Adaptive Model Generation: An Architecture for Deployment of Data Mining-based Intrusion Detection Systems

Andrew Honig, Andrew Howard, Eleazar Eskin, Sal Stolfo
Department of Computer Science
Columbia University
New York, NY 10027
{arh,ahoward,eeskin,sal}@cs.columbia.edu

Abstract

Data mining-based intrusion detection systems (IDSs) have significant advantages over signature-based IDSs since they are designed to generalize models of network audit data to detect new attacks. However, data mining-based IDSs are difficult to deploy in practice due to the complexity of collecting and managing audit data to train the data mining-based detection models. In this paper, we present Adaptive Model Generation (AMG), a real time architecture for implementing data mining-based intrusion detection systems. This architecture solves the problems associated with data mining-based IDSs by automating the collection of data, the generation and deployment of detection models, and the real-time evaluation of data. It is a distributed system with general classes of components that can be easily changed within the framework of the system. We also present specific examples of system components including auditing sub-systems, model generators for misuse detection and anomaly detection, and support for visualization and correlation of multiple audit sources.

1 Introduction

As sensitive information is increasingly being stored and manipulated on networked systems, the security of these networks and systems has become an extremely important issue. Intrusion detection systems (IDSs) are an integral part of any complete security package of a modern, well managed network system. An IDS detects intrusions by monitoring a network or system and analyzing an audit stream collected from the network or system to look for clues of malicious behavior.

The most widely used and commercially available IDSs are signature-based systems. A signature-based system matches features observed from the audit stream to a set of signatures hand crafted by experts and stored in a signature database. Signature-based methods have some inherent limitations. The most significant is that a signature-based method is designed to only detect attacks for which it contains a signature in the database. In addition to the expense in time and human expertise of manually encoding a signature for each and every known attack, the signature-based methods therefore can not detect unknown attacks since there is no signature in the database for them. It is these unknown attacks that are typically the most dangerous because the system is completely vulnerable to them.

Data mining-based methods are another paradigm for building intrusion detection systems. The main advantage of these methods is that they leverage the generalization ability of data mining methods and in order to detect new and unknown attacks. A data mining-based IDS uses machine learning and data mining algorithms on a large set of system audit data to build detection models.

These models have been proven to be very effective [17, 26]. These algorithms are generally classified as either misuse detection or anomaly detection. Misuse detection algorithms learn how to classify normal and attack data from a set of training data which contains both labeled attack and normal data [17]. Anomaly detection algorithms learn a model of normal activity by training on a set of normal data. Anomaly detection algorithms then classify as an attack activity that diverges from this normal pattern based on the assumption that attacks have much different patterns than do normal activity. In this way new unknown attacks can be detected [4, 9, 26, 10].

However, data mining-based IDSs have their own disadvantages. Data to train the models is costly to generate. The data must be collected from a raw audit stream and translated into a form suitable for training. In addition, for misuse detection, each instance of data must be labeled either normal or attack. In the case of anomaly detection each instance of data must be verified to be normal network activity. Since data mining-based IDSs in general do not perform well when trained in one environment and deployed in another, this process of preparing the data must be repeated at every deployment of data mining-based IDS system. Furthermore, for each type of audit data that is to be examined (network packets, host event logs, process traces, etc.) the process of preparing the data needs to be repeated as well. Because of the large volumes of data that needs to be prepared, the deployment of a data mining-based IDS system involves a tremendous amount of manual effort.

Many of parts of these manual processes can be automated, including the collection and aggregation of the data and translating it into a form appropriate for training the data mining-based detection models. In addition, many of these processes are the same across types of audit data. Some of the processes still require some manual intervention such as labeling the data, but even these can be semi-automated.

In this paper we present Adaptive Model Generation (AMG), an architecture to automate the processes of data collection, model generation and data analysis. The AMG system solves many of the practical problems associated with the deployment of data mining-based IDSs.

AMG is an architecture to automate many of the critical processes in the deployment and operation of real time data mining-based intrusion detection systems. AMG abstracts various parts of the data mining-based IDS process and formalizes the automation of this process. This includes abstracting the processes of data collection, data aggregation, detection model evaluation, detection model generation, and model distribution. AMG uses a general XML-based data and model representation scheme that facilitates the translation of data from what is collected at the audit stream to the form necessary for generation of detection models. AMG provides a tool for semi-automating the process of labeling training data and also leverages a new class of intrusion detection algorithms called unsupervised anomaly detection (UAD) algorithms which permits the training of detection models over unlabeled data. These algorithms detect intrusions buried within unlabeled data and can tolerate a small amount of intrusion data (noise) in the training data [6, 20, 7].

The AMG architecture consists of several different types of component sub-systems. Real time components such as sensors and detectors collect information from the monitored system and detect intrusions in real time. The centerpiece of the data management capabilities of AMG is a data warehouse which stores the data collected by all of the sensors in a network. Model generators access this data and train data mining-based detection models using this data. Model distributors transfer the models to the detectors. Finally analysis engines provide data analysis capabilities such as visualization and forensic analysis.

More specifically, AMG has the following major capabilities:

- Automated Data Collection: Data is collected by sensors and automatically sent to detectors for classification and to the data warehouse for aggregation and use in training models.

- Data Warehousing: AMG includes a data warehouse component that stores data from all sensors. This data is used for training detection models but can used to support various types of data analysis such as forensic analysis.
- Automated Model Generation: Detection models are trained from data stored in the data warehouse. The process for converting the data into the appropriate form for training is fully automated.
- Heterogeneous Data Support: Data from different types of audit sources is easily handled in the framework of our system. Any form of data can be represented using a general XML-based language.
- Automated Model Distribution: Once a new model is generated it is deployed to all of the detectors that subscribe to the particular detection models that have been generated.
- Data Analysis Capabilities (Including Forensics): AMG enables evaluation of archival records stored within the data warehouse to search for intrusions.
- Visualization Support: AMG includes generic visualization components that allow in a consistent fashion the visualization of data from different sources.
- Correlation Support: Since data from multiple sensors is stored in the data warehouse AMG can perform analysis over the data from multiple sources and to train detection models which examine audit streams from multiple sensors.

