only towards mobile hosts. These include misuse of lost or stolen mobile device and MAC spoofing in wireless LANs. We motivate an anomaly detector for such issues in Chapter 6, where we modeled spatio temporal contextualities for mobile hosts. The underlying assumption was that the user would generally be at the same place around the same time. Any abnormalities could be due to misuse. We evaluated our technique on real data sets for mobile phone usage as well as 802.11 wireless local area networks. Our techniques had AUC $\geq$ 90% for over 40% wireless LAN users and 24% mobile phone users. Context clustering produced concise models, with 15-25% reduction in model size. We also stressed on the importance of adaptive modeling to accommodate concept drift and curtail false alarms further.

7.2 Future Work

The data cleaning procedure can be integrated with a hybrid of signature and anomaly based systems for better accuracy and the ability to detect novel attacks. A semi-supervised approach could be used with our representations to achieve the same. Our system can also be

used for user profiling and detecting masquerade. In terms of efficiency, the only bottleneck in our system is the motif extraction phase where cross-match is performed pair-wise. Speed-up is possible by using other techniques like suffix trees.

We used system call attributes for creation of richer models for anomaly detection. But our argument and sequence based representations assume fixed size tuples. A possible extension to variable length attribute window for more accurate modeling. More sophisticated features could also be devised from the argument information.

For rule weighting, the rule set size can be limited by eliminating a rule which has been violated many times and its weight falls below a user-defined threshold. We are also exploring other linear weight update functions. An alternate approach for learning is to minimize the rule set after pruning the violated rules. This might reduce the training time, but we suspect that it will also eliminate high coverage (more general) rules, resulting in a larger rule set comprising more specific rules, thereby increasing the test time. We intend to evaluate and compare the accuracy of such a learner.

In the case of spatio-temporal modeling for mobile hosts, coarser contexts could be defined such as beginning/mid/end of month, week and day. Incorporating such features might be sufficient for anomaly detection, especially for low variance users. STAD can also be used in conjunction with collision detection techniques (Fawcett and Provost 1997), and multi-modal biometric systems. In addition to detecting misuse for mobile hosts, the technique can also be applied to other domains such as credit card fraud and tracking employee access in government and other high security buildings.

Bibliography

Adam, N.; Janeja, V.; and Atluri, V. 2004. Neighbor- hood based detection of anomalies in high dimensional spatio-temporal sensor datasets. In *ACM SAC*.

Aggarwal, C., and Yu, P. 2001. Outlier detection for high dimensional data. In *SIGMOD*.

Agrawal, R., and Srikant, R. 1994. Fast algorithms for mining association rules. In *VLDB*.

Agrawal, R.; Imielinski, T.; and Swami, A. 1993. Mining association rules between sets of items in large databases. In *ACM SIGMOD*, 207–216.

Aha, D.; Kibler, D.; and Albert, M. 1991. Instance-based learning algorithms. *Machine Learning*.

Ailisto, H.; Alahuhta, P.; Haataja, V.; Kyllonen, V.; and Lindholm, M. 2002. Structuring context aware applications:five-layer model and example case. In *Concepts and Models for Ubiquitous Computing Workshop at UbiComp*.

Apap, F.; Honig, A.; Hershkop, S.; Eskin, E.; and Stolfo, S. 2002. Detecting malicious software by monitoring anomlaous windows registry accesses. In *Recent Advances in Intrusion Detection*.

Bahl, P., and Padmanabhan, V. 2000. Radar: An in-building rf-based user location and tracking system. In *IEEE Infocom*.

Bernaschi, M.; Gabrielli, E.; and Mancini, L. 2001. Operating system enhancement to prevent the misuse of system calls. In *Computer and Communications Security*.

Blum, A. 1997. Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain. *Machine Learning* 26(5):5–23.

Breunig, M.; Kriegel, H.; Ng, R.; and Sander, J. 2000. Lof: Identifying density-based local outliers. In *SIGMOD*, 93–104.

Chan, P.; Mahoney, M.; and Arshad, M. 2003. Learning rules and clusters for anomaly detection in network traffic. In Kumar, V.; Srivastava, J.; and Lazarevic, A., eds., *Managing Cyber Threats: Issues, Approaches and Challenges*. Kluwer.

Chan, P. 1999. A non-invasive learning approach to building web user profiles. In *KDD Workshop on Web Usage Analysis and User Profiling*.

Chen, G., and Kotz, D. 2000. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College.

Clark, P., and Niblett, T. 1989. The CN2 induction algorithm. *Machine Learning* 3:261–285.

