Adaptive Model Generation for Intrusion Detection Systems

Eleazar Eskin, Matthew Miller, Zhi-Da Zhong, George Yi,
Wei-Ang Lee, Salvatore Stolfo
{eeskin,mmiller,zz31,georgeyi,weiang,sal}@cs.columbia.edu
Department of Computer Science
Columbia University

1 Abstract

In this paper, we present *adaptive model generation*, a method for automatically building detection models for data-mining based intrusion detection systems. Using the same data collected by intrusion detection sensors, adaptive model generation builds detection models on the fly. This significantly reduces the deployment cost of an intrusion detection system because it does not require building a training set. We present a real time system architecture and efficient implementation of automatic model generation. The system uses a model building algorithm that builds anomaly detection models over noisy data. We evaluate the system using the DARPA Intrusion Detection Evaluation data and show an increase in detection performance as more data is collected by the sensors.

2 Introduction

Many recent approaches to intrusion detection (ID) have applied data mining techniques. These approaches build detection models by applying data-mining algorithms to large data sets of audit data collected by a system. These models have been empirically proven to be very effective [14, 21]. However, one major drawback of data mining based approaches is that the data required for training is very expensive to produce. In this paper, we present *adaptive model generation*, an

approach to intrusion detection which collects data and builds detection models on the fly.

Data mining IDSs collect data from *sensors* which monitor some aspect of a system. Sensors may monitor network activity, system calls used by user processes, or file system access. They extract predictive features from the raw data stream being monitored to produce formatted data that can be used for detection. Data gathered by sensors is evaluated by a *detector* using a detection model. This model determines whether or not the data is intrusive. Algorithms for building detection models are usually classified into two categories: *misuse detection* and *anomaly detection*. The nature of the training data sets for each category is different, but are expensive to produce in either case.

Misuse detection algorithms model known attack behavior. They compare sensor data to attack patterns learned from the training data. If the sensor data matches the pattern of some known attack data, the observed data is considered intrusive. Misuse models are typically obtained by training on a large set of data in which the attacks have been manually labeled [14]. This data is very expensive to produce because each piece of data must be labeled as either normal or some particular attack.

Anomaly detection algorithms model normal behavior. Anomaly detection models compare sensor data to normal patterns learned from the training data. If the sensor data deviates from normal behavior, the anomaly detection model classifies the data as to have originated from an attack. Anomaly detection models are popular because they are seen as a possible approach to detecting unknown or new attacks [1, 3, 21, 4]. Most of these algorithms require that the data used for training is purely normal and does not contain any attacks. This data can be very expensive because the process of manually cleaning the data is quite time consuming. Also, some algorithms require a very large amount of normal data which increases the cost.

Models trained on data gathered from one environment may not perform well in some other environment. This means that in order to obtain the best intrusion detection models, data must be collected from each environment in which the intrusion detection system is to be deployed. The cost of generating data sets can be very expensive and the cost incurred is a significant barrier to IDS deployment.

To address these issues, we hope to develop a system that could collect its own data and train itself. The central idea is to have the system that collects the data to detect intrusions, also collect the data used in training detection models.

In this paper we present an IDS that performs *adaptive model generation*. This system streamlines and automates the process of collecting data from sen-

sors, building models, and distributing those models to detectors. The system takes advantage of a new anomaly detection algorithm which can build an effective model over noisy data [2]. This algorithm can build detection models while tolerating a small amount of intrusive data mixed in with normal data. In addition to building anomaly detection models in a completely automatic fashion, the system also provides mechanisms to significantly improve the ability for users to quickly and easily build data sets and models for misuse detection and distribute them to detectors.

We evaluate our adaptive anomaly detection system using data from the DARPA Intrusion Detection Evaluation. We show that the performance of the models improves as the system collects more training data.

2.1 Related Work

The work most similar to adaptive model generation is a technique developed at SRI in the Emerald system [8]. 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. [20] 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 [17].