The paper is organized as follows. We first describe in detail the components that make up the AMG system. We then describe the main capabilities of the AMG system. Then we describe the kinds of data mining algorithms that the AMG system supports. Finally we give examples of model generation algorithms and sensors that are implemented through the AMG framework. Note that in most cases, the sensors and detectors presented in this paper have been described in detail in other prior publications. In this paper we present the details of how the sensor or model is easily integrated into the AMG system in a "plug and play" fashion.

2 Components of Adaptive Model Generation

The adaptive model generation system automates the processes of data collection, data management, detection model generation and distribution, and provides various capabilities for data analysis. The challenge in automating these processes is the need to support different types of data and different types of detection models. In a typical network environment there are many different audit streams that are useful for detecting intrusions. For example, such data includes the packets passing over the network (header values, payload features, or combinations thereof), the system logs on the hosts in the network, and the system calls of processes on these machines. These types of data have fundamentally different properties. In addition, detection models can also vary greatly. The most widely used detection model is a signature-based system while data mining-based approaches use methods such as probabilistic models [8] and support vector machines [3]. The methods for building these detection models as well as executing them in real time vary greatly for each type of detection model.

AMG is a system framework and architecture that can handle virtually any data type and detection model. The AMG system consists of four types of components: real time components which include sensors and detectors, a data warehouse component, detection model management components which include adaptive model generators and detection model distributors and data analysis components which includes components for visualization, forensics, labeling data, correlation and extracting information from multiple records.

Sensors gather information from an environment and send that data to the data warehouse. The data in the data warehouse is accessed by the detection model generators which generate models that classify activity as either malicious or normal. Once a model is created it is stored in the data warehouse. A model distributor deploys that model to a real-time detector. The detector receives the detection model from the detection model distributor and also receives the audit data from the sensor. It then uses the model to evaluate the audit data from the sensor to detect intrusions.

The data analysis engines retrieve data from the data warehouse. The use of the data varies depending on the particular data analysis engine. The results of an analysis are either stored in the data warehouse, or displayed directly to the user. Data analysis engines allow the adaptive model generation system to implement many systems that are helpful in the deployment of an intrusion detection system. New types of data analysis engines can easily be incorporated into the AMG system. The complete architecture of the adaptive model generation system is displayed in Figure 1.

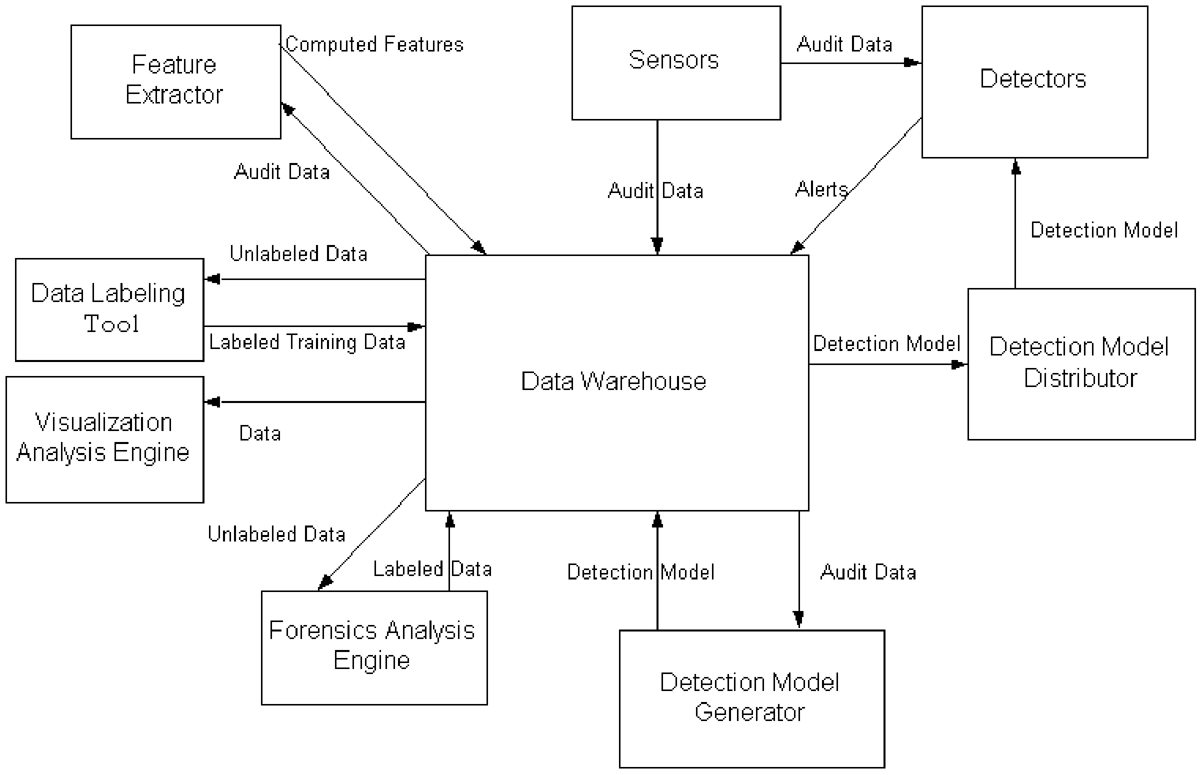


Figure 1: The AMG System Architecture

The adaptive model generation system uses a distributed architecture that consists of autonomous components. By making these components independent, linking them only with a communication protocol with very loose format restrictions, we allow any component conforming to that protocol to interface with our system. The communication protocol uses an XML markup language which is similar to the IDMEF specification in the IETF [13]. The system can easily be adapted to the IDMEF format or any other language format such as CIDF [23].

2.1 Real Time Components

The AMG system uses two basic components to provide real time detection capabilities: sensors and detectors. A sensor is a system that collects data from an audit stream and formats the data into the appropriate form using the XML markup language. This data is then sent to a detector which uses a detection model to determine whether or not the data corresponds to an attack. If the data corresponds to an attack, the detector generates an alert.

In this framework, a traditional IDS system consists of a sensor and a detector which in most cases are integrated together. For example, in our framework, the Snort [21] system contains a sensor which monitors packets and a detector which evaluates signatures over the features extracted from the packets. Our system separates these two components providing a means of managing the overall computation of IDS in reconfigurable architectures. For example, if the detection model evaluation for a particular audit stream is a computational bottleneck, it can be easily distributed to multiple machines.

