# Real Time Data Mining-based Intrusion Detection

Wenke Lee[1], Salvatore J. Stolfo[2], Philip K. Chan[3],
Eleazar Eskin[2], Wei Fan[4], Matthew Miller[2], Shlomo Hershkop[2], and Junxin Zhang[2]

[1]Computer Science Department, North Carolina State University, Raleigh, NC 27695
wenke@csc.ncsu.edu

[2]Computer Science Department, Columbia University, New York, NY 10027
{sal,eeskin,mmiller,sh53,jzhang}@cs.columbia.edu

[3]Computer Science Department, Florida Institute of Technology, Melbourne, FL 32901
pkc@cs.fit.edu

[4] IBM T.J.Watson Research Center, Hawthorne, NY 10532
weifan@us.ibm.com

## Abstract

*In this paper, we present an overview of our research in real time data mining-based intrusion detection systems (IDSs). We focus on issues related to deploying a data mining-based IDS in a real time environment. We describe our approaches to address three types of issues: accuracy, efficiency, and usability. To improve accuracy, data mining programs are used to analyze audit data and extract features that can distinguish normal activities from intrusions; we use artificial anomalies along with normal and/or intrusion data to produce more effective misuse and anomaly detection models. To improve efficiency, the computational costs of features are analyzed and a multiple-model cost-based approach is used to produce detection models with low cost and high accuracy. We also present a distributed architecture for evaluating cost-sensitive models in real-time. To improve usability, adaptive learning algorithms are used to facilitate model construction and incremental updates; unsupervised anomaly detection algorithms are used to reduce the reliance on labeled data. We also present an architecture consisting of sensors, detectors, a data warehouse, and model generation components. This architecture facilitates the sharing and storage of audit data and the distribution of new or updated models. This architecture also improves the efficiency and scalability of the IDS.*

## 1 Introduction

Security of network systems is becoming increasingly important as more and more sensitive information is being stored and manipulated online. Intrusion Detection Systems (IDSs) have thus become a critical technology to help protect these systems.

Most IDSs are based on hand-crafted signatures that are developed by manual encoding of expert knowledge. These systems match activity on the system being monitored to known signatures of attacks. The major problem with this approach is that these IDSs fail to generalize to detect new attacks or attacks without known signatures. Recently, there has been an increased interest in data mining-based approaches to building detection models for IDSs. These models generalize from both known attacks and normal behavior in order to detect unknown attacks. They can also be generated in a quicker and more automated method than manually encoded models that require difficult analysis of audit data by domain experts. Several effective data mining techniques for detecting intrusions have been developed [23, 11, 34, 5], many of which perform close to or better than systems engineered by domain experts.

However, successful data mining techniques are themselves not enough to create deployable IDSs. Despite the promise of better detection performance and generalization ability of data mining-based IDSs, there are some inherent difficulties in the implementation and deployment of these systems. We can group these difficulties into three general categories: accuracy (i.e., detection performance), efficiency, and usability. Typically, data mining-based IDSs (especially anomaly detection systems) have higher false positive rates than traditional hand-crafted signature based methods, making them unusable in real environments. Also, these systems tend to be inefficient (i.e., computationally expensive) during both training and evaluation. This prevents them from being able to process audit data and detect intrusions in real time. Finally, these systems require large amounts of training data and are significantly more complex than traditional systems. In order to be able to deploy real time data mining-based IDSs, these issues must be addressed.

In this paper, we discuss several problems inherent in developing and deploying a real-time data mining-based IDS and present an overview of our research, which addresses these problems. These problems are independent of the actual learning algorithms or models used by an IDS and must be overcome in order to implement data mining methods in a deployable system.

An effective data mining-based IDS must address each of these three groups of issues. Although there are tradeoffs between these groups, each can generally be handled separately. We present the key design elements and group them into which general issues they address.

The rest of this paper is organized as follows. In Section 2, we discuss the issues related to detection performance and provide several algorithm independent approaches to improve the accuracy of a system. In Section 3, we discuss efficiency issues and present techniques for efficient model computation. In Section 4, we discuss general usability issues and techniques to address these issues and in Section 5 we discuss a system architecture for implementing the techniques presented throughout the paper. Section 6 surveys related research, and Section 7 offers discussion of future work and conclusive remarks.

## 2 Accuracy

Crucial to the design and implementation of effective data mining-based IDS is defining specifically how detection performance, or accuracy, of these systems is measured. Because of the difference in nature between a data mining-based system and a typical IDS, the evaluation metrics must take into account factors which are not important for traditional IDSs.

At the most basic level, accuracy measures how well an IDS detects attacks. There are several key components of an accuracy measurement. One important component is detection rate, which is the percentage of attacks that a system detects. Another component is the false positive rate, which is the percentage of normal data that the system falsely determines to be intrusive. These quantities are typically measured by testing the system on a set of data (normal and intrusions) that are not seen during the training of the system in order to simulate an actual deployment.

