**AFRL-IF-RS-TR-2004-84**
**Final Technical Report**
**March 2004**


# A DATA MINING APPROACH FOR BUILDING COST-SENSITIVE AND LIGHT INTRUSION DETECTION MODELS

**North Carolina State University**

**Sponsored by**
**Defense Advanced Research Projects Agency**
**DARPA Order No. K357**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*



**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**
**ROME RESEARCH SITE**
**ROME, NEW YORK**

**STINFO FINAL REPORT**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-84 has been reviewed and is approved for publication.

APPROVED: /s/

THOMAS M. BLAKE
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 074-0188*



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | MARCH 2004 | Final Aug 00 – Aug 03 |

**4. TITLE AND SUBTITLE**
A DATA MINING APPROACH FOR BUILDING COST-SENSITIVE AND LIGHT INTRUSION DETECTION MODELS

**5. FUNDING NUMBERS**
C - F30602-00-1-0603
PE - 62301E
PR - K357
TA - 33
WU - B1

**6. AUTHOR(S)**
Wenke Lee,
Salvatore J. Stolfo,
and Philip K. Chan

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
North Carolina State University
College of Engineering
Campus Box 8207
Raleigh North Carolina 27695-8207

**8. PERFORMING ORGANIZATION REPORT NUMBER**
N/A

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFGB
525 Brooks Road
Rome New York 13441-4505

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
AFRL-IF-RS-TR-2004-84

**11. SUPPLEMENTARY NOTES**
AFRL Project Engineer: Thomas M. Blake/IFGB/(315) 330-1482/ Thomas.Blake@rl.af.mil

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT *(Maximum 200 Words)***
The report provides a summary of the intrusion detection research completed for this effort. The research studied how to build cost-sensitive and light weight intrusion detection models. The main technical components of the research are: 1) Automatic feature construction by analyzing the patterns of normal and intrusion activities computed from large amounts of audit data, 2) Using cost-sensitive machine learning algorithms to construct intrusion detection models that achieve optimal performance on the given (often site-specific) cost metrics, cluster attack signatures and normal profiles and accordingly construct one light model of each cluster to maximize the utility of each model, and 3) Dynamic (re-) configuration of the light models to make an IDS effective and efficient, and resilient to IDS-related attacks. Algorithms and prototype systems were developed and extensive experiment using DARPA datasets and other real-world datasets were conducted. The results showed that the technologies developed in this project are more advanced and better than today's state-of-the-art.

**14. SUBJECT TERMS**
Intrusion Detection System, IDS, Data Mining, Anomaly Detection Algorithms, Alert Correlation, Light-Weight Anomaly Detection, Cost Sensitive Modeling, Adaptive IDS, Alert Correlation, Attack Scenario Analysis, IDS Performance Metrics

**15. NUMBER OF PAGES**
31

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

# Table of Contents

## 1. Summary

Intrusion detection systems (IDSs) are currently developed using pure knowledge-engineering approaches where expert knowledge on network, operating systems, and attack methods are encoded as detection models. The IDSs are not very effective in detecting variations of known attacks and novel attacks because expert knowledge is often incomplete and tends to be too specific to attack instances. Since the manual development process is very slow and expensive, IDSs are often equipped with only one centralized detection module, making them unable to keep up with fast (automated) attacks, and worse, subject to denial-of-service attacks. IDSs are not cost-sensitive because the cost factors, which include the development and operational costs, and the intrusion costs (damages), etc., are simply ignored as unwanted complexities in the IDS life cycle.

The research proposed aims to develop methodologies and tools for building cost-sensitive and light intrusion detection models. The main technical components of the research are:

- Automatically constructing features and anomaly detection models by analyzing the patterns of normal and intrusion activities computed from large amount of audit data.
- Using cost-sensitive machine learning algorithms to construct intrusion detection models that achieve optimal performance on the given (often site-specific) cost metrics, cluster attack signatures and normal profiles and accordingly construct one light model for each cluster to maximize the utility of each model.
- Dynamic (re-)configuration of the light models to make an IDS effective and efficient, and resilient to IDS-related attacks.

We have successfully accomplished the goals of the project. We developed several novel feature construction and anomaly detection algorithms. In particular, we invented very light-weight anomaly detection algorithms that analyze the frequent values of packet header fields or protocol commands in packet payloads and detect deviations (anomalies). Results on DARPA IDS Evaluation data and real-world data showed that these algorithms can effectively detect new attacks.

We studied the problem of cost-sensitive modeling in intrusion detection. We examined the cost factors in intrusion detection, namely, damage cost, response cost, and operation cost. We showed how the performance of an IDS, i.e., a true positive, false positive, true negative, and false negative, affects the total cost incurred. For example, responding to an intrusion with higher response cost than damage cost will cost more than not responding to the intrusion. We developed strategies for an IDS to decide whether (and when) to "ignore" some intrusions in order to minimize cost.

We studied how to dynamically re-configure a real-time IDS to provide the optimal protection, and developed a control and optimization approach that decides the optimal IDS configuration based on resource constraints and traffic and attack conditions. This problem is modeled as a Knapsack problem. Essentially, an IDS has limited real-time

resources and may not be able to process all packets if the traffic rate and volume is too high. The solution is for the IDS enable only the most "valuable" set of tasks so that the corresponding traffic data can be analyzed within the resource constraints. We call such an IDS an Adaptive IDS. We have modified the open-source Snort and Bro to make them adaptive. Experiments showed that these IDSs can automatically change its configurations according to traffic and attack conditions to provide the best values.

In addition to the research tasks outlined in the original proposal, we have also studied the problem of alert correlation (a topic not included in the original proposal). Instead of a pattern-matching approach that can only recognized known attack step relationships, we aim to develop algorithms for detecting new attack step relationships. We developed a statistical causality analysis approach, based on GCT (Granger Causality Test), which with very little prior domain knowledge can find out a pair of alerts of the likely related attack steps. The intuition of this approach is that related attack steps may result in co-occurrences of their alerts in the alert data streams. Therefore, statistical tools such GCT can be applied to find such occurrences. Experiments using DARPA's Grand Challenge Problem (GCP) dataset showed that this approach can indeed find novel attack step relationships that other approaches based patter-matching can't.

The results of this research have been reported in many publications in top conferences and journals. In addition, we have actively engaged in technology transfer throughout the course of the project. In particular, the PIs were involved in the founding of System Detection Inc. The company has been developing commercial products based on findings of this project and the previous DARPA-funded JAM project.

## 2. Introduction

Intrusion detection is the process of identifying and responding to malicious actions that aim to compromise the security of a system, i.e., its confidentiality, integrity, and availability. The basic premises of intrusion detection are: system activities are observable, e.g., via auditing; and normal and intrusion activities leave distinct evidence. Therefore, an ID model has two basic elements: the features, that is, the indicators (evidence), measured using the audit data; and the modeling algorithms that piece together and reason about the indicators. The two main intrusion detection techniques include misuse detection, which uses signatures of specific attacks or system vulnerabilities to pattern-match and detect intrusions; and anomaly detection, which uses established normal profiles of users or system resources to detect significant deviation as probable intrusion. Misuse detection can be very efficient and accurate, however, by definition, it can detect only the instances of known intrusions. Anomaly detection is the only weapon to detect new attacks, however, it often cannot determine the nature of an attack and can have a high false alarm rate. An IDS therefore needs to carefully combine both misuse and anomaly detection models.

Despite the research and commercial efforts in the past two decades, there are still a large gap between the capabilities of IDSs and that of cyber attackers. Results from the 1998 DARPA Intrusion Detection Evaluation showed that although several intrusion detection programs already showed good detection rates on known intrusions and their slight

variations, none of the systems showed acceptable detection rate on "novel" attacks, i.e., those that are not modeled in the detection systems. Most IDSs are designed only to achieve optimal effectiveness (i.e., accuracy). However, for IDSs to be widely deployed, they need to bring economic benefits to organizations. This requires that IDSs balance the requirements of both accuracy and costs, which include development costs, operational costs, damage (intrusion) costs, etc. Real-time IDSs need to avoid becoming a single point of failure because cyber attackers are beginning to devise attacks that aim to elude IDSs through evasion tactics or denial-of-service. Multiple light, fast, and cooperative detection systems are likely to achieve more robust performance than using a monolithic system. Most IDSs only output alarms on individual steps. When IDSs are deployed in a large network, the sheer amount of IDS alerts can overwhelm the security staff and prevent proper and timely response actions. Therefore, we need to develop techniques to reduce the amount of alerts, correlate the alerts and recognize complex attack scenarios that are composed of a number of attack steps.

The traditional manual approaches of encoding expert knowledge cannot meet the challenges of building IDSs that are equipped with the advanced capabilities discussed above. To effectively detect novel attacks, an IDS needs to provide comprehensive and systematic coverage, i.e., modeling, of all network elements and their interactions. Expert knowledge is simply too limited compared with the complexities of a network system. The delicate balance between accuracy and various cost factors, and the need to construct multiple cooperative models also add significant complexities in the development process. In alert analysis and attack scenario analysis the key is to identify the attack steps that are related. There are potentially many possible attack scenarios. Thus it is impossible to know *a priori* what attack step relationships are indicative of attack steps in a scenario.