The advantage of the AMG system over traditional system architectures is the integration of the sensors and detectors with other analysis and distribution sub-systems. In addition to sending the data to a detector, the sensor also sends the data to a data warehouse which aggregates and stores the data. The detector retrieves its detection models from the model distributors. These detection models are created by other components of the AMG system.

Sensors

Sensors are lightweight processes that gather information from a real-time audit stream, format that information into an XML representation and then send that to the detectors for real time intrusion detection and to the data warehouse for aggregation of data and storage. The sensors can gather information from any source. For every different audit stream of data a sensor can be implemented to monitor and collect the data.

Typically, there are two major classes of sensors, network sensors and host sensors. Network sensors monitor the packets moving through the network. We have implemented a network sensor HAUNT[5] which is used to monitor network activity. The system listens to network traffic and reassembles the packets in order to create data records which summarize connections. A network sensor in AMG can be created from existing network IDS systems such as Snort [21] and Bro [19] simply by wrapping the component in the XML-based form required for AMG.

In most circumstances there are multiple sensors for each host. This is because a host usually has several different streams of information that are useful for detecting intrusions. AMG has been implemented with a variety of sensors for both Windows and Linux systems [11]. On the Windows system we have developed Windows event log sensors, software wrappers, netstat sensors, and registry sensors. There are three Windows event log sensors which take information from the application, security, and event logs on the Windows system respectively. Software wrappers are sensors that gather information about system calls. Netstat sensors use the netstat tool that gathers information about network connections on the host. data. Registry sensors monitor the activity of the windows registry when applications are run on the host.

The Linux sensors built into AMG include process sensors, network connection sensors, resource sensors, and software wrappers. Process sensors use the `/proc` virtual file-system to gather information about each running process. Network connection sensors collect information about network connections being made to and from the host machine. Resource sensors gather information about CPU and memory usage on a machine. The software wrappers for Linux systems are the same as for Windows, monitoring system calls made by a process. Details of all the host based sensors can be found in [11].

The sensors themselves are constructed from two major components, the basic auditing module (BAM) and the communication engine. The BAM is the component that extracts the information from the audited source. The communication engine encodes the data and sends it to the data warehouse for storage.

The BAM needs a mechanism to gather the data. This is done differently for each stream of data. Therefore a separate BAM is needed for each source of data that the system monitors. Packet sniffers and Win32 hooks are two examples of ways to tap into the data stream and gather data. The BAM can be seen as an interface to the system being monitored. It hooks into the data stream and has the ability to transfer the information from that data stream to the communication engine. Therefore this system can function without any knowledge of how the sensor work. This makes the system very flexible with respect to sensors.

The communication engine takes the data from the BAM, encodes the data into the AMG XML format and then sends that data to the data warehouse. Along with the data itself the BAM sends the meta data, such as variable types, to the communication engine. This is the information that the communication engine needs to encode the information. The communication engine also needs to know the type of sensor in order to send the data to the right place in the data warehouse. This is specified when the connection is made to the communication engine. An example of a record being generated from the RAD sensor, a sensor that monitors accesses to the Windows registry, can be seen below.

Raw data being read by sensor:
```
Process: IEXPLORE
Key: HKCR\Applications\notepad.exe\shell
Query: Openkey
Result: Success
Response: 0xE22FC4C0
```
The sensor sends the following Data sent to Communication Engine:
```
Process: IEXPLORE
Key: HKCR\Applications\notepad.exe\shell
Query: Openkey
Result: Success
Response: 0xE22FC4C0
Time: Tue Jul 31 14:43:20 EDT 2001
ProcQuery: 1263.4353
KeyProc: 6784.9363
QueryKey: 6621.3521
KeyResponse: 4510.2431
KeyResVal: 8743.3245
```

Note that the combination features are stored as hashes, not the actual values. This is done for efficiency and convenience purposes and implemented in the sensor. The communication engine then encodes the record as the following.

```
<rec>
<process> IEXPLORE </process>
<key> HKCR\Applications\notepad.exe\shell </key>
<query> Openkey </query>
<result> Success </result>
<response> 0xE22FC4C0 </response>
<procQuery> 1263.4353 </procQuery>
<keyProc> 6784.9363 <keyProc>
<queryKey> 6621.3521 </queryKey>
<keyResponse> 4510.2431 </keyResponse>
<keyResVal> 8743.3245 </keyResVal>
```

Detectors

Detectors analyze audit stream data collected from a sensor and detect intrusions by using a detection model. A detector performs *model evaluation* over each record from the sensor. The way a specific detector works depends on the type of model being evaluated. Each different model type has a different detector that implements model evaluation for that model type.

The detector can be viewed as a function that takes as input a data record and outputs an alert if the data is determined to correspond to an intrusion. An example of a detection model type is a signature-based model, which is the algorithm most widely used in commercial intrusion detection systems. A signature-based detection model simply contains a set of “signatures” which correspond to known attack types. Model evaluation consists of matching each signature in the model to a data record. If any of the signatures match, the detector generates an alert.

In our framework, more sophisticated model types can be used as well including data mining-based models that use decision trees, probabilistic models and support vector machines. In the case of a decision tree, the detection model would contain an encoding of a decision tree. The detector would take this model and evaluate the detection model on a record by following the relevant branches of the tree. In the case of a probabilistic model, the detection model would contain a parametrization of the probabilistic model and the detector would compute a probability associated with each record. In the case of a support vector machine, the model would contain a set of support vectors which correspond to a decision boundary in a high dimensional feature space. The detector would effectively map a record to this high dimensional feature space and computes which side of the decision boundary the record falls on to determine whether or not to generate an alert. We give more details on the support vector machine model generation and detection below.

Detectors receive detection models from model distributors which distribute models stored in the data warehouse originally created by model generators. The detectors receive real time updates from the model distributors. This keep the detection models updated as soon as new models are available. Below is an example of the a model that the detector for the RAD system uses to make a classification.

```
<model>
<type> RAD </type>
<target> registrydb </target>
<version> 2.11 </version>
<encode>
```

```
<feature> <name> process </name> <n> 52000 </n> <r> 31 </r>
<values> iexplore.exe, aim.exe, explore.exe, msaccess.exe, pinball.exe, .....
</values> </feature>
<feature> <name> keyval </name> <n> 52000 </n> <r> 1800 </r>
<values> HKLM, HKLM\Appications, ......  </values> </feature>
..........
</encode>
</model>
```

Note that the encoding of this model is explained later in section 2.3. The evaluation of the record shown in the previous section with this model would result in a normal label for the record.