There is an inherent tradeoff between detection rate and false positive rate. One way to represent this tradeoff is by plotting the detection rate versus false positive rate on a curve under different parameter values creating a ROC curve[1]. A method to compare accuracy between two IDSs is to examine their ROC curves.

In practice, only the small portion of a ROC curve corresponding to acceptably low false positives is of interest, as in a deployable system, only a low false positive rate can be tolerated. Hand-crafted methods typically have a fixed detection threshold, they perform at a constant detection rate across different false positive rates. In a ROC curve, we can assume that their curve is a straight line at each detection level. Data mining-based systems have the advantage of potentially being able to detect new attacks that hand-crafted methods tend to miss. Data mining-based IDSs are only useful if their detection rate is higher than a hand-crafted method's detection rate with an acceptably low false positive rate. Given this framework, our goal is to develop a data mining-based IDS that is capable of outperforming hand-crafted signature-based systems at the tolerated false positive rate.

We have developed and applied a number of algorithm-independent techniques to improve the performance of data mining-based IDSs. In this section, we focus on a few particular techniques that have been proven to be empirically successful. We first present a generic framework for extracting features from audit data which help discriminate attacks from normal data. These features can then be used by any detection model building algorithm. We then describe a method for generating artificial anomalies in order to decrease the false positive rate of anomaly detection algorithms. Our research has shown that by generating artificial anomalies, we can improve the accuracy of these ID models. Finally, we present a method for combining anomaly and misuse (or signature) detection models. Typically misuse and anomaly detection models are trained and used in complete isolation from each other. Our research has shown that by combining the two types of models, we can improve the overall detection rate of the system without compromising the benefits of either detection method.

### 2.1 Feature Extraction for IDS

Two basic premises of intrusion detection are that system activities are observable, e.g., via auditing, and there is dis-

[1]Receiver Operating Characteristic (ROC) graphs are used in many detection problems because they depict the tradeoffs between detection rate and false positive rate [4].

tinct 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 [19]. First, raw (binary) audit data is processed and summarized into discrete records containing a number of basic features such as in the case of network traffic: timestamp, duration, source and destination IP addresses and ports, and error condition flags. Specialized data mining programs [24] are then applied to these records to compute frequent patterns describing correlations among features and frequently co-occurring events across many records. A pattern is typically in the form $A, B \rightarrow C, D$ [*confidence*, *support*] which translates to event $A$ and $B$ are followed by events $C$ and $D$ with a certain confidence and occur with a certain frequency in the data (the pattern's support). 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 [25] showed that an IDS model constructed using these 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*). We compared 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 [26], has the highest "intrusion-only" (i.e., unique for the intrusion) score: "$(flag = S0, service = http, dst\_host = victim)$, $(flag = S0, service = http, dst\_host = victim)$ $\rightarrow (flag = S0, service = http, dst\_host = victim)$ $[0.93, 0.03, 2]$." This means that 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 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 belonging to *syn_flood* have values 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 [3], an inductive rule learner, to compute a detection rule for *syn_flood* using these extracted features: **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. Details of the algorithm are given in [24].

## 2.2 Artificial Anomaly Generation

A major difficulty in using machine learning methods for anomaly detection lies in making the learner discover boundaries between known and unknown classes. Since there are no examples of anomalies in our training data (by definition of anomaly), a machine learning algorithm will only uncover boundaries that separate different known classes in the training data. A machine learning algorithm will not specify a boundary between the known data and unseen data (anomalies). We present the technique of *artificial anomaly generation* to enable traditional learners to detect anomalies. Artificial anomalies are injected into the training data to help the learner discover a boundary around the original data. All artificial anomalies are given the class label *anomaly*. Our approach to generating artificial anomalies focuses on "near misses," instances that are close to the known data, but are not in the training data. We assume the training data are representative, hence near misses can be safely assumed to be anomalous.

Since we do not know where the exact decision boundary is between known and anomalous instances, we assume that the boundary may be very close to the existing data. To generate artificial anomalies close to the known data, a useful heuristic is to randomly change the value of one feature of an example to a value that does not occur in the data while leaving the other features unaltered.

Some regions of known data in the instance space may be sparsely populated. We can think of the sparse regions as small islands and dense regions as large islands in an ocean. To avoid overfitting, learning algorithms are usually biased towards discovering more general models. Since we only have known data, we want to prevent models from being overly general when predicting these known classes. That is, we want to avoid the situation where sparse regions may be grouped into dense regions to produce singularly large regions covered by overly general models. It is possible to produce artificial anomalies around the edges of these sparse regions and coerce the learning algorithm to discover the specific boundaries that distinguish these regions from the rest of the instance space. In other words, we want to generate data that will amplify these sparse regions.