We therefore need a new development paradigm. We proposed to build and demonstrate a novel system for rapid development and deployment of effective and cost-sensitive IDSs. The key motivation of our research is to automate as much as possible the analysis tasks in intrusion detection. We consider intrusion detection as a classification problem, that is, we wish to classify each audit record into one of a discrete set of possible categories, normal or a particular kind of intrusion. We can thus apply machine learning approaches to inductively learn classifiers as detection models. Given a set of records, where one of the features is the class label (i.e., the concept), classification algorithms can compute a model that uses the most discriminating feature values to describe each concept. However, before we can apply classification algorithms, we need to first select and construct the right set of system features that may contain evidence (indicators) of normal or intrusions. We developed an automatic feature selection and construction system to systematically discover and construct predictive features that can be used to build effective misuse and anomaly detection models. We developed cost-sensitive classification algorithms to construct ID models that are optimized to provide the best economic benefits (cost-saving). We also studied how to efficiently execute ID models in real-time. In alert analysis, we developed algorithms to recognize new attack step relationships.

## 3. Methods, Assumptions, and Procedures

As academic/university researchers, we aimed to make fundamental contributions, rather than to develop product prototypes. We therefore focused on theoretical studies, algorithm developments, and research prototype implementations. As discussed above, a key motivation is to automate the analysis tasks in intrusion detection. Toward this end, we applied machine learning, data mining, statistical analysis, and control and optimization techniques. For example, in alert analysis and attack scenario analysis, rather than using the approach of pattern-matching of known attack step relationships, which is straightforward but of limited use against new attack scenarios, we developed a statistical causality analysis approach that while still preliminary shows great potentials in its abilities to recognize attack step relationships.

We communicated closely with the program manager and interacted with other research groups in the community to get feedbacks on our research directions and progress. In particular, we utilized the scenarios and data sets provided by DARPA and other project teams as training or validation data sets for our algorithms.

The main goal of our project was to build a development system so that effective, cost-sensitive and light ID models can be quickly built and deployed. We also developed real-time IDSs equipped with our ID models to demonstrate the advanced capabilities of our development system. Our project proceeded as follows.

For the first year, we concentrated on algorithm development. This included: enhancing and integrating existing components of JAM, e.g., the data mining programs for audit data analysis, and the pattern encoding and analysis programs; and developing initial versions of the new algorithmic components. We established the capabilities of automated feature construction. We studied the cost factors in intrusion detection and developed a model to evaluate an IDS based on cost. We also developed a light-weight anomaly detection algorithm that analyzes the frequent values of packet header fields. Experiments using DARPA dataset showed that this algorithm can detect many new attacks.

For the second year, we developed an approach for learning an anomaly detection model over noisy (unclean) data. We studied the problem of dynamically changing the configuration of an IDS to provide optimal value according to the run-time resource constraints and attack conditions. We considered it as a control and optimization problem and developed a solution based on the Knapsack algorithm. We also started to investigate the problem of alert correlation and attack scenario analysis.

For the third year, we developed a new and light-weight anomaly detection algorithm that analyzes the frequent values of protocol commands in packet payloads. Experiments using DARPA dataset showed that this algorithm can detect new attacks. We modified two open-source IDSs, Bro and Snort, to make them adaptive using our Knapsack based approach. Experiments showed that these IDSs can dynamically change their configurations to provide the best detection capabilities according to run-time conditions. We also developed a statistical causality analysis algorithm, based on the Granger

Causality Test (GCT), for identifying new attack step relationships. Experiments using the DARPA Grand Challenge Problem (GCP) dataset showed that it can detect new and stealth attack step relationships that other pattern-matching based approaches can't.

Throughout the whole project duration, we attended regular PI meetings and visit other groups to exchange ideas and share technologies. We published our findings in yearly major conferences so that our work can be reviewed critically from the scientific communities. We also actively participated in technology transfer. In particular, the PIs were part of the founding team of System Detection Inc., which is developing commercial products based on technologies developed in this project and previous DARPA-funded JAM project.

## 4. Results and Discussion

We now summarize the main results of the project.

### 4.1 Feature Construction

Two basic premises of intrusion detection are that system activities are observable, e.g., via auditing, and there is distinct evidence that can distinguish normal and intrusive activities. We call the evidence extracted from raw audit data *features*, and use these features for building and evaluating intrusion detection models. Feature extraction (or construction) is the processes of determining what evidence that can be taken from raw audit data is most useful for analysis. Feature extraction is thus a critical step in building an IDS. That is, having a set of features whose values in normal audit records differ significantly from the values in intrusion records is essential for having good detection performance.

We have developed a set of data mining algorithms for selecting and constructing features from audit data [1]. First, raw (binary) audit data is processed and summarized into discrete records containing a number of basic features, e.g., timestamp, duration, source and destination IP addresses and ports, and error condition flags. Specialized data mining programs [2] are then applied to connection records to compute frequent patterns describing correlations among features and frequently co-occurring events across many connection records. The consistent patterns of normal activities and the "unique" patterns associated with an intrusion are then identified and analyzed to construct additional features for connection records. It can be shown that the constructed features can indeed clearly separate intrusion records from normal ones. Using this approach, the constructed features are more grounded on empirical data, and thus more objective than expert knowledge. Results from the 1998 DARPA Intrusion Detection Evaluation [3] showed that the ID model constructed using our algorithms was one of the best performing of all the participating systems.

As an example, let us consider the SYN-Flood attack. When launching this attack, an attacker uses many spoofed source addresses to open many connections which never become completely established (i.e., only the first SYN packet is sent, and the connection remains in the "S0" state) to some port on a victim host (e.g., *http*). When comparing the

patterns from the 1998 DARPA dataset that contain SYN-Flood attacks with the patterns from a "baseline" normal dataset (of the same network), by first encoding the patterns into numbers and then computing "difference" scores. The following pattern, a frequent episode [4], has the highest "intrusion-only" (i.e., unique for the intrusion): "93% of the time, after two *http* connections with S0 flag are made to host *victim*, within 2 seconds from the first of these two, the third similar connection is made; and this pattern occurs in 3% of the data". Accordingly, our feature construction algorithm parses the pattern and uses the *anatomy* (or structural) information about an intrusion, e.g., "the same *service* (i.e., *port*) is targeted", and the *invariant* information, e.g. flag=S0, to construct the following features: "a count of connections to the same dst_host in the past 2 seconds", and among these connections, "the percentage of those that have the same service, and the percentage of those that have the S0 flag." For the two "percentage" features, the normal connection records have values close to 0, but the connection records belong to SYN-Flood have value above 80%. Once these discriminative features are constructed, it is easy to generate the detection rules via either manual (i.e. hand-coding) or automated (i.e., machine learning) techniques. For example, we use RIPPER [5], an inductive rule learner, to compute a detection rule for *syn-flood*: **if** for the past 2 seconds, the *count of connections to the same dst host* is greater than 4; **and** the *the percentage of those that have the same service* is greater than 75%; **and** *the percentage of those that have the "S0" flag* is greater than 75%, **then** there is a *syn_flood* attack.

We have implemented a system that fully automated the features and model construction process. The inputs are two sets of connection records, one for normal connections and the other contains an attack. The connection records contain the basic features. The system than computes frequent patterns from both sets of connection records, compare the patterns to identify the to 10% intrusion-only patterns, parses the patterns to construct features, and invokes RIPPER to learn rules to detect the intrusion. The learned rules are tested on a given test dataset. If the accuracy is below a pre-defined threshold, the above process is iterated, with different heuristics in pattern computation, until a set of sufficiently accurate rules are computed or a pre-defined limit (e.g., on the number of iterations) is met.

### 4.2 Unsupervised Learning

Traditional model building algorithms typically require a large amount of labeled data in order to create effective detection models. One major difficulty in deploying a data mining-based IDS is the need for labeling system audit data for use by these algorithms. For misuse detection systems, the data needs to be accurately labeled as either normal or attack. For anomaly detection system, the data must be verified to ensure it is completely normal, which requires the same effort. Since models (and data) are specific to the environment on which the training data was gathered, this cost of labeling the data must be incurred for each deployment of the system. Ideally, we would like to build detection models from collected data without needing to manually label it. In this case, the deployment cost would greatly be decreased because the data would not need to be labeled. In order to build these detection models, we need a new class of model building

algorithms. These model building algorithms can take as input unlabeled data and create a detection model. We call these algorithms *unsupervised* anomaly detection algorithms.

We developed an overview of two unsupervised anomaly detection algorithms that have been applied to intrusion detection. These algorithms can also be referred to as anomaly detection over noisy data. The reason the algorithm must be able to handle noise in the data is that we do not want to manually verify that the audit data collected is absolutely clean (i.e., contains no intrusions). Unsupervised anomaly detection algorithms are motivated by two major assumptions about the data which are reasonable for intrusion detection. The first assumption is that anomalies are very rare. This corresponds to the fact that normal use of the system greatly outnumbers the occurrence of intrusions. This means that the attacks compose a relatively small proportion of the total data. The second assumption is that the anomalies are quantitatively different from the normal elements. In intrusion detection this corresponds to the fact that attacks are drastically different from normal usage.