2.2 Data Warehouse

The data warehouse is the centerpiece of the AMG system. It serves as the central storage repository for all of the data collected by the sensors. The model generators access the data in the data warehouse and create detection models using this data. The data warehouse also stores these detection models. The analysis engines also access the data stored in the data warehouse. These components give the AMG system visualization and forensics capabilities as described below.

The core of the data warehouse is a SQL database. This allows for easy manipulation of the data, critical for creating training data sets to build data mining-based detection models. Since we can retrieve an arbitrary subset of data using a SQL query, the data warehouse automates the tedious process of manually creating these data sets. This flexibility is very important in practical deployments of the system.

For example, if there are 40 Windows hosts in a network and we wish to create an anomaly detection model over the Application Event logs for each of the hosts in the AMG framework, we perform the following. We first install a sensor on each of the hosts. This will collect the data and store it in the data warehouse. If each host is typically used in the same way, we may want to create a large data set containing the combined event logs from each of the hosts. On the other hand, if each host is used differently, we may create a separate training set for each individual host. Since the data warehouse uses a SQL database, we can create these different data sets by issuing different queries.

Storing the data in a single repository has several other advantages for correlating sensor outputs. Since the data is stored in a SQL database, we can use "join" statements to facilitate the linking of records from different sensors together into single records. In addition, we can obtain data from two sensors relatively easily because all of the data is stored in the same database.

The data warehouse uses an XML markup language for communication with all of the modules. The communication is specified in a specific XML markup language defined for this purpose. This markup language was influenced by the IDMEF specification [13]. The format for an insert transaction is displayed below.

```
<command>
<tablename>
<begin>
<rec>
<var1 var1type> valueA </var1>
<var2 var2type> valueA </var2>
<var3 var3type> valueA </var3>
```

```
.........  <varN varNtype> valueA </varN>
</rec>
<rec>
<var1 var1type> valueB </var1>
<var2 var2type> valueB </var2>
<var3 var3type> valueB </var3>
.........  <varN varNtype> valueB </varN>
</rec>
<end>
```

The transaction begins with a <command> to direct the data warehouse operation appropriately. The name of the table to be operated upon is then provided via <tablename>, where the pertinent information is stored. Then the information is sent in XML format. The data starts with a <begin> tag. Each record is started with a <rec> tag. Within each record all of the data is sent for that record, variable by variable. The variable name is sent along with its type as the tag, and between the tag is the value for that variable. Any number of records can be sent at a given time using this protocol. This greatly reduces the cost in many cases when there are many records being sent to the database by a sensor. When the data warehouse decodes the XML format, it checks to see if each variable has a column in the table where the data is being inserted. If that column does not exist then it is created on the fly.

Below is a sample transaction. It is a single record being inserted into the nfr1 database by the HAUNT network sensor.

```
<insert>
<nfr1>
<begin>
<rec>
<ID i> 96 </ID>
<dst-bytes i> 490 </dst-bytes>
<rerror-rate f> 0.18786 </rerror-rate>
<sensor-rate f> 0.09760 </sensor-rate>
<src-bytes i> 1381 </src-bytes>
<src-count i> 151 </src-count>
<src-serror-rate f> 0.16265 </src-serror-rate>
<label str> normal </label>
<src str> 128.59.22.66 </src>
<dst str> 12.59.22.87 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>
<end>
```

The HAUNT sensor connects to the data warehouse to transfer a record. It begins by sending an insert command to let the data warehouse know that it wants to insert data. Then it specifies the table nfr1 where the data is to be stored. Then it opens its first record with an opening <rec> tag. Then in order each variable is sent to the data warehouse. First the ID of the transaction which is an integer is sent over and that is 96. Next the destination numbers of bytes, also an integer, is sent. Then each variable is sent sequentially until the entire record is sent to the data warehouse. For convenience we abbreviated the types int, float, and string with i, f, and str respectively.

2.3 Detection Model Management

The AMG system manages the creation and distribution of detection models. The detection models are generated using data collected stored in the data warehouse. They are distributed to the detectors by the model distributors.

Detection Model Generation

Our adaptive model generation system is designed to work with any model generation algorithm. Thus, the model generator components can be viewed as black boxes that are "plugged" into the architecture with relative ease. These components take the training set of data as input and output a model of malicious activity. Different types of model building algorithms require different types of data. In our architecture we allow the model generators to select any data that it wants through the use of general or specific queries. This means that the architecture is robust enough to handle any type of model generation algorithm.

The model generation modules request the data from the data warehouse when they want to create a model. They form their request based on the information that the model needs to train on. The generator then runs and creates a model. This model is then encoded into XML and sent to the data warehouse. Model generators also signal the model distributor to let it know that a new model is available. A sample XML encoding of a model generated by the RAD system is shown below.

```
<model> <type> RAD <type>
<target> registrydb </target>
<version> 2.11 </version>
<encode>
<feature> <name> process </name> <n> 52000 </n> <r> 31 </r>
<values> iexplore.exe, aim.exe, explore.exe, msaccess.exe, pinball.exe, .....
</values> </feature>
<feature> <name> keyval </name> <n> 52000 </n> <r> 1800 </r>
<values> HKLM, HKLM\Appications, ......  </values> </feature>
..........
</encode>
</model>
```

The model encoding begins with some meta-data about the model itself. The type field is used to notify the detector how to decode the rest of the model. The target specifies which table in the database this model applies to. The version information is used to coordinate with the model distributor in order to ensure that detectors have the most recent detection model. The model specifies information for evaluating the model follows the version information. This particular algorithm requires information and statistics about each feature in the data, and the values observed for that feature. This information is sent over one feature at a time. The encoding is specific to the type of model. All of the data between the `<encode>` and `</encode>` is specific to the model type, and needs to be defined for each new detection model generation algorithm. This flexibility is what allows the adaptive model generation system to work with any types of models.

Detection Model Distribution

Many model generation algorithms can be used in real-time environments. This creates the need for model distribution ensuring that all detectors have the most recent models. Detectors do not continuously check for updates in the data warehouse because this would be inefficient, and the real-time requirements of the system as a whole depends on the detectors being lightweight components. The model distributors are used to automatically send model updates to the detectors whenever the model generators create them.