Sparse regions are characterized by infrequent values of individual features. To amplify sparse regions, we proportionally generate more artificial anomalies around sparse regions depending on their sparsity using the *Distribution Based Artificial Anomaly* algorithm (DBA2) presented in detail in [7] along with results of experiments demonstrating the effectiveness of the method. The algorithm uses the frequency distribution of each feature's values to proportionally generate a sufficient amount of anomalies.

### 2.3 Combined Misuse and Anomaly Detection

Traditionally, anomaly detection and misuse detection are considered separate problems. Anomaly detection algorithms typically train over normal data while misuse algorithms typically train over labeled normal and intrusion data. Intuitively a hybrid approach should perform better, in addition, it has the obvious efficiency advantages in both model training and deployment than using two different models. We use the artificial anomaly generation method to create a single model that is both a misuse and anomaly detection method. This allows us to use traditional supervised inductive learning methods for both anomaly detection and misuse detection at the same time. We train a single model from a set of data that contains both normal records and records corresponding to intrusions. In addition, we also generate artificial anomalies using the DBA2 algorithm and train a learning algorithm over the combined dataset. The learned model can detect anomalies and intrusions concurrently.

We learn a single ruleset for combined misuse and anomaly detection. The ruleset has rules to classify a connection to be normal, one of the known intrusion classes, or anomaly. In order to evaluate this combined approach, we group intrusions together into a number of small clusters. We create multiple datasets by incrementally adding each cluster into the dataset and re-generating artificial anomalies. This is to simulate the real world process of developing and discovering new intrusions and incorporating them into the training set. We learn models that contain misuse rules for the intrusions that are known in the training data, anomaly detection rules for unknown intrusions in left-out clusters, and rules that characterize normal behavior.

We have seen that the detection rates of known intrusions classified by misuse rules in models learned with and without artificial anomalies are indistinguishable or completely identical. This observation shows that the proposed DBA2 method does not diminish the effectiveness of misuse rules in detecting particular categories of known intrusions. Our experiments with pure anomaly and combined anomaly and misuse detection can be found in [7].

## 3 Efficiency

In typical applications of data mining to intrusion detection, detection models are produced off-line because the learning algorithms must process tremendous amounts of archived audit data. These models can naturally be used for off-line intrusion detection (i.e., analyzing audit data off-line after intrusions have run their course). Effective intrusion detection should happen in real-time, as intrusions take place, to minimize security compromises. In this section, we discuss our approaches to make data mining-based ID models work efficiently for real-time intrusion detection.

In contrast to off-line IDSs, a key objective of real-time IDS is to detect intrusions as early as possible. Therefore, the efficiency of the detection model is a very important consideration. Because our data mining-based models are computed using off-line data, they implicitly assume that when an event is being inspected (i.e., classified using an ID model), all activities related to the event have completed so that all features have meaningful values available for model checking. As a consequence, if we use these models in real time without any modification, then an event is not inspected until complete information about that event has arrived and been summarized, and all temporal and statistical features (i.e., the various temporal statistics of the events in the past $n$ seconds, see Section 2.1) are computed. This scheme can fail miserably under real-time constraints. When the volume of an event stream is high, the amount of time taken to process the event records within the past $n$ seconds and calculate statistical features is also very high. Many subsequent events may have terminated (and thus completed with attack actions) when the "current" event is finally inspected by the model. That is, the detection of intrusions is severely delayed. Unfortunately, DoS attacks, which typically generate a large amount of traffic in a very short period time, are often used by intruders to first overload an IDS, and use the detection delay as a window of opportunity to quickly perform their malicious intent. For example, they can even seize control of the host on which the IDS lives, thus eliminating the effectiveness of intrusion detection altogether.

It is necessary to examine the time delay associated with computing each feature in order to speed up model evaluation. The time delay of a feature includes not only the time spent for its computation, but also the time spent waiting for its readiness (i.e., when it can be computed). For example, in the case of network auditing, the *total duration* of a network connection can only be computed after the last packet of the connection has arrived, whereas the *destination host* of a connection can be obtained by checking the header of the first packet.

From the perspective of cost analysis, the efficiency of an intrusion detection model is its *computational cost*, which is the sum of the time delay of the features used in the model. Based on the feature construction approaches discussed in Section 2.1, we can categorize features used for network intrusion detection into 4 cost levels:

- 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.

In order to conveniently estimate the cost of a rule, we assign a cost of 1 to the level 1 features, 5 to the level 2 features, 10 to level 3, and 100 to level 4. These cost assignments are very close to the actual measurements we have obtained via extensive real-time experiments. However, we have found that the cost of computing level 4 features is linearly dependent on the amount of connections being monitored by the IDS within the time window used for computation, as they require iteration of the complete set of recent connections.

In this section, we discuss approaches to reduce computational cost and improve the efficiency of the real-time intrusion detection models. In Section 3.1, we describe our techniques for *cost-sensitive modeling*. In Section 3.2, we discuss a real-time system for implementing *distributed feature computation* for evaluating multiple cost-sensitive models.