Since anomalies are very rare and quantitatively different from the normal data, they stand out as outliers in the data set. Thus, we can cast the problem of detecting the attacks into an outlier detection problem. Outlier detection is the focus of much literature in the field of statistics [6]. In intrusion detection, intuitively, if the ratio of attacks to normal data is small enough, then because the attacks are different, the attacks stand out against the background of normal data. We can thus detect the attack within the dataset.
We have performed experiments with two types of unsupervised anomaly detection algorithms, each for a different type of data. We applied a probabilistic based unsupervised anomaly detection algorithm to building detection models over system calls and a clustering based unsupervised anomaly detection algorithm to network traffic. The probabilistic algorithm approached detecting outliers by estimating the likelihood of each element in the data. We partition the data into two sets, normal elements and anomalous elements. Using a probability modeling algorithm over the data, we compute the most likely partition of the data. Details and experimental results of the algorithm applied to system call data are given in [7]. The clustering approach detects outliers by clustering the data. The intuition is that the normal data will cluster together because there is a lot of it. Because anomalous data and normal data are very different from each other, they do not cluster together. Since there is very little anomalous data relative to the normal data, after clustering, the anomalous data will be in the small clusters. The algorithm first clusters the data and then labels the smallest clusters as anomalies.

Details and experimental results applied to network data are given in [8].

### 4.3 Light-Weight Anomaly Detection

Most network anomaly systems such as ADAM [9], NIDES [10], and SPADE [11] monitor IP addresses, ports, and TCP state. This catches user misbehavior, such as attempting to access a password protected service (because the source address is unusual) or probing a nonexistent service (because the destination address and port are unusual). However, this misses attacks on public servers or the TCP/IP stack that might otherwise

be detected because of anomalies in other parts of the protocol. Often these anomalies occur because of software errors in the attacking or victim program, because of anomalous output after a successful attack, or because of misguided attempts to elude the IDS. Our anomaly detection algorithms have two nonstationary components developed and tested on the 1999 DARPA IDS evaluation test set [12], which simulates a local network under attack. The first component is a packet header anomaly detector (PHAD) which monitors the entire data link, network, and transport layer, without any preconceptions about which fields might be useful. The second component is an application layer anomaly detector (ALAD) which combines a traditional user model based on TCP connections with a model of text-based protocols such as HTTP, FTP, and SMTP. Both systems learn which attributes are useful for anomaly detection, and then use a nonstationary model, in which events receive higher scores if no novel values have been seen for a long time.

### 4.3.1 LEARNING NONSTATIONARY MODELS

The goal of intrusion detection is, for any given event *x*, to assign odds that *x* is hostile, e.g., odds (x_is_hostile) = P(attack|x) / P(no_attack|x)

By Bayes law, we can write:

P(attack|x) = P(x|attack)P(attack) / P(x)
P(no_attack|x) = P(x|no_attack)P(no_attack) / P(x)

By dividing these equations, and letting odds(attack) = P(attack) / P(no_attack), we have:

odds(x_is_hostile) = odds(attack)P(x|attack) / P(x | no_attack)

We have factored the intrusion detection problem into three terms: odds(attack), the background rate of attacks; P(x|attack), a signature detection model, and 1 / P(x|no_attack), an anomaly detection model. In this paper, we address only the anomaly detection component, 1 / P(x|no_attack). Thus, we model attack- free data, and assign (like SPADE) anomaly scores inversely proportional to the probability of an event based on this training. Anomaly detection models like ADAM, NIDES, and SPADE are stationary, in that P(x) depends on the average rate of x in training and is independent of time. For example, the probability of observing some particular IP address is estimated by counting the number of observations in training and dividing by the total number of observations. However, this may be incorrect. Paxson and Floyd [13] showed that many types of network processes, such as the rate of a particular type of packet, have self-similar or fractal behavior. This is a nonstationary model, one in which no sample, no matter how short or long can predict the rate of events for any other sample. Instead, they found that events tend to occur in bursts separated by long gaps on all time scales, from milliseconds to months. We believe this behavior is due to changes of state in the system, such as programs being started, users logging in, software and hardware upgrades, and so on. We can adapt to state changes by exponentially decaying the training counts to favor recent events, and many models do just that. One problem with this approach is that we

have to choose either a decay rate (half life) or a maximum count in an *ad-hoc* manner. We avoid this problem by taking training decay to the extreme, and discarding all events (an attribute having some particular value) before the most recent occurrence. In our model, the best predictor of an event is the time since it last occurred. If an event x last occurred $t$ seconds ago, then the probability that x will occur again within one second is $1/t$. We do not care about any events prior to the most recent occurrence of $x$.
In an anomaly detection system, we are most interested in those events that have the lowest probability. As a simplification, we assign anomaly scores only to those events that have *never* occurred in training, because these are certainly the least likely. We use the PPMC model of novel events, which is also used in data compression [14]. This model states that if an experiment is performed $n$ times and $r$ different outcomes are observed, then the probability that the next outcome will not be one of these $r$ values is approximately $r/n$. Stated another way, the fraction of events that were novel in training is $r/n$, and we expect that rate to continue. This probably overestimates the probability that the next outcome will be novel, since most of the novel events probably occurred early during training. Nevertheless, we use it. Because we have separate training data (without attacks) and test data (with attacks), we cannot simply assign an anomaly score of $1/P(x) = n/r$. If we did, then a subsequent occurrence of $x$ would receive the same score, even though we know (by our nonstationary argument) that a second occurrence is very likely now. We also cannot add it to our model, because the data is no longer attack-free. Instead, we record the time of the event, and assign subsequent occurrences a score of $t/P(x) = tn/r$, where $t$ is the time since the previous anomaly. On the first occurrence of $x$, $t$ is the time since the last novel observation in training. An IDS monitors a large number of attributes of a message, each of which can have many possible outcomes. For each attribute with a value never observed in training, an anomaly score of $tn/r$ is computed, and the sum of these is then assigned to the message. If this sum exceeds a threshold, then an alarm is signaled.

$$\text{anomaly score} = \sum_i t_i \, n_i / r_i, \text{ where attribute } i \text{ is novel in training}$$

We next describe two models, PHAD and ALAD. In PHAD (packet header anomaly detection), the message is a single network packet, and the attributes are the fields of the packet header. In ALAD (application layer anomaly detection), the message is an incoming server TCP connection. The attributes are the application protocol keywords, opening and closing TCP flags, source address, and destination address and port number.

#### 4.3.1.1 Packet Header Anomaly Detection (PHAD)

PHAD monitors 33 fields from the Ethernet, IP, and transport layer (TCP, UDP, or ICMP) packet header. Each field is one to four bytes, divided as nearly as possible on byte boundaries as specified by the RFCs (request for comments) that specify the protocols, although we had to combine fields smaller than 8 bits (such as the TCP flags) or split fields longer than 32 bits (such as the Ethernet addresses). The value of each field is an integer. Depending on the size of the field, the value could range from 0 to $2^{32} - 1$. Because it is impractical to represent every observed value from such a large range, and because we wish to generalize over continuous values, we represent the set of observed

values with a set of contiguous ranges or clusters. Each new observed value forms a cluster by itself. If the number of clusters exceeds a limit, C, then we merge the two closest ones into a single cluster. For example, if C = 3 and we have {3-5, 8, 10-15, 20}, then we merge the two closest to form {3-5, 8-15, 20}. For the purposes of anomaly detection, the number of novel values, *r*, is the number of times the set of clusters is updated.

#### 4.3.1.2 Application Layer Anomaly Detection (ALAD)

The second component of our anomaly detection model is the application layer anomaly detector (ALAD). Instead of assigning anomaly scores to each packet, it assigns a score to an incoming server TCP connection. TCP connections are reassembled from packets. ALAD, unlike PHAD, is configured knowing the range of IP addresses it is supposed to protect, and it distinguishes server ports (0-1023) from client ports (1024-65535). We do this because most attacks are initiated by the attacker (rather than by waiting for a victim), and are therefore against servers rather than clients. We tested a large number of attributes and their combinations that we believed might make good models, and settled on five that gave the best performance individually (high detection rate at a fixed false alarm rate) on the DARPA IDS evaluation data set [12]. These are:

1. **P(src IP | dest IP)**, where *src IP* is the external source address of the client making the request, and *dest IP* is the local host address. This differs from PHAD in that the probability is conditional (a separate model for each local dest IP), only for TCP, and only for server connections (destination port < 1024). In training, this model learns the normal set of clients or users for each host. In effect, this models the set of clients allowed on a restricted service.
2. **P(src IP | dest IP, dest port)**. This model is like (1) except that there is a separate model for each server on each host. It learns the normal set of clients for each server, which may be differing across the servers on a single host.
3. **P(dest IP, dest port)**. This model learns the set of local servers which normally receive requests. It should catch probes that attempt to access nonexistent hosts or services.
4. **P(TCP flags | dest port**). This model learns the set of normal TCP flag sequences for the first, next to last, and last packet of a connection. A normal sequence is SYN (request to open), FIN-ACK (request to close and acknowledge the previous packet), and ACK (acknowledge the FIN). The model generalizes across hosts, but is separate for each port number, because the port number usually indicates the type of service (mail, web, FTP, telnet, etc.). An anomaly can result if a connection fails or is opened or closed abnormally, possibly indicating an abuse of a service.
5. **P(keyword | dest port)**. This model examines the text in the incoming request from the reassembled TCP stream to learn the allowable set of keywords for each application layer protocol. A *keyword* is defined as the first word on a line of input, i.e. the text between a linefeed and the following space. ALAD examines only the first 1000 bytes, which is sufficient for most requests. It also examines only the header part (ending with a blank line) of SMTP (mail) and HTTP (web) requests, because the header is more rigidly structured and easier to model than

the body (text of email messages or form uploads). An anomaly indicates the use of a rarely used feature of the protocol, which is common in many R2L (remote-to-local) attacks.