2.4 Data Analysis Engines

An analysis engine is any component that takes as its input a set of data from the database and performs some kind of analysis over this data. The analysis engines have the capability of inserting the results of the analysis into the database. In our system the analysis engine queries the data warehouse for a set of data and then inserts new information into the data warehouse using the SQL interface. This can be useful for several purposes. The data analysis engine uses the same XML format that the rest of the system uses with some specific tags designed specifically for data analysis.

Currently, we have implemented four types of analysis engines: a visualization client, a forensics tool, a data labeling tool and a feature extractor.

Visualization Analysis Engine

The visualization analysis engine provides a system administrator with a mechanism to view all of the data in the data warehouse. An example of a visualization agent implemented by the adaptive model generation system is displayed in Figure 2.

The visualization analysis engine is integrated with the database which allows the use of SQL queries to filter the data to be viewed. An example of the interface and a SQL query is shown in Figure 3.

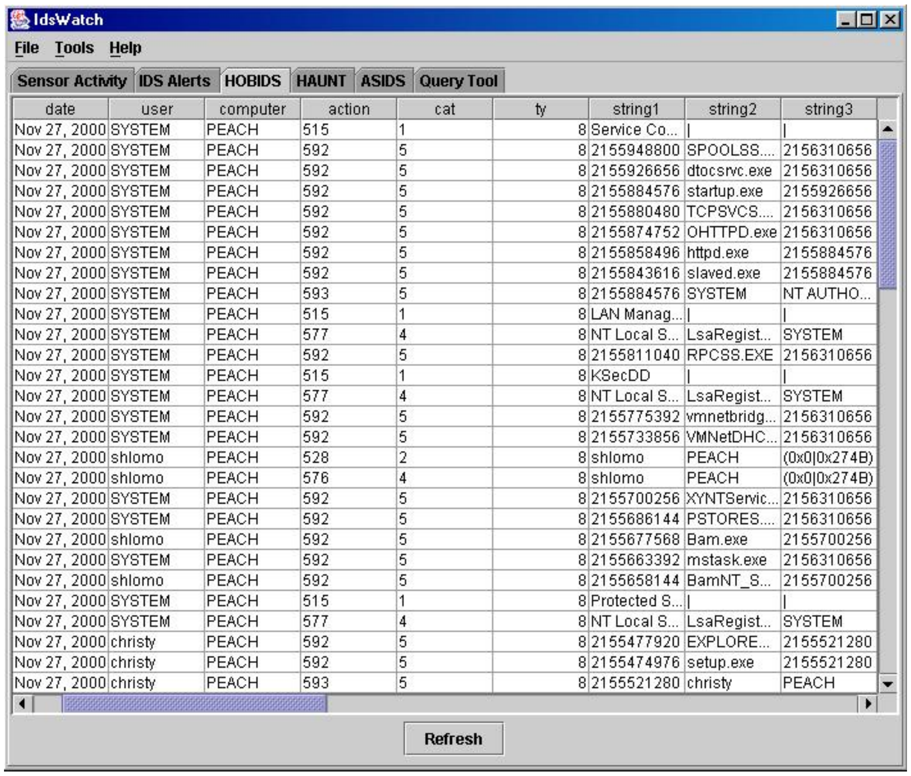
Forensics Analysis Engine

One of the more important types of data analysis is forensic analysis. A forensic system retrieves a set of historical data from the data warehouse, typically the data of interest is a set of data which we suspect contains intrusions. The tool must retrieve a specific set type of data appropriate to the algorithm in question. Once the data set is retrieved the forensics analysis engine can apply a detection algorithm algorithm to find suspicious activity in the data set. The suspicious activity is then labeled (either anomalous or normal) using SQL statements to mark the appropriate data. Note that this requires that a column be added to the table in the database in order to store the label. The data warehouse has the capability to do this on the fly.

A sample input and output of a forensics analysis tool being used on RAD data can be seen below.

Input data from the data warehouse:

```
<rec>
<ID i> 96 </ID>
<dst-bytes i> 490 </dst-bytes>
<rerror-rate f> 0.18786 </rerror-rate>
<sensor-rate f> 0.09760 </sensor-rate>
```