Clarkson, B., and Pentland, A. 1998. Extracting context from environmenal audio. In *Second International Symposium on Wearable Computers*.

Cohen, W., and Singer, Y. 1999. A simple, fast, and effective rule learner. In *AAAI*.

Cohen, W. 1995. Fast effective rule induction. In *International Conference on Machine Learning*, 115–123.

Coull, S.; Branch, J.; Szymanski, B.; and Breimer, E. 2003. Intrusion detection: A bioinformatics approach. In *Annual Computer Security Applications Conference*.

Cover, T., and Hart, P. 1967. Nearest neighbour pattern classification. *IEEE Transactions on Information Theory* 13(1):21–27.

Dedo, D. 2004. Windows mobile-based devices and security: Protecting sensitive business information. Technical report, Microsoft Corporation.

Denning, D. 1987. An intrusion detection model. *IEEE Transactions on Software Engineering* 13(2):222–232.

Dey, A.; Salber, D.; Abowd, G.; and Futakawa, M. 1999. The conference assistant: Combining context-awareness with wearable computing. In *3rd International Symposium on Wearable Computers*, 21–28.

Domingos, P., and Pazzani, M. 1997. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning* 29:103–130.

Eagle, N., and Pentland, A. 2005. Social serendipity: Mobilizing social software. *IEEE Pervasive Computing Special Issue: The Smart Phone* 28–34.

Eagle, N., and Pentland, A. 2006. Reality mining: Sensing complex social systems. *Personal and Ubiquitous Computing* 10(4):255–268.

Eskin, E.; Arnold, A.; Prerau, M.; Portnoy, L.; and S., S. 2002. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In Barbara, D., and Jajodia, S., eds., *Applications of Data Mining in Computer Security*. Kluwer.

Fawcett, T., and Provost, F. 1997. Adaptive fraud detection. *Data Mining and Knowledge Discovery* 1:291–316.

Feng, H.; Kolesnikov, O.; Fogla, P.; Lee, W.; and Gong, W. 2003. Anomaly detection using call stack information. In *IEEE Symposium on Security and Privacy*.

Flach, P. 2004. The many faces of roc analysis in machine learning. In *ICML Tutorial*.

Flanagan, J. A.; Mäntyjarvi, J.; and Himberg, J. 2002. Unsupervised clustering of symbol strings and context recognition. In *IEEE International Conference on Data Mining*.

Flickenger, R. 2003. *Wireless Hacks*. O'Reilly.

Forrest, S.; Hofmeyr, S.; Somayaji, A.; and T., L. 1996. A sense of self for unix processes. In *IEEE Symposium on Security and Privacy*.

Freund, Y., and Schapire, R. 1996. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, 148–156.

Furnkranz, J., and Widmer, G. 1994. Incremental reduced error pruning. In *International Conference on Machine Learning*.

Furnkranz, J. 1997. Pruning algorithms for rule learning. *Machine Learning* 27:139–171.

Ghosh, A., and Schwartzbard, A. 1999. A study in using neural networks for anomaly and misuse detection. In *USENIX Security Symposium*.

Gibbs, A., and McIntyre, G. 1970. The diagram, a method for comparing sequences. its use with amino acid and nucleotide sequences. *Eur. J. Biochem* 16.

Goode, A. 2006. Mobile data security - access, content, identity and threat management, 2006-2011. Technical report, Juniper Research.

Guha, S.; Rastogi, R.; and Shim, K. 1998. Cure: An efficient clutering algorithm for large databases. In *ACM SIGMOD*.

Guha, S.; Rastogi, R.; and Shim, K. 2000. Rock: A robust clustering algorithm for categorical attributes. *Information Systems* 25(5):345–366.

Himberg, J.; Korpiaho, K.; Mannila, H.; Tikanmäki, J.; and Toivonen, H. T. 2001. Time series segmentation for context recognition in mobile devices. In *IEEE International Conference on Data Mining*.

Jain, A., and Dubes, R. 1988. *Algorithms for Clustering Data*. Prentice Hall.

Jiang, N.; Hua, K.; and Sheu, S. 2002. Considering both intra-pattern and inter-pattern anomalies in intrusion detection. In *IEEE International Conference on Data Mining*.

Karypis, G.; Han, E.; and Kumar, V. 1999. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*.

Kendell, K. 1999. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, MIT, Cambridge, MA.

Keogh, E., and Pazzani, M. 2000. Scaling up dynamic time warping for data mining applications. In *ACM International Conference on Knowledge Discovery and Data Mining*.

Keogh, E., and Pazzani, M. 2001. Derivative dynamic time warping. In *SIAM International Conference on Data Mining*.