### 3.1 Cost-Sensitive Modeling

In order to reduce the computational cost of an IDS, detection rules need to use low cost features as often as possible while maintaining a desired accuracy level. We propose a multiple ruleset approach in which each ruleset uses features from different cost levels. Low cost rules are always evaluated first by the IDS, and high cost rules are used only when low cost rules cannot predict with sufficient accuracy.

In the domain of network intrusion detection, we use four different levels of costs to compute features, as discussed in Section 3. Features of costs 1, 5, and 10 are computed individually and features of costs 100 can be computed in a single lookup of all the connections in the past $n$ seconds. With the above costs and goals in mind, we use the following multiple ruleset approach:

- We first generate multiple training sets $T_{1-4}$ using different feature subsets. $T_1$ uses only cost 1 features. $T_2$ uses features of costs 1 and 5, and so forth, up to $T_4$, which uses all available features.
- Rulesets $R_{1-4}$ are learned using their respective training sets. $R_4$ is learned as an ordered ruleset[2] for its efficiency, as it may contain the most costly features. $R_{1-3}$ are learned as un-ordered rulesets[3], as they will contain accurate rules for classifying $normal$ connections.
- A $precision$ measurement $p_r$[4] is computed for *every rule*, $r$, except for the rules in $R_4$.
- A threshold value $\tau_i$ is obtained for every single class which determines the tolerable precision required in order for a classification to be made by any ruleset except for $R_4$.

In real-time execution, the feature computation and rule evaluation proceed as follows:

- All cost 1 features used in $R_1$ are computed for the connection being examined. $R_1$ is then evaluated and a prediction $i$ is made.
- If $p_r > \tau_i$, the prediction $i$ is fired. In this case, no more features are computed and the system examines the next connection. Otherwise, additional features required by $R_2$ are computed and $R_2$ is evaluated in the same manner as $R_1$.
- Evaluation continues with $R_3$, followed by $R_4$, until a prediction is made.
- When $R_4$ (an ordered ruleset) is reached, features are computed as needed while evaluation proceeds from the top of the ruleset to the bottom. The evaluation of $R_4$ does not require any firing condition and will always generate a prediction.

[2]An ordered ruleset is in the form of "**if** condition$_1$ **then** action$_1$ **else if** condition$_2$ **then** action$_2$ ... **else** action$_n$." The rules are checked sequentially. We typically place rule for the most prevalent class, i.e., *normal*, as the first rule.

[3]An unordered ruleset is in the form of "**if** condition$_1$ **then** action$_1$; **if** condition$_2$ **then** action$_2$; ...; **if** condition$_n$ **then** action$_n$." The rules can be checked in parallel.

[4]Precision describes how accurate a prediction is. Precision is defined as $p = \frac{|P \cap W|}{|P|}$, where $P$ is the set of predictions with label $i$, and $W$ is the set of all instances with label $i$ in the data set.

In our experiments, we used data from the 1998 DARPA evaluation. The detailed experimental set-up and results can be found in [8]. In summary, our multiple model approach can reduce the computational cost by as much as 97% without compromising predictive accuracy, where the cost for inspecting a connection is the total computational cost of all unique features used before a prediction is made. If multiple features of cost 100 are used, the cost is counted only once since they can all be calculated in a single iteration through the table of recent connections.

### 3.2 Distributed Feature Computation

We have implemented a system that is capable of evaluating a set of cost-sensitive models in real-time. This system uses a sensor for extracting light-weight, or "primitive," features from raw network traffic data to produce connection records, and then offloads model evaluation and higher level feature computation to a separate entity, called JUDGE. The motivation for offloading this computation and evaluation is that it is quite costly and we do not wish to overburden the sensor (in this case a packet sniffing engine).

JUDGE uses models that have been learned using the techniques described in Section 3.1. That is, there exists a sequence of models, each of which uses increasingly more costly features than the previous model. Models are evaluated and higher level features are computed at different points in a connection by JUDGE as more primitive features become available.

The sensor informs JUDGE of new feature values, or updates to feature values that are maintained throughout the connection's life, whenever there is a change in the connection's state (e.g., a connection has gone from SYN_WAIT to CONNECTED). Sensors also update certain feature values whenever there is an "exception" event. Exceptions are certain occurrences which should immediately update the value of a specific feature. For example, if two fragmented packets come in and the offsets for defragmentation are correct, the *bad_frag_offset* feature must be updated immediately.

Upon each state change and exception event, JUDGE computes the set of features that are available for the given connection. If the set of features is a proper subset of the set of light-weight features (the level 1 and level 2 features described earlier) used by one of the ID models, then higher level features are computed and that model is evaluated. The logic for determining when a prediction is made is the same as is described in Section 3.1.

Once a prediction is made by JUDGE, a complete connection record, with the label, is inserted into a data warehouse, as described in our system architecture outline in Section 5.