As with PHAD, the anomaly score is $tn/r$, where $r$ different values were observed out of $n$ training samples, and it has been $t$ seconds since the last anomaly was observed. An anomaly occurs only if the value has never been observed in training.

### 4.3.2 EXPERIMENTAL RESULTS

We evaluated PHAD and ALAD by running them at the same time on the 1999 DARPA IDS evaluation data set and merging the results. Each system was trained on week 3 (7 days, attack free) and evaluated on the 180 detectable labeled attacks from weeks 4 and 5. To merge the results, we set the two thresholds so that equal numbers of alarms were taken from both systems, and so that there were 100 total false alarms (10 per day including the missing day) after removing duplicate alarms. An alarm is considered a duplicate if it identifies the same IP address and the same attack time within 60 seconds of a higher ranked alarm from either system. We chose 60 seconds because DARPA criteria allows a detection to be counted if the time is correctly identified within 60 seconds of any portion of the attack period. Also, to be consistent with DARPA, we count an attack as detected if it identifies any IP address involved in the attack (either target or attacker). Multiple detections of the same attack (that remain after removing duplicates) are counted only once, but all false alarms are counted. In Table 1 we show the results of this evaluation. In the column labeled *det* we list the number of attacks detected out of the number of detectable instances, which does not include missing data (week 4, day 2) or the three attack types (*ntfsdos, selfping, snmpget*) that generate no inside traffic. Thus, only 180 of the 201 attack instances are listed. In the last column of Table 1, we describe the PHAD and ALAD anomalies that led to the detection, prior to removing duplicate alarms. For PHAD, the anomaly is the packet header field that contributed most to the overall score. For ALAD, each of the anomalous components (up to 5) is listed. Based on these descriptions, we adjusted the number of detections (column *det*) to remove simulation artifacts and coincidental detections, and to add detections by Ethernet address rather than IP address, which would not otherwise be counted by DARPA rules. The latter case occurs for *arppoison*, in which PHAD detects anomalous Ethernet addresses in non-IP packets. *Arppoison* disrupts network traffic by sending spoofed responses to ARP-who-has requests from a compromised local host so that IP addresses are not correctly resolved to Ethernet addresses.

The two coincidences are *mscan* (an anomalous Ethernet address, overlapping an *arppoison* attack), and *illegalsniffer* (a TCP checksum error). *Illegalsniffer* is a probe by a compromised local host being used to sniff traffic, and is detectable only in the simulation because it makes reverse DNS lookups to resolve sniffed IP addresses to host names. Because the attack is prolonged, and because all of the local hosts are victims, coincidences are likely.

Table 1. Attacks in the 1999 DARPA IDS data set [12], and the number detected (*det*) out of the total number in the available data. Detections are for merged PHAD and ALAD at 100 total false alarms, after removing coincidences and simulation artifacts (TTL field) and adding detections by Ethernet address (*arppoison*). Attacks listed do not include the 12 attacks in week 4 day 2 (missing data) or 9 attacks that leave no evidence in the inside network traffic (*selfping, snmpget*, and *ntfsdos*). Hard to detect attacks (identified by *) are those types which were detected no more than half of the time by any of the 18 original participants [12, Table 4]. Attack descriptions are due to [15].

| **Type** | **Attack and description (* = hard to detect)** | **Det** | **How detected** |
|---|---|---|---|
| Probe | illegalsniffer - compromised local host sniffs traffic | 0/2 | (1 coincidental TCP checksum error) |
| Probe | ipsweep (clear) - ping random IP addresses | 1/4 | 1 Ethernet packet size = 52, (1 TTL = 253) |
| Probe | *ipsweep (stealthy - slow scan) | 0/3 | (2 TTL = 253) |
| Probe | *ls - DNS zone transfer | 0/2 | |
| Probe | mscan - test multiple vulnerabilities | 1/1 | 1 dest IP/port, flags (1 coincidental Ethernet dest) |
| Probe | ntinfoscan - test multiple NT vulnerabilities | 2/3 | 2 HTTP "HEAD", 1 FTP "quit", 1 "user", TCP RST, (2 TTL) |
| Probe | portsweep (clear) - test multiple ports | 1/4 | 1 FIN without ACK, (1 TTL) |
| Probe | *portsweep (stealthy - slow scan) | 2/11 | 2 FIN without ACK, (5 TTL) |
| Probe | *queso - malformed packets fingerprint OS | 3/4 | 2 FIN without ACK (1 TTL) |
| Probe | *resetscan - probe with RST to hide from IDS | 0/1 | |
| Probe | satan - test multiple vulnerabilities | 2/2 | 2 HTTP/ 1 SMTP "QUIT", finger /W, IP length, src IP, (TTL) |
| DOS | apache2 - crash web server with long request | 3/3 | 3 source IP, 1 HTTP "x" and flags, TCP options in reply |
| DOS | *arppoison - spoofed replies to ARP-who-has | 3/5 | 3 Ethernet src/dest address (non-IP packet) |
| DOS | back - crash web server with "GET /////..." | 0/4 | |
| DOS | crashiis - crash NT webserver | 5/7 | 4 source IP address, 1 unclosed TCP connection |
| DOS | *dosnuke - URG data to NetBIOS crashes Windows | 4/4 | 3 URG pointer, 4 flags = UAPF |
| DOS | land - identical src/dest addr/ports crashes SunOS | 0/1 | |
| DOS | mailbomb - flood SMTP mail server | 3/3 | 3 SMTP lowercase "mail" (1 TTL = 253) |
| DOS | neptune - SYN flood crashes TCP/IP stack | 0/4 | (2 TTL = 253) |
| DOS | pod (ping of death) - oversize IP pkt crashes TCP/IP | 4/4 | 4 IP fragment pointer |
| DOS | processtable - server flood exhausts UNIX processes | 1/3 | 1 source IP address |
| DOS | smurf - reply flood to forged ping to broadcast address | 1/5 | 1 source IP address (2 TTL) |
| DOS | syslogd - crash server with forged unresolvable IP | 0/4 | |
| DOS | *tcpreset - local spoofed RST closes connections | 1/3 | 1 TCP connection not opened or closed |
| DOS | teardrop - IP fragments with gaps crashes TCP/IP stack | 3/3 | 3 frag ptr |
| DOS | udpstorm - echo/chargen loop flood | 2/2 | 2 UDP checksum error |
| DOS | *warezclient - download illegal files by FTP | 1/3 | 1 source IP address |
| DOS | warezmaster - upload illegal files by FTP | 1/1 | 1 source IP address |
| R2L | dict (guess telnet/ftp/pop) - dictionary password guessing | 3/7 | 2 FTP "user", 1 dest IP/port (POP3), 1 src IP |
| R2L | framespoofer - trojan web page | 0/1 | |
| R2L | ftpwrite - upload "+ +" to .rhosts | 0/2 | |
| R2L | guest - simple password guessing | 0/3 | |
| R2L | httptunnel - backdoor disguised as web traffic | 0/2 | |
| R2L | imap - mailbox server buffer overflow | 0/2 | |
| R2L | named - DNS nameserver buffer overflow | 0/3 | |
| R2L | *ncftp - FTP server buffer overflow | 4/5 | |
| R2L | *netbus - backdoor disguised as SMTP mail traffic | 2/3 | 4 dest IP/port, 1 SMTP "RSET", 3 auth "*xxxx*,25" |
| R2L | *netcat - backdoor disguised as DNS traffic | 2/4 | 2 source IP address, (3 TTL) |
| R2L | phf - exploit bad Apache CGI script | 2/3 | 1 src/dest IP, (1 TTL) |
| R2L | ppmacro - trojan PowerPoint macro in web page | 1/3 | 2 source IP, 1 null byte in HTTP header |
| R2L | sendmail - SMTP mail server buffer overflow | 2/2 | 1 source IP (and TTL) |
| R2L | *sshtrojan - fake ssh client steals password | 1/3 | 2 source IP address, 2 global dest IP, 1 "Sender:" |
| R2L | xlock - fake screensaver steals password | 0/3 | 1 source IP address |
| R2L | xsnoop - keystrokes intercepted on open X server | 0/3 | |
| U2R | anypw - NT bug exploit | 0/1 | |
| U2R | casesen - NT bug exploit | 2/3 | |
| U2R | eject - UNIX *suid root* buffer overflow | 1/2 | 2 FTP upload (dest IP/port 20, flags, FTP "PWD"), (1 TTL) |
| U2R | fdformat - UNIX *suid root* buffer overflow | 2/3 | 1 FTP upload (src IP, flags) |
| U2R | ffbconfig - UNIX *suid root* buffer overflow | 1/2 | 2 FTP upload (src IP, flags, FTP "STOR") |
| U2R | *loadmodule - UNIX trojan shared library | 0/2 | 1 SMTP source IP address (email upload) |
| U2R | *perl - UNIX bug exploit | 0/4 | |
| U2R | ps - UNIX bug exploit | 0/3 | |
| U2R | *sechole - NT bug exploit | 1/2 | |
| U2R | *sqlattack - database app bug, escape to user shell | 0/2 | |
| U2R | xterm - UNIX *suid root* buffer overflow | 1/3 | 1 FTP upload (dest IP/port, flags, FTP "STOR"), (1 TTL) |
| U2R | yaga - NT bug exploit | 1/4 | |
| Data | secret - copy secret files or access unencrypted | 0/4 | 1 FTP upload (source IP, dest IP/port) |
| **Total** | | **70/180** | 1 FTP upload (src IP, FTP lowercase "user" ) |
| | | | **(39%) ; and 23/65 (35%) of hard to detect attacks** |