Knorr, E., and Ng, R. 1998. Algorithms for mining distance-based outliers in large data sets. In *VLDB*. Morgan Kaufmann.

Korpipaa, P.; Koskinen, M.; Peltola, J.; Makela, S.; and Seppanen, T. 2003. Bayesian approach to sensor-based context awareness. *Personal and Ubiquitous Computing* 7(2):113–124.

Kotler, J., and Maloof, M. 2003. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *IEEE International Conference on Data Mining*, 123–130.

Kotz, D.; Henderson, T.; and Abyzov, I. 2005. CRAWDAD trace set dartmouth/campus/movement (v. 2005-03-08). Downloaded from http://crawdad.cs.dartmouth.edu/dartmouth/campus/movement.

Kruegel, C.; Mutz, D.; Valeur, F.; and Vigna, G. 2003. On the detection of anomalous system call arguments. In *European Symposium on Research in Computer Security*.

Laasonen, K.; Raento, M.; and Toivonen, H. 2004. Adaptive on-device location recognition. In *Pervasive Computing: Second International Conference*, 287–304.

Laasonen, K. 2005. Clustering and prediction of mobile user routes from cellular data. In *PKDD*.

Lane, T., and Brodley, C. 1997a. Detecting the abnormal: Machine learning in computer security. Technical Report TR-ECE 97-1, Purdue University, Lafayette, IN.

Lane, T., and Brodley, C. 1997b. Sequence matching and learning in anomaly detection for computer security. In *AI Approaches to Fraud Detection and Risk Management*.

Lazarevic, A.; Ertoz, L.; Ozgur, A.; Srivastava, J.; and Kumar, V. 2003. A comparative study of anomaly detection schemes in network intrusion detection. In *SIAM International Conference on Data Mining*.

Li, X.; Han, J.; Kim, S.; and Gonzalez, H. 2007. Roam: Rule- & motif-based anomaly detection in massive moving object data sets. In *SIAM International Conference on Data Mining*.

Liao, Y., and Vemuri, R. 2002. Use of text categorization techniques for intrusion detection. In *USENIX Security Symposium*.

Lippmann, R.; Haines, J.; Fried, D.; Korba, J.; and Das, K. 2000. The 1999 darpa off-line intrusion detection evaluation. *Computer Networks*.

Littlestone, N., and Warmuth, M. 1994. The weighted majority algorithm. *Information and Computation* 108(2):212–261.

Littlestone, N. 1988. Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. *Machine Learning* 2:285–318.

Mahalanobis, P. 1930. On tests and measures of groups divergence. *International Journal of the Asiatic Society of Bengal* 26(541).

Mahoney, M., and Chan, P. 2002. Learning non-stationary models of normal network traffic for detecting novel attacks. In *ACM Knowledge Discovery and Data Mining*.

Mahoney, M., and Chan, P. 2003. Learning rules for anomaly detection of hostile network traffic. In *IEEE International Conference on Data Mining*.

Mazeroff, G.; De-Cerqueira, V.; Gregor, J.; and Thomason, M. 2003. Probabilistic trees and automata for application behavior modeling. In *ACM Southeast Regional Conference*.

Michalski, R. S.; Mozetic, I.; Hong, J.; and Lavrac, N. 1986. The aq15 inductive learning system: An overview and experiments. Technical Report UIUCDCS-R-86-1260, University of Illinois, Urbana.

Mitchell, T.; Caruana, R.; Freitag, D.; Mcdermott, J.; and Zabowski, D. 1994. Experience with a learning personal assistant. *Communications of the ACM* 37(7):81–91.

Moffat, A. 1988. A note on the ppm data compression algorithm. Technical Report 88/7, University of Melbourne, Victoria, Australia.

Netstumbler. 2004. www.stumbler.net.

Paxson, V., and Floyd, S. 1995. The failure of poisson modeling. *IEEE/ACM Trans. Networking* 3:226–244.

Peikair, C., and Fogie, S. 2003. *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*. Addison-Wesley.

Portnoy, L. 2000. Intrusion detection with unlabeled data using clustering. Technical report, Columbia University, NY.

Quinlan, J. R. 1993. *C4.5: programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Raento, M.; Oulasvirta, A.; Petit, R.; and Toivonen, H. 2005. Contextphone - a prototyping platform for context-aware mobile applications. *IEEE Pervasive Computing* 4(2):51–59.

Ramaswamy, S.; Rastogi, R.; and Kyuseok, S. 2000. Efficient algorithms for mining outliers from large data sets. In *ACM SIGMOD*.