Many different approaches to building anomaly detection models have been proposed. A survey and comparison of anomaly detection techniques is given in [21]. Stephanie Forrest presents an approach for modeling normal sequences using look ahead pairs [3] and contiguous sequences [6]. Helman and Bhangoo [5] present a statistical method to determine sequences which occur more frequently in intrusion data as opposed to normal data. Lee et al. [13, 12] uses a prediction model trained by a decision tree applied over the normal data. Ghosh and Schwartzbard [4] use neural networks to model normal data. Lane and Brodley [9, 10, 11] examine unlabeled data for anomaly detection by looking at user profiles and comparing the activity during an intrusion to the activity under normal use.

In intrusion data representation, related work is the IETF Intrusion Detection Exchange Format project [7] and the CIDF effort [18].

3 Adaptive Model Generation

We propose an IDS architecture that contains three components, a sensor, a detector, and an adaptive model generator. The sensor feeds formatted data to the detector for analyzing and responding to occurring intrusions, and also sends data to the adaptive model generator for learning new detection models. The adaptive model generator, upon learning a new model, feeds this model to the detector. This architecture is shown in Figure 1.

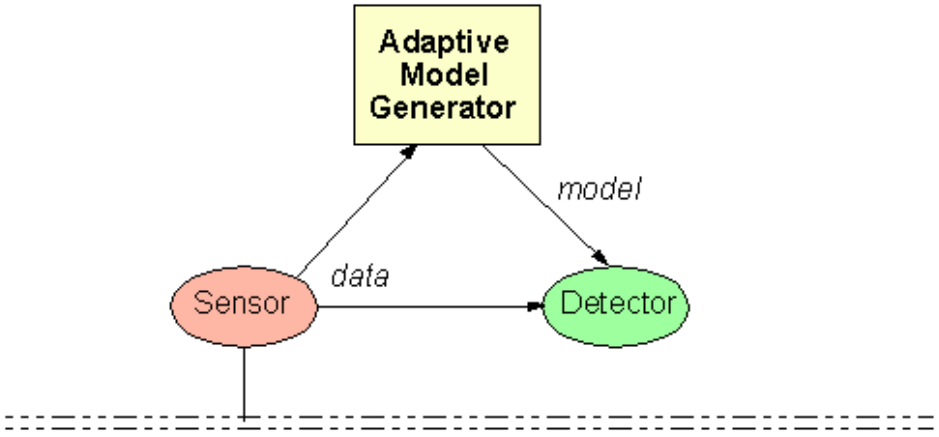

Figure 1. Adaptive Model Generation Overview

The adaptive model generation system consists of 4 components: the data receiver, the data warehouse, the model generators, and the model distributor. The data receiver collects the data from a sensor and converts it into a form to be ready to insert into the data warehouse. The data warehouse component stores the data in a database. Training sets can be generated for the model generation components using database queries. The model generation components build models using training sets obtained from the data warehouse component. The models are then distributed to the detectors using the model distributor component. The adaptive model generation architecture is shown in Figure 2.

3.1 Model and Data Representation

This general framework can utilize any sensor or model type. The representation of the data collected by the sensor is particular to the type of raw data that is