There are 25 attacks detected by anomalous TTL values in PHAD, which we believe to be simulation artifacts. TTL (time to live) is an 8-bit counter decremented each time an IP packet is routed in order to expire packets to avoid infinite routing loops.
Although small TTL values might be used to elude an IDS by expiring the packet between the IDS and the target [16], this was not the case because the observed values were large, usually 126 or 253. Such artifacts are unfortunate, but probably inevitable, given the difficulty of simulating the Internet. A likely explanation for these artifacts is that the machine used to simulate the attacks was a different real distance from the inside sniffer than the machines used to simulate the background traffic. We did not count attacks detected solely by TTL. After adjusting the number of detections in the *det* column, we detect 70 of 180 (39%) of attacks at 100 false alarms. Among the poorly detected attacks [12, Table 1], we detect 23 of 77 (30%), or 23 of 65 (35%) of the 180 detectable attacks in our data set, almost the same rate as for the well detected attacks. This is a good result because an anomaly detection system such as ours would not be used by itself, but rather in combination with other systems such as those in the original evaluation that use signature detection or host based techniques. In order for the combination to be effective, there must be a significant non-overlap, and our results show that. We should also point out that when we developed PHAD and ALAD, we did so with the goal of improving the overall number of detections rather than just the poorly detected attacks.

More detail about algorithms and experimental results is described in [17].

### 4.4 Cost-Sensitive Modeling

Intrusion detection systems must maximize the realization of security goals while minimizing costs. In this project, we studied the problem of building cost-sensitive intrusion detection models. We examined the major cost factors associated with an IDS, which include development cost, operational cost, damage cost due to successful intrusions, and the cost of manual and automated response to intrusions. These cost factors can be qualified according to a defined attack taxonomy and site-specific security policies and priorities. We defined cost models to formulate the total expected cost of an IDS, and developed cost-sensitive machine learning techniques that can produce detection models that are optimized for user-defned cost metrics. Empirical experiments showed that our cost-sensitive modeling and deployment techniques are effective in reducing the overall cost of intrusion detection.

#### 4.4.1 Cost Factors and Metrics

In order to build cost-sensitive ID models, we must first understand the relevant cost factors and the metrics used to define them. Borrowing ideas from the related fields of credit card and cellular phone fraud detection, we identify the following major cost factors related to intrusion detection: damage cost, response cost, and operational cost. Damage cost (DCost) characterizes the amount of damage to a target resource by an attack when intrusion detection is unavailable or ineffective. Response cost (RCost) is the cost of acting upon an alarm or log entry that indicates a potential intrusion. Operational

cost (OpCost) is the cost of processing the stream of events being monitored by an IDS and analyzing the activities using intrusion detection models.

Cost-sensitive models can only be constructed and evaluated when cost metrics are given. The issues involved in the measurement of cost factors have been studied by the computer risk analysis and security assessment communities. The literature suggests that attempts to fully quantify all factors involved in cost modeling usually generate misleading results because not all factors can be reduced to discrete dollars (or some other common unit of measurement) and probabilities [18, 19, 20, 21, 22]. It is recommended that qualitative analysis be used to measure the *relative magnitudes* of cost factors. It should also be noted that cost metrics are often site-specific because each organization has its own security policies, information assets, and risk factors [23].

#### 4.4.1.1 Attack Taxonomy

An attack taxonomy is essential in producing meaningful cost metrics. The taxonomy groups intrusions into different types so that cost measurement can be performed for categories of similar attacks. Intrusions can be categorized and analyzed from different perspectives. Lindqvist and Jonsson introduced the concept of the *dimension* of an intrusion and used several dimensions to classify intrusions [24]. The *intrusion results* dimension categorizes attacks according to their effects (e.g., whether or not denial-of service is accomplished). It can therefore be used to assess the damage cost and response cost. The *intrusion techniques* dimension categorizes attacks based on their methods (e.g., resource or bandwidth consumption). It therefore affects the operational cost and the response cost. Also, the *intrusion target* dimension categorizes attacks according to the resource being targeted and affects both damage and response costs.

For example, using the DARPA Intrusion Detection Evaluation dataset, our attack taxonomy first categorizes the intrusions occurring in the dataset into ROOT, DOS, R2L, and PROBE, based on their intrusion results. Then within each of these 5 categories, the attacks are further partitioned by the techniques used to execute the intrusion. The ordering of sub-categories is of increasing complexity of the attack method. Attacks of each sub-category can be further partitioned according to the attack targets. For simplicity, the *intrusion target* dimension is not shown.

#### 4.4.1.2 Cost Factors

**Damage Cost** There are several factors that determine the damage cost of an attack. Northcutt uses *criticality* and *lethality* to quantify the damage that may be incurred by some intrusive behavior. Criticality measures the importance, or value, of the target of an attack. This measure can be evaluated according to a resource's functional role in an organization or its relative cost of *replacement*, *unavailability*, and *disclosure* [21]. Similar to Northcutt's analysis, we assign 5 points for firewalls, routers, or DNS servers, 4 points for mail or Web servers, 2 points for UNIX workstations, and 1 point for Windows or DOS workstations. Lethality measures the degree of damage that could potentially be caused by some attack. For example, a more lethal attack that helped an

intruder gain root access would have a higher damage cost than if the attack gave the intruder local user access. Other damage may include the discovery of knowledge about network infrastructure or preventing the offering of some critical service. For each main attack category in our attack taxonomy, we define a relative lethality scale and use it as the *base damage cost*, or $base_D$. When assigning damage cost according to the criticality of the target, we can use the *intrusion target* dimension. Using these metrics, we can define the damage cost of an attack targeted at some resource as $criticality \times base_D$. For example, a DOS attack targeted at a firewall has DCost=150, while the same attack targeted at a Unix workstation has DCost=60. In addition to criticality and lethality, we define the *progress* of an attack to be a measure of how successfully an attack is in achieving its goals. For example, a Denial-of-Service (DOS) attack via resource or bandwidth consumption (e.g. SYN flooding) may not incur damage cost until it has progressed to the point where the performance of the resource under attack is starting to suffer. The progress measure can be used as an estimate of the percentage of the maximum damage cost that should be accounted for. That is, the actual cost is $progress \times criticality \times base_D$. However, in deciding whether or not to respond to an attack, it is necessary to compare the maximum possible damage cost with the response cost. This requires that we assume a worst-case scenario in which $progress = 1.0$.

**Response Cost** Response cost depends primarily on the type of response mechanisms being used. This is usually determined by an IDS's capabilities, site-specific policies, attack type, and the target resource [25]. Responses may be either automated or manual, and manual responses will clearly have a higher response cost. Responses to intrusions that may be automated include the following: termination of the offending connection or session (either killing a process or resetting a network connection), rebooting the targeted system, recording the session for evidence gathering purposes and further investigation, or implementation of a packet-filtering rule [26, 23]. In addition to these responses, a notification may be sent to the administrator of the offending machine via e-mail in case that machine was itself compromised. A more advanced response which has not been successfully employed to date could involve the coordination of response mechanisms in disparate locations to halt intrusive behavior closer to its source. Additional manual responses to an intrusion may involve further investigation (perhaps to eliminate action against false positives), identification, containment, eradication, and recovery [23]. The cost of manual response includes the labor cost of the response team, the user of the target, and any other personnel that participate in response. It also includes any downtime needed for repairing and patching the targeted system to prevent future damage. We estimate the relative complexities of typical responses to each attack type in Table 1 in order to define the relative *base response cost*, or $base_R$. Again, we can take into account the criticality of the attack target when measuring response cost. That is, the cost is $criticality \times base_R$. In addition, attacks using simpler techniques generally have lower response costs than more complex attacks, which require more complex mechanisms for effective response.

**Operational Cost** The main cost inherent in the operation of an IDS is the amount of time and computing resources needed to extract and test features from the raw data stream that is being monitored1. We associate OpCost with time because a real-time IDS

must detect an attack while it is in progress and generate an alarm as quickly as possible so that damage can be minimized. A slower IDS which uses features with higher computational costs should therefore be penalized. Even if a computing resource has a "sunken cost" (e.g., a dedicated IDS box has been purchased in a single payment), we still assign some cost to the expenditure of its resources as they are used. If a resource is used by one task, it may not be used by another task at the same time. The cost of computing resources is therefore an important factor in prioritization and decision making. Some features cost more to gather than others. However, costlier features are often more informative for detecting intrusions. For example, features that examine events across a larger time window have more information available and are often used for "correlation analysis" in order to detect extended or coordinated attacks such as slow host or network scans. Computation of these features is costly because of their need to store and analyze larger amounts of data. Based on our extensive experience in extracting and constructing predictive features from network audit data, we classify features into four relative levels, based on their computational costs:

- Level 1 features can be computed from the first packet, e.g., the *service*.
- Level 2 features can be computed at any point during the life of the connection, e.g., the *connection state* (*SYN WAIT, CONNECTED, FIN WAIT,* etc.).
- Level 3 features can be computed at the end of the connection, using only information about the connection being examined, e.g., the *total number of bytes sent from source to destination*.
- Level 4 features can be computed at the end of the connection, but require access to data of potentially many other prior connections. These are the temporal and statistical features and are the most costly to compute. The computation of these features may require values of the lower level (i.e., levels 1, 2, and 3) features.