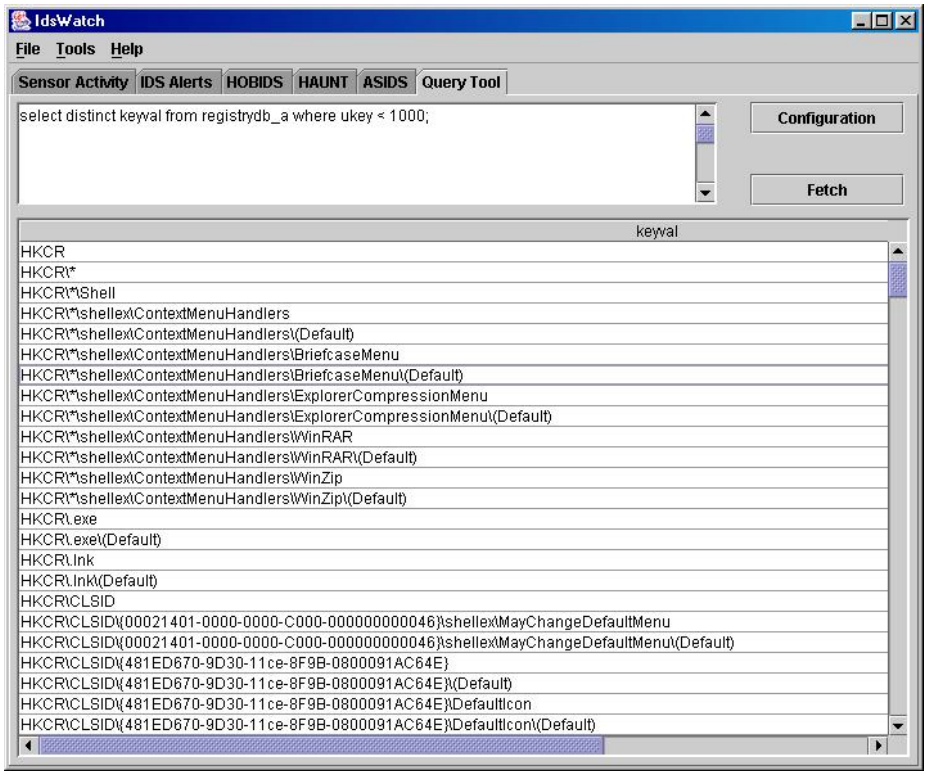


date	user	computer	action	cat	ty	string1	string2	string3
Nov 27, 2000	SYSTEM	PEACH	515	1	8	Service Co...	|	|
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155948800	SPOOLSS....	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155926656	dtocsrvc.exe	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155884576	startup.exe	2155926656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155880480	TCPSVCS....	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155874752	OHTTPD.exe	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155858496	httpd.exe	2155884576
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155843616	slaved.exe	2155884576
Nov 27, 2000	SYSTEM	PEACH	593	5	8	2155884576	SYSTEM	NT AUTHO...
Nov 27, 2000	SYSTEM	PEACH	515	1	8	LAN Manag...	|	|
Nov 27, 2000	SYSTEM	PEACH	577	4	8	NT Local S...	LsaRegist...	SYSTEM
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155811040	RPCSS.EXE	2156310656
Nov 27, 2000	SYSTEM	PEACH	515	1	8	KSecDD	|	|
Nov 27, 2000	SYSTEM	PEACH	577	4	8	NT Local S...	LsaRegist...	SYSTEM
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155775392	vmnetbridg...	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155733856	VMNetDHC...	2156310656
Nov 27, 2000	shlomo	PEACH	528	2	8	shlomo	PEACH	(0x0|0x274B)
Nov 27, 2000	shlomo	PEACH	576	4	8	shlomo	PEACH	(0x0|0x274B)
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155700256	XYNTServic...	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155686144	PSTORES....	2156310656
Nov 27, 2000	shlomo	PEACH	592	5	8	2155677568	Bam.exe	2155700256
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155663392	mstask.exe	2156310656
Nov 27, 2000	shlomo	PEACH	592	5	8	2155658144	BamNT_S...	2155700256
Nov 27, 2000	SYSTEM	PEACH	515	1	8	Protected S...	|	|
Nov 27, 2000	SYSTEM	PEACH	577	4	8	NT Local S...	LsaRegist...	SYSTEM
Nov 27, 2000	christy	PEACH	592	5	8	2155477920	EXPLORE...	2155521280
Nov 27, 2000	christy	PEACH	592	5	8	2155474976	setup.exe	2155521280
Nov 27, 2000	christy	PEACH	593	5	8	2155521280	christy	PEACH

Figure 2: Visualization of Data in Database

```
<src-bytes i> 1381 </src-bytes>
<src-count i> 151 </src-count>
<src-serror-rate f> 0.16265 </src-serror-rate>
<src str> 128.59.22.66 </src>
<dst str> 12.59.22.87 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>

<rec>
<ID i> 99 </ID>
<dst-bytes i> 420 </dst-bytes>
<rerror-rate f> 0.12786 </rerror-rate>
<sensor-rate f> 0.16760 </sensor-rate>
<src-bytes i> 1281 </src-bytes>
<src-count i> 132 </src-count>
<src-serror-rate f> 0.19325 </src-serror-rate>
<src str> 128.59.22.69 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>
.....
```



```
select distinct keyval from registrydb_a where ukey < 1000;
```

keyval
HKCR
HKCR*
HKCR*\Shell
HKCR*\shellex\ContextMenuHandlers
HKCR*\shellex\ContextMenuHandlers\(Default)
HKCR*\shellex\ContextMenuHandlers\BriefcaseMenu
HKCR*\shellex\ContextMenuHandlers\BriefcaseMenu\(Default)
HKCR*\shellex\ContextMenuHandlers\ExplorerCompressionMenu
HKCR*\shellex\ContextMenuHandlers\ExplorerCompressionMenu\(Default)
HKCR*\shellex\ContextMenuHandlers\WinRAR
HKCR*\shellex\ContextMenuHandlers\WinRAR\(Default)
HKCR*\shellex\ContextMenuHandlers\WinZip
HKCR*\shellex\ContextMenuHandlers\WinZip\(Default)
HKCR\\.exe
HKCR\\.exe\(Default)
HKCR\\.lnk
HKCR\\.lnk\(Default)
HKCR\CLSID
HKCR\CLSID\{00021401-0000-0000-C000-000000000046}\shellex\MayChangeDefaultMenu
HKCR\CLSID\{00021401-0000-0000-C000-000000000046}\shellex\MayChangeDefaultMenu\(Default)
HKCR\CLSID\{481ED670-9D30-11ce-8F9B-0800091AC64E}
HKCR\CLSID\{481ED670-9D30-11ce-8F9B-0800091AC64E}\(Default)
HKCR\CLSID\{481ED670-9D30-11ce-8F9B-0800091AC64E}\DefaultIcon
HKCR\CLSID\{481ED670-9D30-11ce-8F9B-0800091AC64E}\DefaultIcon\(Default)

Figure 3: Visualization of SQL Query

The output sent back to the data warehouse contains the same data with a label appended to the end. In this example, the first record was labeled as an attack and the second record was labeled as normal.

```
<rec>
<ID i> 96 </ID>
<dst-bytes i> 490 </dst-bytes>
<rerror-rate f> 0.18786 </rerror-rate>
<sensor-rate f> 0.09760 </sensor-rate>
<src-bytes i> 1381 </src-bytes>
<src-count i> 151 </src-count>
<src-serror-rate f> 0.16265 </src-serror-rate>
<src str> 128.59.22.66 </src>
<dst str> 12.59.22.87 </dst>
<ip-overlap str> 0 </ip-overlap>
<label str> attack </label>
</rec>

<rec>
<ID i> 99 </ID>
<dst-bytes i> 420 </dst-bytes>
<rerror-rate f> 0.12786 </rerror-rate>
```

```
<sensor-rate f> 0.16760 </sensor-rate>
<src-bytes i> 1281 </src-bytes>
<src-count i> 132 </src-count>
<src-serror-rate f> 0.19325 </src-serror-rate>
<src str> 128.59.22.69 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
<label str> normal </label>
</rec>
........
```

Data Labeling Tool

Another data analysis engine is the data labeling tool. The data labeling tool takes the list of known attacks and uses that information to label all of the records in the database which corresponds to these attacks. The labeling tool is used to create labeled training data. The list of known attacks could be process names, time stamps, or anything else that is also contained in the data records and can be matched to the known attacks. The labeling tool is a significant improvement over the difficult manual labeling of records in a database. The manual labeling of data is the greatest cost for deploying a data mining-based intrusion detection system. This cost is cut significantly through the use of this data labeling tool.