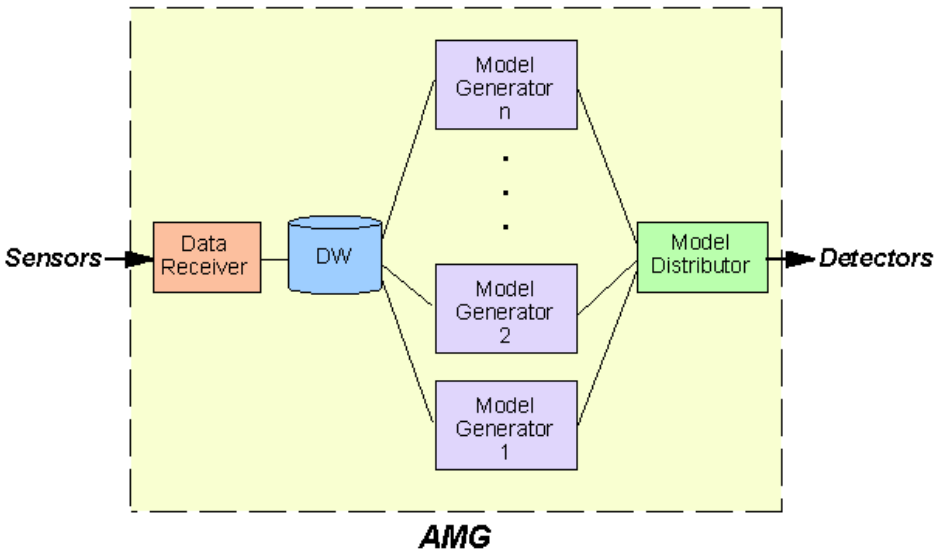

Figure 2. Adaptive Model Generation Architecture

being monitored. Detection can be done using any model evaluation engine and the model may be in the form of a neural network, a set of rules, or in our case, a probabilistic model. An adaptive model generation system must have a robust mechanism for dealing with heterogeneous data and model representations. For this reason we use XML for representation of both the data and the models. The XML encodes all of the information necessary for the entry of the information into the database. This information includes the names of the features of the actual data and the identification information of the source of the data such as the machine the data came from as well as information such as the time stamp of the data.

3.2 Data Warehousing

The main advantage of storing the sensor data in a database is the ability to retrieve an arbitrary subset of the data with a single query. Since different IDS model building algorithms have different properties, a database is useful because it is easy to manipulate what data which will compose the training set. For example, if we are building detection models based on system call data, it has been shown that effective models can be built by modeling the system calls for one program at a time [3]. If all of the system call data is stored in the database, we can retrieve

all of the system call data for a given process with a single query.

3.3 Model Generation and Distribution

The adaptive model generation system is designed to work with any model generation algorithm and the model generator components may be viewed as a black box. These components take the training set as input and output a learned model. In our case, we use a probabilistic model generation system, but in fact we could use any model. The model generation algorithm must be able to handle training data that may contain attacks. We present such an algorithm below. The architecture is designed to support many different types of model generation systems.

After the model generation components have learned their models, the model distributor sends the models to the detectors. The detectors are designed with the capability to change their models on the fly. This approach is motivated by the work of Stolfo et al. on the JAM architecture used for real time fraud detection in the network-based financial system [19].

3.4 Efficiency Considerations

One important consideration in the design of intrusion detection systems is efficiency. The IDS must be efficient enough to allow for timely detection of attacks and should also minimize its use of the resources of the system it is protecting. The latter is especially true in host-based detection systems. In the case of a host-based detection system, the IDS is taking away computational resources directly from the system it is trying to protect. The adaptive model generation framework is designed to be a distributed system. The model generation is done on a separate system to minimize the resources used by the system protected. In addition, the detector could be easily moved off the system as well further minimizing the load on the protected system. In this case only the sensor would be taking resources away from the system being protected.

4 Anomaly Detection over Noisy Data

In order to build models automatically from sensor data, we apply an algorithm for building anomaly detection models over noisy data. The reason the algorithm must be able to handle noise in the data is because we can not guarantee that the

data collected from the sensors is necessarily clean. We present a probability-based algorithm, but any algorithm that can tolerate a certain amount of noise can be substituted into the system.

An assumption of our approach is that the normal use of the system being monitored greatly outnumbers the occurrence of intrusions. This means that the attacks compose a relatively small proportion of the total data. 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 create anomaly detection models by first detecting the attacks that are buried within the training set. After removing those elements, we build an anomaly detection model over the remaining clean data.

We use a probabilistic approach to detect anomalies within a data set. We use a mixture model technique to model the presence of the anomalies. We then train a machine learning method over the data set to obtain a probability distribution of the data.

Since the number of anomalies is very small, we first assume that every element is normal. Motivated by our model of anomalies, we use the probability distribution to test each element to determine whether or not it is an anomaly.

4.1 Mixture Model

The mixture model assumes that there are two cases for each piece of data. We assume that with probability $(1 - \lambda)$ a given piece of data is a legitimate use of the system, while with probability λ the data corresponds to an intrusion. This motivates a mixture model for explaining the presence of anomalies in the data set.