We can assign relative magnitudes to these features according to their computational costs. For example, level 1 features may cost 1, level 2 features may cost 5, level 3 features may cost 10, and level 4 features may cost 100. These estimations have been verified empirically using a prototype system for evaluating our ID models in real-time that has been built in coordination with Network Flight Recorder [27].

### 4.4.2 Cost Models

A cost model formulates the total expected cost of intrusion detection. It considers the trade-off among all relevant cost factors and provides the basis for making appropriate cost-sensitive detection decisions. We first examine the cost trade-off associated with each possible outcome of observing some event *e*, which may represent a network connection, a user's session on a system, or some logical grouping of activities being monitored. In our discussion, we say that *e=(a,p,r)* is an event described by the attack type *a* (which can be *normal* for a truly normal event), the progress *p* of the attack, and the target resource *r*. The detection outcome of *e* is one of the following: false negative (FN), false positive (FP), true positive (TP), true negative (TN), or misclassified hit. The costs associated with these outcomes are known as *consequential costs* (CCost), as they are incurred as a consequence of prediction, and are outlined in Table 2.

*FN Cost* is the cost of not detecting an attack, and is always incurred by systems that do not install IDSs. When an IDS falsely decides that a connection is not an attack and does not respond to the attack, the attack will succeed, and the target resource will be damaged. The FN Cost is therefore defined as the damage cost associated with event *e*, or DCost(*e*).

*TP Cost* is incurred in the event of a correctly classified attack, and involves the cost of detecting the attack and possibly responding to it. To determine whether response will be taken, RCost and DCost must be considered. If the damage done by the attack to resource *r* is less than RCost, then ignoring the attack actually reduces the overall cost. Therefore, if RCost(e) > DCost(e), the intrusion is not responded to beyond simply logging its occurrence, and the loss is DCost(e). Otherwise, the intrusion is acted upon and the loss is limited to RCost(e). In reality, however, by the time an attack is detected and response ensues, some damage may have incurred. To account for this, TP cost may be defined as RCost(e) + εDCost(e), where $\varepsilon \in [0,1]$ is a function of the progress p of the attack.

*FP Cost* is incurred when an event is incorrectly classified as an attack, i.e., when e=(normal,p,r) is misidentified as e=(a,p',r) for some attack. If RCost(e')≤DCost(e'), a response will ensue and the response cost, RCost(e'), must be accounted for as well. Also, since normal activities may be disrupted due to unnecessary response, false alarms should be penalized. For our discussion, we use PCost(e) to represent the penalty cost of treating a legitimate event e as an intrusion. For example, if e is aborted, PCost(e) can be the damage cost of a DOS attack on resource r , because a legitimate user may be denied access to r.

*TN Cost* is always 0, as it is incurred when an IDS correctly decides that an event is normal. We therefore bear no cost that is dependent on the outcome of the decision.

*Misclassified Hit Cost* is incurred when the wrong type of attack is identified, i.e., an event e=(a,p,r) is misidentified as e'=(a',p',r). If RCost(e') ≤ DCost(e'), a response will ensue and RCost(e') needs to be accounted for. Since the response taken is effective against attack type e' rather than e, some damage cost of εDCost(e), where $\varepsilon \in [0,1]$, will be incurred due to the true attack.

We can now define the cost model for an IDS. When evaluating an IDS over some labeled test set E, where each event, $e \in E$ has a label of *normal* or one of the intrusions, we define the cumulative cost of the IDS as follows:

$$\text{CumulativeCost}(E)=\sum_e(\text{OpCost}(e)+\text{CCost}(e))$$

where CCost(e), the consequential cost of the prediction by the IDS on e, is defined in Table 2.

Table 2: Model for Consequential Cost

| Outcome | Consequential Cost *CCost(e)* | Condition |
|---|---|---|
| Miss (*FN*) | DCost(e) | |
| False Alarm (*FP*) | RCost(e')+PCost(e);<br>0 | If DCost(e')≥RCost(e');<br>Otherwise |
| Hit (*TP*) | RCost(e) + εDCost(e);<br>DCost(e) | If DCost(e)≥RCost(e);<br>Otherwise |
| Normal (*TN*) | 0 | |
| Misclassified Hit | RCost(e') + εDCost(e);<br>DCost(e) | If DCost(e')≥RCost(e');<br>Otherwise |

### 4.4.3 Reducing Operational Cost

In order to reduce OpCost, ID models need to use low cost features as often as possible while still maintaining a desired level of accuracy. Our approach is to build multiple ID models, each of which uses different sets of features at different cost levels. Low cost models are always evaluated first by the IDS, and high cost models are used only when the low cost models cannot make a prediction with sufficient accuracy. We implemented this multiple-model approach using RIPPER [5], a rule induction algorithm.

### 4.4.4 Reducing Consequential Cost

A traditional IDS that does not consider the trade-off between RCost and DCost will attempt to respond to every intrusion that it detects. As a result, the consequential cost for FP, TP, and misclassified hits will always include some response cost. We use a cost-sensitive decision module to determine whether response should ensue based on whether DCost is greater than RCost. The decision module takes as input an intrusion report generated by the detection module. The report contains the name of the predicted intrusion and the name of the target, which are then used to look up the pre-determined DCost and RCost. If DCost ≥ RCost, the decision module invokes a separate module to initiate a response; otherwise, it simply logs the intrusion report.

### 4.4.5 Experimental Results

Our experiments used data that was distributed by the 1998 DARPA Intrusion Detection Evaluation Program. We used 80% of the data for training the detection models. The remaining 20% were used as a test set for evaluation of the cost-sensitive models.

Our results showed that the multiple-model approach can achieve a 78% reduction in operational cost, and that the consequential cost can be reduced 90%.

### 4.5 Adaptive IDS

We advocate enabling an IDS to provide performance adaptation, that is, the best possible performance for the given operation environment. It is extremely difficult, if not impossible, for an IDS to be 100% accurate. The optimal performance of an IDS should be determined by not only its ROC (Receiver Operating Characteristics) curve of detection rate versus false alarm rate, but also its cost metrics (e.g., damage cost of intrusion) and the probability of intrusion. Accordingly, performance adaptation means that an IDS should always maximize its cost-benefits for the given (current) operational conditions. For example, if an IDS is forced to miss some intrusions (that can otherwise be detected using its "signature base"), for example, due to stress or overload attacks, it should still ensure that the best value (or minimum damage) is provided according to cost-analysis on the circumstances. As a simple example, if we regard buffer-overflow as more damaging than port-scan (and for argument sake all other factors, for example, attack probability, detection probability, are equal), then missing a port-scan is better than missing a buffer-overflow. In this research, we developed a framework for considering the trade-offs of IDS performance objectives. We have developed techniques for run-time performance measurement and monitoring, and for dynamic adaptation and reconfiguration of IDS policies and mechanisms. We focused our work on misuse detection systems.

#### 4.5.1 IDS Performance Metrics

##### 4.5.1.1 Expected Value

The purpose of a real-time IDS is to detect intrusions and prevent damages. Instead of using mere statistical accuracy, we should evaluate an IDS according to its value (or cost-benefit). For each attack $A_i$, an IDS equipped with the detection rule $R_i$ (and the necessary preprocessing and logging tasks) for $A_i$ provides the expected value:

$$V_i = C^{\beta}_i \, p_i \, (1- \beta_i) - C^{\alpha}_i \, (1- p_i) \, \alpha_i$$

$C^{\beta}_i$ is the damage cost, $p_i$ is the prior priority of the intrusion, $\beta_i$ is the false negative rate, $C^{\alpha}_i$ is the false alarm cost, $\alpha_i$ is the false alarm rate. The first term is the loss (damage) prevented because of true detection, and the second term is the loss incurred because of false alarms. The total value of an IDS depends on its configuration, that is, its collection of analysis tasks and hence the attacks that it "covers". It is simply $\sum_i V_i$.

##### 4.5.1.2 Response Time

Upon arrival in the system, audit records are placed in a (common) queue (e.g., the libpcap buffer). The queue has only one server, the audit data processing and intrusion analysis unit. The processing and analysis tasks for each audit record are applied

sequentially. That is, each event goes through a sequence of analysis tasks. The process terminates if a detection rule $R_i$ determines that the event is (part of) an intrusion. Or the process ends when all analysis is done and the event is deemed normal.

The expected system time of a newly arrived audit record includes the queuing time plus the service time the record. The queuing time is simply the sum of the service time for the audit records that are already in the IDS. The service time of an audit record is the sum of processing time of each task that is applied on the record.