The data labeling tool is implemented using SQL joins with the sensor data in the data warehouse and the attack list. For example, let us assume we have a table full of Windows host based information from the application log. All actions in the application log are stored in the data warehouse with all available information from the log, including process name. Now assume that we have an attack list that is a list of all process names corresponding to attacks. We can automatically insert that attack list into the data warehouse in a temporary table. This temporary table could then be joined with the table of sensor data and the resulting table would be the sensor data labeled with its attack classification. This is a labeled set of training data that was created automatically from an attack list and a large set of sensor data. An example of the data labeling tool being used on the RAD data is seen below.

Input from the two tables in the data warehouse:

```
Raw data:
<rec> <process> iexplore.exe </process> <query> queryKay </query> ...
</rec>
<rec> <process> happy99.exe </process> </query> createKey </query> ...
</rec>
<rec> <process> outlook.exe </process> </query> openKey </query> ...
</rec>
.....
```

Attack List of process name:

```
<process> happy99.exe </process>
<process> bo2k.exe </process>
.....
```

Labeled Data:

```
<rec> <process> iexplore.exe </process> <query> queryKay </query> ...
<label> normal </label> </rec>
```

```
<rec> <process> happy99.exe </process> </query> createKey </query> ...
<label> attack </label> </rec>
<rec> <process> outlook.exe </process> </query> openKey </query> ...
<label> normal </label> </rec>
.....
```

Feature Extraction

Features are important discriminating attributes derived from raw audit data that are employed in detection models. A feature extractor is any module that takes as input raw audit data and outputs additional pieces of information that were computed from the raw data. These new features are augmented to the original record. This can be thought of as a more general version of the forensic analysis engine.

Many features are computed by using information that spans several individual records. This is because many times records by themselves are not meaningful, but in combination with other records they could represent an attack. The data warehouse has the capability to provide the feature extractor with any subset of data necessary. This could be the past n records for use with algorithms based on sequences, or those that compute temporal statistical features of connections or sessions. The flexibility of this system allows any group of record to be used to create a feature.

Features can also be created from a single record. In this case the feature extractor needs only to retrieve a single record and perform any calculations necessary to compute the feature.

Once the feature or features have been calculated they must be appended to the data in the data warehouse. A column is added to the table using the SQL interface to store the values of the new feature. An example of extracting some features gathered from the HAUNT sensor is shown below.

This shows features extracted from three records. In reality features could be extracted from any number of records. This example shows only the calculation of the number of http connections seen by the sensor thus far.

```
<rec>
<ID i> 99 </ID>
<dst-bytes i> 420 </dst-bytes>
<rerror-rate f> 0.12786 </rerror-rate>
<sensor-rate f> 0.16760 </sensor-rate>
<src-bytes i> 1281 </src-bytes>
<src-count i> 132 </src-count>
<src-serror-rate f> 0.19325 </src-serror-rate>
<src str> 128.59.22.69 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>

<rec>
<ID i> 100 </ID>
<dst-bytes i> 325 </dst-bytes>
<rerror-rate f> 0.13426 </rerror-rate>
<sensor-rate f> 0.12450 </sensor-rate>
<src-bytes i> 1341 </src-bytes>
```

```
<src-count i> 242 </src-count>
<src-serror-rate f> 0.12435 </src-serror-rate>
<src str> 128.59.22.63 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>

<rec>
<ID i> 101 </ID>
<dst-bytes i> 425 </dst-bytes>
<rerror-rate f> 0.12456 </rerror-rate>
<sensor-rate f> 0.12654 </sensor-rate>
<src-bytes i> 1311 </src-bytes>
<src-count i> 102 </src-count>
<src-serror-rate f> 0.21325 </src-serror-rate>
<src str> 128.59.22.63 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>
```

The updated records contain a new feature `num_http` which stores the new information.

```
<rec>
<ID i> 99 </ID>
<dst-bytes i> 420 </dst-bytes>
<rerror-rate f> 0.12786 </rerror-rate>
<sensor-rate f> 0.16760 </sensor-rate>
<src-bytes i> 1281 </src-bytes>
<src-count i> 132 </src-count>
<src-serror-rate f> 0.19325 </src-serror-rate>
<src str> 128.59.22.69 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
<num_http> 1 </num_http>
</rec>

<rec>
<ID i> 100 </ID>
<dst-bytes i> 325 </dst-bytes>
<rerror-rate f> 0.13426 </rerror-rate>
<sensor-rate f> 0.12450 </sensor-rate>
<src-bytes i> 1341 </src-bytes>
<src-count i> 242 </src-count>
<src-serror-rate f> 0.12435 </src-serror-rate>
<src str> 128.59.22.63 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
<num_http> 2 </num_http>
```

```
</rec>

<rec>
<ID i> 101 </ID>
<dst-bytes i> 425 </dst-bytes>
<rerror-rate f> 0.12456 </rerror-rate>
<sensor-rate f> 0.12654 </sensor-rate>
<src-bytes i> 1311 </src-bytes>
<src-count i> 102 </src-count>
<src-serror-rate f> 0.21325 </src-serror-rate>
<src str> 128.59.22.63 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
<num_http> 3 </num_http>
</rec>
```

2.5 Efficiency consideration

An important consideration when designing an intrusion detection system is efficiency. A real-time system must be able to respond to intrusions in a timely manner so that action can be taken, without utilizing too many of the resources of the system it is intended to protect. This is especially important in the case of host based systems. The adaptive model generation framework emphasizes light components and a distributed architecture. Resource heavy components can be separate from the system that the IDS is trying to protect. The only component that needs to be run on the system being protected is the lightweight sensor. This greatly minimizes the amount of computational resources taken by the IDS.

An example of where this advantage is useful is in the HAUNT [5] system which is a network intrusion detection system. In section 7 we describe the deployment of the HAUNT system in the AMG framework.

3 Capabilities of Adaptive Model Generation

The adaptive model generation system has many advantages over a traditional intrusion detection system. AMG facilitates real time detection, data collection and storage, model generation and distribution, and various types of analysis of the data. It also facilitates correlation of various data streams.