In this framework, there are two probability distributions which generate the data, a *majority* distribution and an *anomalous* distribution. An element x_i is either generated from the majority distribution, $\mathbf{M}$, or with probability λ from the alternate distribution, $\mathbf{A}$. Our generative distribution for the entire data, $\mathbf{D}$, is then:

$$\mathbf{D} = (1 - \lambda)\mathbf{M} + \lambda\mathbf{A} \tag{1}$$

The mixture framework for explaining the presence of anomalies in the data is independent of the properties or types of the distributions $\mathbf{M}$ and $\mathbf{A}$.

The set of elements D, generated by distribution, $\mathbf{D}$, is partitioned into two subsets, M and A, corresponding to which elements were generated by distribu-

tion $\mathbf{M}$ and which elements were generated by distribution $\mathbf{A}$. We use the notation M_t (A_t) to denote the set of normal (anomalous) elements after processing element x_t. Initially, we have not detected any anomalies so the set of majority elements is the entire data set ($M_0 = D$) and the set of anomaly elements is empty ($A_0 = \emptyset$).

4.2 Detecting Anomalies

Detecting anomalies, in this framework, is equivalent to determining which elements were generated by the distribution $\mathbf{A}$ and which elements were generated by the distribution $\mathbf{M}$. Elements generated by $\mathbf{A}$ are anomalies, while elements generated by $\mathbf{M}$ are not.

For each element x_t we determine whether it is an anomaly and should be moved to A_{t+1} or is not an anomaly and should remain in M_{t+1}.

In order to make this determination, we examine the likelihood of the two cases.

The likelihood, L, of the distribution $\mathbf{D}$ at time t is:

$$L_t(\mathbf{D}) = \prod_{i=1}^{N} P_D(x_i) = \left((1-\lambda)^{|M_t|} \prod_{x_i \in M_t} P_{M_t}(x_i) \right) \left(\lambda^{|A_t|} \prod_{x_j \in A_t} P_{A_t}(x_j) \right)$$

where P_{M_t} and P_{A_t} are the probability distributions over the majority and anomalous data respectively. We compute whether an element is more likely to be an anomaly or a normal element.

In other words, we examine the change in L_t if:

$$M_t = M_{t-1} \setminus \{x_t\} \tag{2}$$

$$A_t = A_{t-1} \cup \{x_t\} \tag{3}$$

If this ratio (L_t / L_{t-1}) is greater than some value c, we declare the element an anomaly and permanently move the element from the majority set to the anomaly set. Otherwise, the element remains in the normal distribution.

$$M_t = M_{t-1} \tag{4}$$

$$A_t = A_{t-1} \tag{5}$$

c is a parameter to the method which sets the sensitivity to anomalies. We repeat this process for every element and in the end we get a partition of data set into a set of majority elements and a set of anomalous elements.

A complete description of this algorithm as well as performance comparisons to other anomaly detection algorithms is given in [2].

Table 1: Lincoln Labs System Call Data Summary

Program Name	# Intrusion Traces	# Intrusion System Calls	# Normal Traces	# Normal System Calls	% Intrusion System Calls
ftpd	1	350	943	66842	0.05%
ps	21	996	208	35092	2.7%
eject	6	726	7	1278	36.3%

5 Evaluation

We applied adaptive model generation to detect intrusions based on the analysis of process system calls. We examined a set of system call data that contains intrusions.

The data analyzed is a set of system call *traces* for a given program. A trace is the history (or list) of system calls made by a process of the given program from beginning of execution to the termination of the process. The arguments to the system calls are ignored for the analysis. In the data set, there is a set of clean traces and a set of intrusion traces.

The attacks that are contained in the data set are "user to root" attacks. These attacks target programs on UNIX machines that run with superuser privileges. By exploiting bugs in the programs, an attacker can gain superuser privileges by causing the programs to execute a shell. This shell allows the attacker complete access to the system.

The set of system calls that are observed under the normal usage of the system are significantly different than the set of system calls that are observed when an attacker is exploiting bugs in the program and obtaining a shell. This difference can be detected and is the basis of anomaly detection methods on system calls.