Obviously, if an IDS has a response time that is larger than the inter-arrival-time of audit records, the queue can be filled up and newly arrived records will be "dropped". As a result, the IDS cannot reliable detect intrusions or may output more false alarms. Therefore, it is important that an IDS operates under the constraint that its response time is smaller than the inter-arrival-time of audit records.

### 4.5.2 Performance Optimization and Adaptation

It is not always possible to run an IDS with its "full" configurations, i.e., with all analysis tasks enabled. For example, if there is a high-volume and high-speed network traffic, the inter-arrival-time of packets will be very small. If the IDS continues to run in its full configuration, its response time is likely to exceed the inter-packet-arrival-time.

Our goal is then to configure an IDS to provide the best value while operating under the above constraints. That is, if an IDS cannot accommodate all desirable analysis tasks (without violating the constraints), it should just include the more valuable tasks (we also assume that additional and orthogonal optimization techniques, such as rule-set ordering, can be used). For example, an IDS should always detect "buffer-overflow" and only analyze "slow scan" when time permits. More formally, we need to solve the following performance optimization problem: select a set of analysis tasks for the IDS in such a way that the total value of the IDS is maximized while it still operates under the constraint that its response time is smaller than the inter-arrival-time of audit records.

We note that the solution to the above optimization problem depends on the traffic and attack conditions. This means that in run-time, if we use a pre-computed IDS configuration, it may not provide the optimal value because traffic and attack conditions can change. We define performance adaptation as the process of dynamically reconfiguring an IDS to provide the optimal value given the current run-time constraints.

Performance adaptation relies on performance monitoring in run-time to detect the conditions (e.g., "stress") that cause performance degradation and to measure the parameter values needed for solving the optimization problem.

### 4.5.3 Performance Optimization and Adaptation

We experimented with two open-source IDSs, Bro and Snort. For both systems, we showed that an attacker can purposely create stress conditions, by flooding the network

with traffic that will require a lot of processing, and then launch attacks that the IDSs will miss because of packet drops.

We then modified both Bro and Snort. We added performance measurement and monitoring codes, and modules for solving the performance optimization problem. Our experiments showed that the modified systems can dynamically change configurations when stressed, in such a way that although some types of packets will not be processed, the more important attacks are still detected. That is, the modified systems have performance adaptation abilities.

Please see [28] for more detail of this work.

### 4.6 Alert Analysis and Attack Scenario Analysis

The individual alerts from IDSs alone may not be sufficient to detect or decipher the stealth or sophisticated attack activities. A higher-level analysis is necessary.
The main focus of this research task was to develop analysis algorithms that can *discover new (or novel) relationships among alerts*. Rather than relying on *a priori* alert correlation knowledge, our algorithm uses a statistical causality analysis technique call Granger Causality Test (GCT) to correlate alerts and discover (new) relationship among attack steps or anomaly activities.

The intuition is that attack steps that do not have well-known patterns or obvious relationships may nonetheless have some statistical correlations in the alert data.
GCT uses statistical functions to test if *lagged* information on a time-series variable *x* provides any statistically significant information about another time-series variable *y*. If the answer is yes, we say variable *x* Granger-causes *y*.

We model variable *y* by the Autoregressive Model (AR Model) and Autoregressive Moving Average Model (ARMA Model). GCT compares the residuals of both AR Model and ARMA Model. GCT compares the residuals of the AR Model with the residuals of the ARMA Model. Specifically, for two time series variables *y* and *x* with size *N*, the AR Model and ARMA Model of *y* are defined as:

$$\text{AR Model: } y(k) = \sum_{i=1}^{p} \theta_i y(k-i) + e_0(k);$$

$$\text{ARMA Model: } y(k) = \sum_{i=1}^{p} \alpha_i y(k-i) + \sum_{i=1}^{p} \beta_i u(k-i) + e_1(k)$$

Where *p* is a particular lag length, and parameters $\alpha_i, \beta_i, \theta_i$ $(1 \le i \le p)$ are computed in the process of solving the Ordinary Least Square (OLS) problem. The residuals of the AR Model and ARMA Model are: $R_0 = \sum_{k=1}^{T} e_0^2(k)$ and $R_1 = \sum_{k=1}^{T} e_1^2(k)$ respectively with $T=N-p$.

The Null Hypothesis $H_0$ of GCT is $H_0$: $\beta_i=0$, i=1, 2, …, p. That is, *x* does not affect *y* up

to a delay of $p$ time units. We denote $g$ as the Granger Causality Index (GCI):

$$g = \frac{(R_0 - R_1)/p}{R_1/(T-2p-1)} \sim F(p, T-2p-1)$$

Here, *F(a,b)* is Fisher's $F$ distribution with parameters $a$ and $b$ [29]. *F-test* is conducted to verify the validity of the Null Hypothesis. If the value of g is larger than a threshold in the *F-test*, then we reject the Null Hypothesis and conclude that $x$ Granger-causes $y$. The intuition of GCI $g$ is that it indicates how better variable $y$ can be predicted using histories of both variable $x$ and than it can using the history of $y$ alone. We say that variable $x_1(k)$ is more likely to be causally related with $y(k)$ than $x_2(k)$ if $g_1 > g_2$ and both have passed the *F-test*, where $g_i$, $i = 1, 2$ denotes the GCI for the input-output pair $(x_i, y)$.

Applying GCT to alert correlation, the task is to analyze the (timestamped) alert streams and determine which pairs of alerts have causal relationships. In a preliminary recent study as part of our DARPA Cyber Panel Program project, we applied our algorithms to the datasets of the DRAPA Grand Challenge Problem (GCP). The GCP dataset includes multiple stealth worm attack scenarios. Our alert correlation algorithms can correctly discover both the obvious and hidden pattern of causal relationships among attacks. For example, in Scenario I, we can also discover the mutual causal relationship between worm's malicious activities of illegal file access (to install agent software and collect sensitive data), uploading the stolen data to an external site, and downloading new agent software. In Scenario II, we can discover the causality between worm attack and server's abnormal service status.

More details can be found in [30].

## 5. Conclusions

In this project, we studied how to build cost-sensitive and light intrusion detection models. Our goal was to automate as much as analysis tasks in intrusion detection as possible. The main research activities were in:

- Automatic feature construction by analyzing the patterns of normal and intrusion activities computed from large amount of audit data
- Light-weight anomaly detection algorithms using patterns of packet headers and payloads
- Study of cost factors in intrusion detection. Using cost-sensitive machine learning algorithms to construct intrusion detection models that achieve optimal performance on the given cost metrics
- Dynamic (re-)configuration to make IDS more effective and efficient, and resilient to IDS-related attacks
- Using statistical causality analysis to discover new attack step relationships

We have developed algorithms and prototype systems, and have conducted extensive experiments using DARPA datasets and other real-world datasets. The results showed that the technologies we developed in this project are far more advanced and better than the state-of-the-art.

The aims of the project were met. In fact, we went beyond the original proposal. The results of this research have been reported in many publications. In addition, we have actively engaged in technology transfer throughout the course of the project. In particular, the PIs were involved in the founding of System Detection Inc.

## 6. Reference


[1] W. Lee. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*. PhD thesis, Columbia University, June 1999.

[2] W. Lee, S. J. Stolfo, and K. W. Mok. Algorithms for mining audit data. In T. Y. Lin, editor, *Data Mining, Rough Sets, and Granular Computing* , T. Y. Lin, Y. Y. Yao, and L. A. Zadeh (eds), Physica-Verlag, 2002.

[3] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunninghan, and M. Zissman. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, January 2000.

[4] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining, Portland, Oregon, August 1996.*

[5] W. W. Cohen. Fast effective rule induction. In *Machine Learning: the 12th International Conference*, Lake Taho, CA, 1995. Morgan Kaufmann.

[6] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, 1994.

[7] E. Eskin. Anomaly detection over noisy data using learned probability distributions. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, 2000.

[8] L. Pornoy. Intrusion detection with unlabeled data using clustering. In *Undergraduate Thesis*, Columbia University, Department of Computer Science, 2000.

[9] Barbará, D., N. Wu, S. Jajodia, Detecting Novel Network Intrusions using Bayes Estimators, First SIAM International Conference on Data Mining, 2001, http://www.siam.org/meetings/sdm01/pdf/sdm01_29.pdf

[10] Anderson, Debra, Teresa F. Lunt, Harold Javitz, Ann Tamaru, Alfonso Valdes. Detecting unusual program behavior using the statistical component of the Next generation Intrusion Detection Expert System (NIDES), Computer Science Laboratory SRI-CSL 95-06 May 1995. http://www.sdl.sri.com/papers/5/s/5sri/5sri.pdf

[11] SPADE, Silicon Defense, http://www.silicondefense.com/software/spice/

[12] Lippmann, R., et al., The 1999 DARPA Off-Line Intrusion Detection Evaluation, *Computer Networks*, 34(4) 579-595, 2000.

[13] Paxson, Vern, and Sally Floyd. The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking* (3) 226-244, 1995.

[14] Bell, Timothy, Ian H. Witten, John G. Cleary. "Modeling for Text Compression". *ACM Computing Surveys* (21)4, pp. 557-591, Dec. 1989.

[15] Kendall, Kristopher. A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems. Masters Thesis, MIT, 1999.

[16] Ptacek, Thomas H., and Timothy N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. January, 1998.