Rigoutsos, I., and Floratos, A. 1998. Combinatorial pattern discovery in biological sequences. In *Bioinformatics*.

Salvador, S., and Chan, P. 2004. Learning states and rules for time series anomaly detection. In *FLAIRS*. AAAI Press.

Schapire, R. 1990. The strength of weak learnability. *Machine Learning* 5:197–226.

Sekar, R.; Bendre, M.; Dhurjati, D.; and Bollineni, P. 2001. A fast automaton-based method for detecting anomalous program behaviors. In *Symposium on Security and Privacy*.

Shavlik, J., and Shavlik, M. 2004. Selection, combination, and evaluation of effective software sensors for detecting abnormal computer usage. In *ACM Knowledge Discovery and Data Mining*.

Smyth, P., and Goodman, R. 1991. Rule induction using information theory. *Knowledge discovery in databases* 159–176.

Smyth, P., and Goodman, R. 1992. An information theoretic approach to rule induction from databases. *IEEE Trans. Know. Data Engineering* 4(4):301–316.

Stolfo, S.; Apap, F.; Eskin, E.; Heller, K.; Hershkop, S.; Honig, A.; and Svore, K. 2004. Detecting malicious software by monitoring anomalous windows registry accesses. Technical report, Columbia University.

Tan, K., and Maxion, R. 2002. Why 6? defining the operational limits of stide. In *IEEE Symposium on Security and Privacy*, 188–201. IEEE.

Tandon, G., and Chan, P. 2003. Learning rules from system call arguments and sequences for anomaly detection. In *IEEE ICDM Workshop on Data Mining for Computer Security*, 20–29.

Tandon, G., and Chan, P. 2005. Learning useful system call attributes for anomaly detection. In *18th International Florida Artificial Intelligence Research Society (FLAIRS) Conference*, 405–410.

Tandon, G., and Chan, P. 2006. On the learning of system call attributes for host-based anomaly detection. *International Journal on Artficial Intelligence Tools* 15(6):875–892.

Tandon, G., and Chan, P. 2007. Weighting versus pruning in rule validation for detecting network and host anomalies. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 697–706.

Tandon, G.; Chan, P.; and Mitra, D. 2004. Morpheus: Motif oriented representations to purge hostile events from unlabeled sequences. In *ACM CCS Workshop on Visualization and Data Mining for Computer Security*, 16–25.

Tandon, G.; Chan, P.; and Mitra, D. 2006. Data cleaning and enriched representations for anomaly detection in system calls. In Maloof, M., ed., *Machine Learning and Data Mining for Computer Security - Methods and Applications*. Springer. 137–156.

Tandon, G.; Mitra, D.; and Chan, P. 2004. Motif-oriented representation of sequences for a host-based intrusion detection system. In *International Conference on Industrial and Engineering Applications of Artifical Intelligence and Expert Systems*, 605–615.

Van Laerhoven, K., and Cakmakci, O. 2000. What shall we teach our pants? In *Fourth International Symposium on Wearable Computers*, 77–83.

Wagner, D., and Soto, P. 2002. Mimicry attacks on host-based intrusion detection systems. In *Computer and Communications Security*.

Wang, Y., and Wong, A. 2003. From association to classification: inference using weight of evidence. *IEEE Trans. Knowledge and Data Engineering* 15(3).

Want, R.; Hopper, A.; Falcao, V.; and Gibbons, J. 1992. The active badge location system. *ACM Transactions on Information Systems* 10(1):91–102.

Warrender, C.; Forrest, S.; and Pearlmutter, B. 1999. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*.

Welch, C., and Lathrop, S. 2003. A survey of 802.11a wireless security threats and security mechanisms. Technical report, United States Military Academy.

Wespi, A.; Dacier, M.; and Debar, H. 1999. An intrusion-detection system based on the teiresias pattern-discovery algorithm. In *European Institute for Computer Anti Virus Research Conference*.

Wespi, A.; Dacier, M.; and Debar, H. 2000. Intrusion detection using variable-length audit trail patterns. In *Recent Advances in Intrusion Detection*, Lecture Notes in Computer Science. Springer-Verlag.

Whittaker, J., and De-Vivanco, A. 2002. Neutralizing windows-based malicious mobile code. In *ACM SAC*, 242–246. ACM.

Witten, I., and Bell, T. 1991. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory* 37(4):1085–1094.

Yin, J.; Chai, X.; and Yang, Q. 2004. High level goal recognition in wireless lan. In *AAAI*.

Youssef, M.; Agrawala, A.; and Shankar, A. 2003. Wlan location determination via clustering and probability distributions. In *IEEE PerCom*.