The data set is from the BSM (Basic Security Module) data portion of the 1999 DARPA Intrusion Detection Evaluation data created by MIT Lincoln Labs [15]. The data consists of 5 weeks of BSM data of all processes run on a Solaris machine. We applied adaptive model generation to build models over three weeks of traces of the programs which were attacked during that time. The programs attacked were: *eject*, *ps*, and *ftp*. Table 1 gives a summary of the data set.

Using the system call traces, we decompose each trace into a set of system call records. Each record is a sequence of five consecutive system calls obtained by sliding a window over the traces. Our probability model computes the likelihood

of each of the sequences. The likelihood of a sequence is the likelihood of the fifth system call given the previous four. This measures how well the first four system calls predict the last system call. Intuitively, since the attack processes take a different execution path than the normal system calls, the point at which the paths diverge would have very low likelihood in this model. This model is equivalent to computing the Markov chain probability:

$$P(x_5|x_4x_3x_2x_1) \tag{6}$$

where x_i is the i-th system call in the record. Training a Markov chain consists of computing the counts of system calls with their contexts.

After applying the method described in section 4, we can remove the anomalous elements and train the system only on the normal data. To evaluate the model, we compute the likelihood for each record in a process trace. If the likelihood is below some threshold, then we declare the process an intrusion.

Using the data to simulate a host based intrusion detection system, we apply adaptive model generation. We build a model after collecting a weeks worth of data. Week 1's model is trained on the first week of data. Week 2's model is trained on the first two weeks of data, while week 3's model is trained on all three weeks of data.

We evaluate the model after processing each week of data. To keep the evaluation from being biased to when the attacks occurred, we created a test set which contains attack and normal data. The same test set was used to evaluate each model. We also spread the attack data evenly between the three weeks of training data to further remove bias. As expected, the detection performance improves as more training data is collected.

There are two important measurements in evaluating the performance of an intrusion detection system, the *detection rate* and the *false positive rate*. The detection rate is the percentage of attacks present in the data that a system detected. The false positive rate is the percentage of normal data which the system claims to be attacks. Performance is optimized when the detection rate is maximized and the false positive rate is minimized. Typically there is a tradeoff between the detection rate and the false positive rate. We can examine this tradeoff by varying the detection threshold.

By plotting the detection rate versus the false positive rate at different thresholds, we obtain an ROC curve. An effective method for comparing models is by comparing their ROC curves on the same data [16]. Figure 3 shows the ROC curves for three models (Week1, Week2, Week3) over the three programs that