[17] Matthew V. Mahoney and Philip K. Chan. Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks. In *Proceedings of the SIGKDD '02*, July, 2002.

[18] A. M. Anderson. Comparing risk analysis methodologies. In D. T. Lindsay and W. L. Price, editors, *Information Security*. Elsevier Science Publishers, 1991.

[19] R. P. Campbell and G. A. Sands. A modular approach to computer security risk management. In *AFIPS Conference Proceedings*. AFIPS Press, 1979.

[20] DARCOM. *Engineering Design Handbook: Army Weapon Systems Analysis, Part Two (DARCOM-P 706-102)*. US Army Materiel Development And Readiness Command, 1979.

[21] D. Denning. *Information Warfare and Security*. Addison Wesley, 1999.

[22] S. Glaseman, R. Turn, and R. S. Gaines. Problem areas in computer security assessment. In *Proceedings of the National Computer Conference*, 1977.

[23] S. Northcutt. *Intrusion Detection: An Analyst's Handbook*. New Riders, 1999.

[24] U. Lindqvist and E. Jonsson. How to systematically classify computer security intrusions. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland CA, May 1997.

[25] R. Bace. *Intrusion Detection*. Macmillan Technical Publishing, 2000.

[26] E. Amoroso. *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Traps, Trace Back, and Response*. Intrusion.Net Books, 1999.

[27] Network Flight Recorder Inc. Network _ight recorder. http://www.nfr.com, 1997.

[28] Wenke Lee, Joao B. D. Cabrera, Ashley Thomas, Niranjan Balwalli, Sunmeet Saluja, and Yi Zhang. Performance Adaptation in Real-Time Intrusion Detection Systems. In *Proceedings of The 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, Zurich, Switzerland, October 2002.

[29] C.W.J. Granger. Investigating causal relations by econometric methods and cross-spectral methods. *Econometrica*, 34:424-428, 1969.

[30] Xinzhou Qin and Wenke Lee. Statistical Causality Analysis of INFOSEC Alert Data. *In Proceedingso of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID),* 2003.

## 7. Project Publications

- Xinzhou Qin and Wenke Lee. Statistical Causality Analysis of INFOSEC Alert Data. In *Proceedings of The 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, Pittsburgh, PA, September 2003.
- Henry H. Feng, Oleg Kolesnikov, Prahlad Fogla, Wenke Lee, and Weibo Gong. Anomaly Detection Using Call Stack Information. In *Proceedings of The 2003 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2003.
- Wenke Lee, Joao B. D. Cabrera, Ashley Thomas, Niranjan Balwalli, Sunmeet Saluja, and Yi Zhang. Performance Adaptation in Real-Time Intrusion Detection Systems. In *Proceedings of The 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, Zurich, Switzerland, October 2002.
- Wenke Lee, Wei Fan, Matt Miller, Sal Stolfo, and Erez Zadok. Toward Cost-Sensitive Modeling for Intrusion Detection and Response. *Journal of Computer Security,* Vol. 10, Numbers 1,2, 2002
- Xinzhou Qin, Wenke Lee, Lundy Lewis, and Joao B. D. Cabrera. Integrating Intrusion Detection and Network Management. In *Proceedings of The IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)*, Florence, Italy, May 2002.
- Wei Fan, Matt Miller, Sal Stolfo, Wenke Lee, and Phil Chan. Using Artificial Anomalies to Detect Unknown and Known Network Intrusions. In *Proceedings of The First IEEE International Conference on Data Mining*, San Jose, CA, November 2001.
- Wenke Lee, Sal Stolfo, Phil Chan, Eleazar Eskin, Wei Fan, Matt Miller, Shlomo Hershkop, and Junxin Zhang. Real Time Data Mining-based Intrusion Detection. In *Proceedings of The 2001 DARPA Information Survivability Conference and Exposition (DISCEX II)* (selected for presentation), Anaheim, CA, June 2001.
- Wenke Lee and Dong Xiang. Information-Theoretic Measures for Anomaly Detection. In *Proceedings of The 2001 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.
- Salvatore J. Stolfo,Shlomo Hershkop, Ke Wang, Olivier Nimerkern and Chia-Wei Hu. A Behavior-based Approach to Securing Email Systems. *"Mathematical*

*Methods, Models and Architectures for Computer Networks Security"*, Proceedings published by Springer Verlag, Sept. 2003. [ PDF]

- Katherine A Heller, Krysta M Svore, Angelos D. Keromytis, and Salvatore J. Stolfo. "One Class Support Vector Machines for Detecting Anomalous Window Registry Accesses". *3rd IEEE Conference Data Mining Workshop on Data Mining for Computer Security,* Florida, November 19, 2003. [ PDF]
- Salvatore J. Stolfo, Shlomo Hershkop, Ke Wang, Olivier Nimeskern, and Chia-Wei Hu. ``Behavior Profiling of Email" *1st NSF/NIJ Symposium on Intelligence & Security Informatics(ISI 2003).* June 2-3,2003,Tucson,Arizona,USA. [full paper, PDF]
- Manasi Bhattacharyya, Shlomo Hershkop, Eleazar Eskin, and Salvatore J. Stolfo. ``MET: An Experimental System for Malicious Email Tracking." *In Proceedings of the 2002 New Security Paradigms Workshop (NSPW-2002)*. Virginia Beach, VA: September 23rd - 26th, 2002. [full paper, PDF]
- Frank Apap, Andrew Honig, Shlomo Hershkop, Eleazar Eskin, Salvatore J. Stolfo. ``Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses." *In Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID-2002)*. Zurich, Switzerland: October 16-18, 2002. [full paper, PDF]
- Suhail Mohiuddin, Shlomo Hershkop, Rahul Bhan, Salvatore J. Stolfo. ``Defending against a large Scale Denail of Service Attack" *In Proceedings of the 3rd Annual IEEE Information Assurance Workshop* . United States Military Academy West Point, New York: June 17-19, 2002. [full paper, PDF]
- Leonid Portnoy, Eleazar Eskin and Salvatore J. Stolfo. ``Intrusion detection with unlabeled data using clustering" *To Appear in Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*. Philadelphia, PA: November 5-8, 2001. [ full paper, PDF]
- Eleazar Eskin, Wenke Lee and Salvatore J. Stolfo. ``Modeling System Calls for Intrusion Detection with Dynamic Window Sizes." *Proceedings of DISCEX II*. June 2001. [ full paper, PDF]
- Matthew G. Schultz, Eleazar Eskin, and Salvatore J. Stolfo. ``Malicious Email Filter - A UNIX Mail Filter that Detects Malicious Windows Executables." *Proceedings of USENIX Annual Technical Conference - FREENIX Track*. Boston, MA: June 2001. (Best Student Paper Award) [ full paper, PDF]
- Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. ``Data Mining Methods for Detection of New Malicious Executables". *In Proceedings of IEEE Symposium on Security and Privacy*. Oakland, CA: May 2001. [ full paper, PDF]
- Leonid Portnoy. ``Intrusion Detection with Unlabeled Data using Clustering" *Undergraduate Thesis*. Columbia University: December, 2000. [ full paper, PDF]
- Eleazar Eskin, Matthew Miller, Zhi-Da Zhong, George Yi, Wei-Ang Lee, Sal Stolfo. ``Adaptive Model Generation for Intrusion Detection Systems" *Workshop on Intrusion Detection and Prevention, 7th ACM Conference on Computer Security*, Athens, GR: November, 2000. [full paper]
- Eskin, Eleazar. ``Anomaly Detection over Noisy Data using Learned Probability Distributions" *ICML00*, Palo Alto, CA: July, 2000. [abstract, full paper]

- Wei Fan, Wenke Lee, Sal Stolfo, and Matthew Miller. ``A Multiple Model Cost-Sensitive Approach for Intrusion Detection" *Eleventh European Conference on Machine Learning (ECML '00)* 2000. [ full paper]
- Andrew Honig, Andrew Howard, Eleazar Eskin, and Salvatore Stolfo. "Adaptive Model Generation: An Architecture for the Deployment of Data Mining-based Intrusion Detection Systems." in Data Mining for Security Applications. Kluwer 2002.
- M. Mahoney & P. Chan. Learning Rules for Anomaly Detection of Hostile Network Traffic. *Proc. Third IEEE Intl. Conf. on Data Mining (ICDM)*, pp. 601-4, 2003.
- G. Tandon & P. Chan. Learning Rules from System Call Arguments and Sequences for Anomaly Detection. *ICDM Workshop on Data Mining for Computer Security (DMSEC)*, pp. 20-29, 2003.
- R. Vargiya & P. Chan. Boundary Detection in Tokenizing Network Application Payload for Anomaly Detection. *ICDM Workshop on Data Mining for Computer Security (DMSEC)*, pp. 50-59, 2003.
- P. Chan, M. Mahoney & M. Arshad. Learning Rules and Clusters for Anomaly Detection in Network Traffic. *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava & A. Lazarevic (editors), Kluwer, to appear, 2003.
- M. Mahoney and P. Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. *Proc. 6th Intl. Symp. Recent Advances in Intrusion Detection*, p. 220-237, 2003.
- M. Mahoney and P. Chan. Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks. *Proc. Eighth Intl. Conf. Knowledge Discovery and Data Mining*, p376-385, 2002.