We have currently implemented this system using NFR's Network Flight Recorder as the sensor, although the protocol for communication between the sensor and JUDGE would allow any sensor which extracts features from a data stream to be used.

## 4 Usability

A data mining-based IDS is significantly more complex than a traditional system. The main cause for this is that data mining systems require large sets of data from which to train. The hope to reduce the complexity of data mining systems has led to many active research areas.

First, management of both training and historical data sets is a difficult task, especially if the system handles many different kinds of data. Second, once new data has been analyzed, models need to be updated. It is impractical to update models by retraining over all available data, as retraining can take weeks, or even months, and updated models are required immediately to ensure the protection of our systems. Some mechanism is needed to adapt a model to incorporate new information. Third, many data mining-based IDSs are difficult to deploy because they need a large set of clean (i.e., not noisy) labeled training data. Typically the attacks within the data must either be manually labeled for training signature detection models, or removed for training anomaly detection models. Manually cleaning training data is expensive, especially in the context of large networks. In order to reduce the cost of deploying a system, we must be able to minimize the amount of clean data that is required by the data mining process.

We present an approach to each of these problems. We use the technique *adaptive learning*, which is a generic mechanism for adding new information to a model without retraining. We employ *unsupervised anomaly detection* which is a new class of intrusion detection algorithms that do not rely on labeled data. In the next section, we present a system architecture which automates model and data management.

### 4.1 Adaptive Learning

We propose to use ensembles of classification models ($R_1, \ldots, R_4$ described earlier is an example of an ensemble of classification models) as a general and algorithm-independent method to adapt existing models in order to detect newly established patterns. Our goal is to improve the efficiency of both learning and deployment. In reality, when a new type of intrusion is discovered, it is very desirable to be able to quickly adjust an existing detection system to detect the new attack, even if the adjustment is temporary and may not detect all occurrence of the new attack. At the same time, after we have at least some method of defense, we can look for possibly better ways to detect the attack which involves recomputing the detection model and may take a much longer period of time to compute. When a better model is computed, we may choose to replace the temporary model. For such purposes, we seek a "plug-in" method, i.e., we efficiently generate a simple model that is only good at detecting the new intrusion, and plug or attach it to the existing models to enable detection of new intrusions. Essentially, we efficiently generate a light-weight classifier (i.e., classification model) for the new pattern. The existing main detection model remain the same. When the old model detects an anomaly, this data record is sent to the new classifier for further classification. The final prediction is a function of the predictions of both the old classifier and the new classifier. Computing the new classifier is significantly faster than generating a monolithic model for all established patterns and anomalies.

In [7], different configurations for adaptive learning are discussed. In one such configuration, given an existing classifier $H_1$, an additional classifier, $H_2$, is trained from data containing normal records and records corresponding to the new intrusion. We refer to $H_1$ as the existing IDS model, while $H_2$ refers to a new model trained specifically for a new or recently discovered attack. The decision rules in Figure 1 are evaluated to compute the final outcome. This method is independent of the actual model building algorithm. Each classifier can be anything from a decision tree, to a rule based learner, to a neural network, etc.

We have experimented with different configurations to test the effectiveness of this approach. As shown in [7], the cost of training of the proposed method (as measured in Section 3.1) is almost 150 times less expensive than learning a monolithic classifier trained from all available data, and the accuracy of both are essentially equivalent.

### 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.

In this section, we present the problem of unsupervised anomaly detection and relate it to the problem of outlier detection in statistics [1]. We present 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 [1].

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 build detection models over system calls and a clustering based unsupervised anomaly detection algorithm for network traffic.

The probabilistic approaches detect 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 [5].

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

- **if** $(H_1(\mathbf{x}) = normal) \vee (H_1(\mathbf{x}) = anomaly)$ **then**
  - **if** $H_2(\mathbf{x}) = normal$
    **then** $output \leftarrow H_1(\mathbf{x})$ ($normal$ or $anomaly$)
  - **else** $output \leftarrow new\_intrusion$
- **else** $output \leftarrow H_1(\mathbf{x})$

**Figure 1. Ensemble-based Adaptive Learning Configuration**

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 [27].

## 5 System Architecture

The overall system architecture is designed to support a data mining-based IDS with the properties described throughout this paper. As shown in Figure 2, the architecture consists of sensors, detectors, a data warehouse, and a model generation component. This architecture is capable of supporting not only data gathering, sharing, and analysis, but also data archiving and model generation and distribution.

The system is designed to be independent of the sensor data format and model representation. A piece of sensor data can contain an arbitrary number of features. Each feature can be continuous or discrete, numerical or symbolic. In this framework, a model can be anything from a neural network, to a set of rules, to a probabilistic model. To deal with this heterogeneity, an XML encoding is used so each component can easily exchange data and/or models.