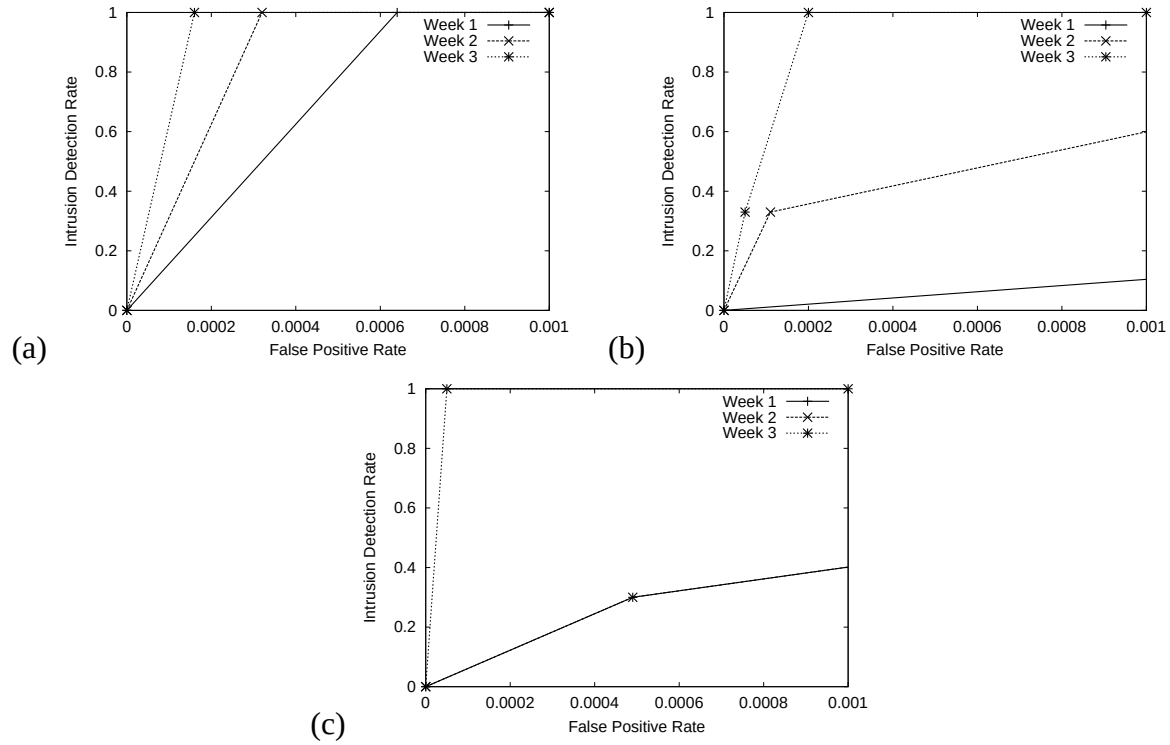

Figure 3: The performance of adaptive model generation over 3 weeks of system call data. ROC curves show the performance of the system by plotting the Intrusion Detection rate versus the False Positive rate. As expected, model performance improves from the models generated after 1 week of training data to the models generated after 3 weeks of training data. (a) ftpd. (b) ps. (c) eject.

contained attack data in the dataset. Notice that the ROC curves increase as the system collects more data which signifies an improvement in model performance.

6 Conclusion

In this paper we presented adaptive model generation, a method for automatically building intrusion detection models. The method uses the data collected by the sensors to train intrusion detection models and distribute the models to detectors. To build the models, adaptive model generation uses an algorithm that builds anomaly detection models over noisy data. The detection models are periodically

updated by the system automatically as more data is collected. Adaptive model generation may significantly reduce the cost of deploying an IDS system because it removes the need for manually creating a training set.

Future work includes extending the data warehousing component to allow for the combining of information from different sensors. Each sensor only sees a portion of what is happening on a system. Since all of the data is stored in the data warehousing component, the data can easily be combined to a more complete view of what is happening on a system. This can be an approach for allowing the system to build more effective models.

We can also extend the data representation to take advantage of linking capabilities of (or associated with) XML such as links among models and the data sets used to generate them.

Another direction for future work would be to use the infrastructure provided by the system to assist in building misuse detection training sets. The data can be collected from the sensors and stored in the database in the same way. However, during the data collection process, the system would be attacked and the time stamp of the attacks as well as their target would be recorded. The data of when the attacks occurred can be inserted directly into the database. The actual training set can be generated from the database. This approach can potentially reduce the cost of building a training set.

References

[1] D.E. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, SE-13:222–232, 1987.

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

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

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

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

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

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

[8] H. S. Javitz and A. Valdes. The nides statistical component: Description and justification. Technical report, SRI International, 1993.

[9] 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.

[10] 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.

[11] 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.

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

[13] W. Lee, S. J. Stolfo, and P. K. Chan. Learning patterns from unix processes execution traces for intrusion detection. In *Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56. Menlo Park, CA: AAAI Press, 1997.

[14] 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.

[15] MIT Lincoln Labs. 1999 DARPA intrusion detection evaluation In *http://www.ll.mit.edu/IST/ideval/index.html*, 1999.

[16] Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*, July 1998.

[17] 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.

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

[19] Salvatore Stolfo, Andreas L. Prodromidis, Shelley Tselepis, Wenke Lee, Wei Fan, and Philip Chan. JAM: Java agent for meta learning over distributed databases. In *Proceedings of the Third International Conference in Knowledge Discovery and Data Mining (KDD'97)*, 1997.

[20] 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.

[21] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 133–145. IEEE Computer Society, 1999.