Our design was influenced by the work in standardizing the message formats and protocols for IDS communication and collaboration: the Common Intrusion Detection Framework (CIDF, funded by DARPA) [29] and the more recent Intrusion Detection Message Exchange Format (IDMEF, by the Intrusion Detection Working Group of IETF, the Internet Engineering Task Force). Using CIDF or IDMEF, IDSs can securely exchange attack information, encoded in the standard formats, to collaboratively detect distributed intrusions. In our architecture, data and model exchanged between the components are encoded in our standard message format, which can be trivially mapped to either CIDF or IDMEF formats. The key advantage of our architecture is its high performance and scalability. That is, all components can reside in the same local network, in which case, the work load is distributed among the components; or the components can be in different networks, in which case, they can also participate in the collaboration with other IDSs in the Internet.

In the following sections we describe the components depicted in Figure 2 in more detail. A complete description of the system architecture is given in [6].

### 5.1 Sensors

Sensors observe raw data on a monitored system and compute features for use in model evaluation. Sensors insulate the rest of the IDS from the specific low level properties of the target system being monitored. This is done by having all of the sensors implement a Basic Auditing Module (BAM) framework. In a BAM, features are computed from the raw data and encoded in XML.

### 5.2 Detectors

Detectors take processed data from sensors and use a detection model to evaluate the data and determine if it is an attack. The detectors also send back the result to the data warehouse for further analysis and report.

There can be several (or multiple layers of) detectors monitoring the same system. For example, work loads can be distributed to different detectors to analyze events in parallel. There can also be a "back-end" detector, which employs very sophisticated models for correlation or trend analysis, and several "front-end" detectors that perform quick and simple intrusion detection. The front-end detectors keep up with high-speed and high-volume traffic, and must pass data to the back-end detector to perform more thorough and time consuming analysis.

### 5.3 Data Warehouse

The data warehouse serves as a centralized storage for data and models. One advantage of a centralized repository for the data is that different components can manipulate the same piece of data asynchronously with the existence of a database, such as off-line training and manually labeling. The same type of components, such as multiple sensors, can manipulate data concurrently. Relational database features support "stored procedure calls" which enable easy implementation of complicated calculations, such as efficient data sampling carried out automatically on the server. Arbitrary

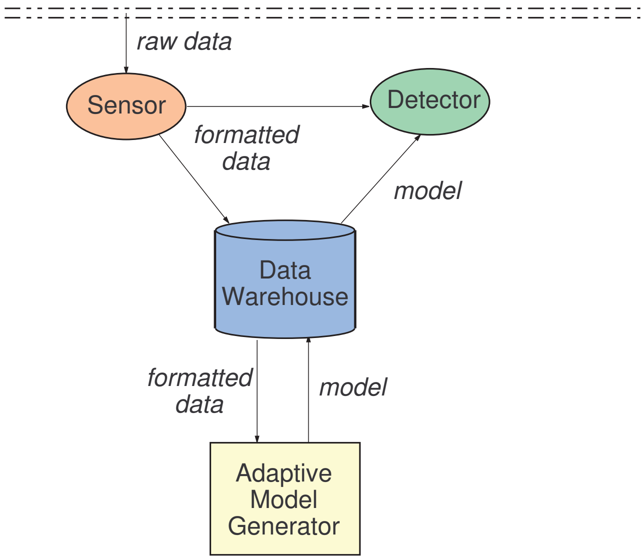


**Figure 2. The Architecture of Data Mining-based IDS**

amount of sensor data can also be retrieved by a single SQL query. Distribution of detection models can be configured to push or pull.

The data warehouse also facilitates the integration of data from multiple sensors. By correlating data/results from different IDSs or data collected over a longer period of time, the detection of complicated and large scale attacks becomes possible.

### 5.4 Model Generator

The main purpose of the model generator is to facilitate the rapid development and distribution of new (or updated) intrusion detection models. In this architecture, an attack detected first as an anomaly may have its exemplary data processed by the model generator, which in turn, using the archived (historical) normal and intrusion data sets from the data warehouse, automatically generates a model that can detect the new intrusion and distributes it to the detectors (or any other IDSs that may use these models). Especially useful are unsupervised anomaly detection algorithms because they can operate on unlabeled data which can be directly collected by the sensors.

We have successfully completed a prototype implementation of a data mining and CIDF based IDS [20]. In this system, a data mining engine, equipped with our feature extraction programs (see Section 2.1) and machine learning programs, serves as the model generator for several detectors. It receives audit data for anomalous events (encoded as a GIDO, the Generalized Intrusion Detection Objects) from a detector, computes patterns from the data, compares them with historical normal patterns to identify the "unique" intrusion patterns, and constructs features accordingly. Machine learning algorithms are then applied to compute the detection model, which is encoded as a GIDO and sent to all the detectors. Much of the design and implementation efforts had been on extending the Common Intrusion Specification Language (CISL) to represent intrusion detection models (see [20] for details). Our preliminary experiments show that the model generator is able to produce and distribute new effective models upon receiving audit data.

## 6 Related Work

Our research encompasses many areas of intrusion detection, data mining, and machine learning. In this section, we briefly compare our approaches with related efforts.

In terms of feature construction for detection models, DC-1 (Detector Constructor) [9], first invokes a sequence of operations for constructing features (indicators) before constructing a cellular phone fraud detector (a classifier). We are faced with a more difficult problem here because there is no standard record format for connection or session records (we had to invent our own). We also need to construct temporal and statistical features not just for individual records, but also over different connections and services. That is, we are modeling different logical entities that take on different roles and whose behavior is recorded in great detail. Extracting these from a vast and overwhelming stream of data adds considerable complexity to the problem.

The work most similar to unsupervised model generation is a technique developed at SRI in the Emerald system

[15]. Emerald uses historical records to build normal detection models and compares distributions of new instances to historical distributions. Discrepancies between the distributions signify an intrusion. One problem with this approach is that intrusions present in the historical distributions may cause the system to not detect similar intrusions in unseen data.

Related to automatic model generation is adaptive intrusion detection. Teng et al. [33] perform adaptive real time anomaly detection by using inductively generated sequential patterns. Also relevant is Sobirey's work on adaptive intrusion detection using an expert system to collect data from audit sources [28].

Many different approaches to building anomaly detection models have been proposed. A survey and comparison of anomaly detection techniques is given in [34]. Stephanie Forrest presents an approach for modeling normal sequences using look ahead pairs [10] and contiguous sequences [13]. Helman and Bhangoo [12] present a statistical method to determine sequences which occur more frequently in intrusion data as opposed to normal data. Lee et al. [22, 21] uses a prediction model trained by a decision tree applied over the normal data. Ghosh and Schwartzbard [11] use neural networks to model normal data. Lane and Brodley [16, 17, 18] examine unlabeled data for anomaly detection by looking at user profiles and comparing the activity during an intrusion to the activity under normal use.

Cost-sensitive modeling is an active research area in the data mining and machine learning communities because of the demand from application domains such as medical diagnosis and fraud and intrusion detection. Several techniques have been proposed for building models optimized for given cost metrics. In our research we study the principles behind these general techniques and develop new approaches according to the cost models specific to IDSs.

In intrusion data representation, related work is the IETF Intrusion Detection Exchange Format project [14] and the CIDF effort [30].

## 7 Conclusion

In this paper, we have outlined the breadth of our research efforts to address important and challenging issues of accuracy, efficiency, and usability of real-time IDSs.

We have implemented feature extraction and construction algorithms for labeled audit data (i.e., when both normal and intrusion data sets are given) [19]. We are implementing algorithms for unlabeled data (which can be purely normal or possibly containing unknown intrusions).

We have developed several anomaly detection algorithms. In particular, we have completed the implementation of and extensive experimentation with "artificial anomaly generation" approaches. We are exploring the use of information-theoretic measures, i.e., entropy, conditional entropy, relative entropy, information gain, and information cost to capture intrinsic characteristics of normal data and use such measures to guide the process of building and evaluating anomaly detection models. We are also developing efficient approaches that use statistics on packet header values for network anomaly detection.

We studied the computational costs of features and models, and have implemented a multiple model based approach for building models that incurs minimal computational cost while maintaining accuracy. We have also developed a real-time system, "Judge," for evaluating models learned using this method.

We are developing adaptive learning algorithms to facilitate model construction and incremental updates. We are also developing unsupervised anomaly detection algorithms to reduce the reliance on labeled training data. We have completed the design and specification of our system architecture with sensor, detector, data warehouse, and modeler components. A prototype system has been implemented [20] and we will continue to build on and experiment with this system.

We are developing algorithms for data mining over the output of multiple sensors. This is strongly motivated by the fact that single sensors do not typically observe entire attack scenarios. By combining the information from multiple sensors we hope to improve detection accuracy.

The ultimate goal of our research is to not only demonstrate the advantages of our approaches but also provide useful architectures, algorithms, and tool sets to the community to build better IDSs in less time and with greater ease. Toward this end, we are integrating our feature construction and unsupervised anomaly detection algorithms into the model generator, and building detectors that are equipped with our misuse and anomaly detection algorithms. We are deploying our prototype IDS on real-world networks in order to improve our techniques.

A serious limitation of our current approaches (as well as with most existing IDSs) is that we only do intrusion detection at the network or system level. However, with the advent and rapid growth of e-Commerce (or e-Business) and e-Government (or digital government) applications, there is an urgent need to do intrusion and fraud detection at the application-level. This is because many attacks may focus on applications that have no effect on the underlying network or system activities. We have previously successfully developed data mining approaches for credit card fraud detection [2, 31, 32]. We plan to start research efforts on IDSs for e-Commerce and e-Government applications in the near future. We anticipate that we will be able to extend our current approaches to develop application-level IDSs because the system architecture and many of our data mining algorithms are generic (i.e., data format independent). For example, we can develop (and deploy) a sensor for a specific application, and extend the correlation algorithms, with application domain knowledge, in the detectors to combine evidences from the application and the underlying system in order to detect intrusion and frauds.

The relevant reports detailing the results described in this paper can be found at `http://www.csc.ncsu.edu/faculty/lee/project/id.html`, `http://www.cs.columbia.edu/ids`, and `http://www.cs.fit.edu/~pkc/id`.


## Acknowledgment

This research is supported in part by grants from DARPA (F30602-00-1-0603).


## References


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

[2] P. Chan, W. Fan, A. Prodromidis, and S. Stolfo. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems*, pages 67–74, Nov/Dec 1999.

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

[4] J. P. Egan. Signal detection theory and roc analysis. In *Series in Cognition and Perception*. Academic Press, New York, 1975.

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

[6] E. Eskin, M. Miller, Z.-D. Zhong, G. Yi, W.-A. Lee, and S. Stolfo. Adaptive model generation for intrusion detection. In *Proceedings of the ACMCCS Workshop on Intrusion Detection and Prevention*, Athens, Greece, 2000.

[7] W. Fan. *Cost-senstive, Scalable and Adaptive Learning Using Ensemble-based Methods*. PhD thesis, Columbia University, Feb 2001.

[8] W. Fan, W. Lee, S. Stolfo, and M. Miller. A multiple model approach for cost-sensitive intrusion detection. In *Proc. 2000 European Conference on Machine Learning*, Barcelona, Spain, May 2000.

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

[10] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *In 1996 IEEE Symposium on Security and Privacy*, pages 120–128. IEEE Computer Society, 1996.

[11] A. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *Proceedings of the Eighth USENIX Security Symposium*, 1999.

[12] P. Helman and J. Bhangoo. A stiatistically base system for prioritizing information exploration under uncertainty. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 27(4):449–466, 1997.

[13] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detect using sequences of system calls. *Journal of Computer Security*, 6:151–180, 1998.

[14] Internet Engineering Task Force. Intrusion detection exchange format. In *http://www.ietf.org/html.charters/idwg-charter.html*, 2000.

[15] H. S. Javitz and A. Valdes. The nides statistical component: description and justification. In *Technical Report, Computer Science Labratory, SRI International*, 1993.

[16] T. Lane and C. E. Brodley. Sequence matching and learning in anomaly detection for computer security. In *Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 43–49. Menlo Park, CA: AAAI Press, 1997.

[17] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 150–158, 1998.

[18] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2:295–331, 1999.

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

[20] W. Lee, R. Nimbalkar, K. Yee, S. Patil, P. Desai, T. Tran, and S. J. Stolfo. A data mining and CIDF based approach for detecting novel and distributed intrusions. In *Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection (RAID 2000)*, October 2000. to appear.

[21] W. Lee and S. J. Stolfo. Data mining approaches for intrusion detection. In *In Proceedings of the 1998 USENIX Security Symposium*, 1998.

[22] W. Lee, S. J. Stolfo, and P. K. Chan. Learning patterns from unix processes execution traces for intrusion detection. In *In AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56. AAAI Press, 1997.

[23] W. Lee, S. J. Stolfo, and K. Mok. Data mining in work flow environments: Experiences in intrusion detection. In *Proceedings of the 1999 Conference on Knowledge Discovery and Data Mining (KDD-99)*, 1999.

[24] W. Lee, S. J. Stolfo, and K. W. Mok. Algorithms for mining audit data. In T. Y. Lin, editor, *Granular Computing and Data Mining*. Springer-Verlag, 2000. to appear.

[25] 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.

[26] 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.

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

[28] M. Sobirey, B. Richter, and M. Konig. The intrusion detection system aid. architecture, and experiences in automated audit analysis. In *Proc. of the IFIP TC6 / TC11 International Conference on Communications and Multimedia Security*, pages 278 – 290, Essen, Germany, 1996.

[29] S. Stainford-Chen. Common intrusion detection framework. http://seclab.cs.ucdavis.edu/cidf.

[30] S. Staniford-Chen, B. Tung, and D. Schnackenberg. The common intrusion detection framework (cidf). In *Proceedings of the Information Survivability Workshop*, October 1998.

[31] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan. Cost-sensitive modeling for fraud and intrusion detection: Results from the JAM project. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, January 2000.

[32] S. J. Stolfo, A. L. Prodromidis, S. Tselepis, W. Lee, D. W. Fan, and P. K. Chan. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 74–81, Newport Beach, CA, August 1997. AAAI Press.

[33] H. S. Teng, K. Chen, and S. C. Lu. Adaptive real-time anomaly detection using inductively generated sequential patterns. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 278–284, Oakland CA, May 1990.

[34] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. In *In 1999 IEEE Symposium on Security and Privacy*, pages 133–145. IEEE Computer Society, 1999.