3.1 Real Time Detection Capabilities

The sensor and detector provide real time detection capabilities to the AMG system.

Both are as light weight as possible and the main advantage of the distributed framework is the ability to isolate the sensor and detector from the remaining components of the system to maximize their real-time performance.

3.2 Automatic Data Collection and Data Warehousing

The distributed architecture of the adaptive model generation system allows for the automation of the data collection and data warehousing. In the AMG framework, simply deploying a sensor will automatically collect and aggregate that sensors data in the data warehouse. There are many reasons why it is desirable to aggregate the date. For example, we may be interested in performing forensic analysis of archival data. It may also be useful to look back at errors made by the intrusion detection system in order to improve performance and study weaknesses.

Heterogeneous Data Support

The distributed architecture of adaptive model generation allow the intrusion detection system to gather data from heterogeneous systems. A set of standard guidelines in a flexible format are placed on sensor data. There are many different types of information that IDSs use such as network packets, application logs, Windows registry accesses, etc. The ability to accommodate these different sources of information in a consistent way is a large advantage of the adaptive model generation system.

This is easily accomplished because all of the data gathered by the system is transmitted to the data warehouse using our XML mark up language. The data warehouse system is flexible enough to store all types of information.

Labeling collected data

Labeling collected data is necessary to create training data for data mining-based detection models. To accomplish this without a tool an administrator would have to manually go through the data record by record and label each piece of data individually. The adaptive model generation system provides a tool that automates the process of labeling training data.

3.3 Model Generation and Management

In a typical network environment, we can conceivably have hundreds of models deployed throughout the network. These models can also become out of data. The management of the models quickly can become a very difficult task.

The adaptive model generation solves this problem by creating a mechanism for the creating and management of detection models. The models are created using the detection model generators. They are then stored in the data warehouse. The data warehouse is robust enough to handle any types of models and therefore the system can be used with any types of models. The data warehouse is also stable enough that failure of model storage is not a concern while the protected machine is under attack. The use of model distributors allows the system to update and alter models on the fly with a minimal computational overhead. This is very advantageous because it allows the system to be deployed for a long period of time without the need for maintenance by an administrator.

3.4 Data Analysis Capabilities

The adaptive model generation provides users with the functionality to implement different data analysis tools. A data analysis tool is anything that retrieves the data from the data warehouse, performs some sort of computation with the data, and then either sends new information back into the data warehouse or uses the new information in some other way. There are many cases where this can be a very useful and important tool. The system is designed so that any data analysis tool can be created and work with the system. We have implemented three types of data analysis tools. They are forensics, feature extraction, and visualization.

Forensics

Forensics is a very important field in computer security. Currently forensic analysis of data is done manually. Computer experts have to search through large amounts of data, sometimes millions of records, individually and look for suspicious behavior. This is an extremely inefficient and expensive process. The adaptive model generation framework contains components that can automate the forensic analysis process.

Forensics can be done with misuse detection models if there is a learned detection model for that data. If a learned detection model exists if can be run over the data and we could find the intrusions in the data after the data has already been collected. The method can be applied with signature-based models which are used by commercial systems today. We can also use anomaly detection models if there exists a normal model for the data set.

In some cases, we do not have an appropriate model of any kind to perform forensics. In these cases, we can use an unsupervised anomaly detection algorithm over the data. Unsupervised anomaly detection algorithms can be used to perform forensic analysis on unlabeled data. The adaptive model generation framework enables this process.

Unsupervised anomaly detection algorithms detect intrusions buried within an unlabeled data set. Unsupervised anomaly detection algorithms are described in section 4.3.

Visualization

The adaptive model generation system provides a visualization tool so that an administrator can examine all of the data in the data warehouse.

This can also provide an administrator or researcher with information about the strengths and weaknesses of a particular intrusion detection system. If an administrator recognizes activity as an attack but the system does not, he can act and the system can be protected even though the system missed the attack. In addition by seeing the activity during an intrusion this can provide insight into the vulnerabilities of the host as well, and better explain how attacks work. This will help to more accurate detection models in the future and provide security experts with the knowledge they need to improve security systems.

Feature Extraction

The success of a model generation algorithm depends largely on the quality and correlation of the data. Feature extractors are components that transform the basic features gathered by the sensors into more meaningful ones, often referred to as advanced features. For example the time stamp on a packet is not a very important feature when considered alone. However using the time stamp to compute the number of packets within the last two seconds can be a crucial piece of information in determining certain types of network attacks[15]. Models learned over well-computed features are generally far superior to those computed over raw pieces of information[14].

Feature extractors can be seen as data analysis engines by the adaptive model generation system. They retrieve data from the data warehouse and then perform computations on that data. Once these computations are completed the new data is sent back to the warehouse and appended with the new information.

In many cases the feature extractors are built into the sensors. This makes the number of components smaller and easier to manage. However this means that a specialized feature extractor must be made for each sensor. This is not a drawback in many cases because features are carefully selected based on analysis of the data gathered by that sensor. In those cases using a feature extractor for two or more different sensors doesn't make any sense. However there exist feature extraction algorithms that can work on multiple types of data, even ones that can work on any data. It is in these cases where feature extraction is particularly useful. This is because the feature extractor can be viewed as an autonomous unit that can be used by any sensor to alter the data and improve performance, with almost no additional production overhead.

Another concern with combining the feature extraction with the sensor is that many feature extraction algorithms can be very computationally expensive. The sensor is the only component that must be run on the system it is protecting. It is therefore crucial that the sensor is very lightweight. Separate feature extraction modules can be extremely helpful in keeping the sensors lightweight.

3.5 Correlation of Multiple Sensors

Distributed models are models that are trained and evaluated over multiple sets of data from multiple sources. Traditional intrusion detection systems would have difficulty combining data from multiple different sources, especially across different networks. Intuitively if a machine learning algorithm has more data from more sources then it will perform better. By eliminating dependencies between sensors, model generators, and detectors the adaptive model generation system has enabled correlation algorithms to be constructed the same as any other algorithm.

The distributed architecture and the data warehouse allow us to implement correlation algorithms with no additional implementation overhead. The data warehouse will allow us to retrieve any subset of the data in the database with a single query. This means that data from multiple sources can be retrieved just as easily as data from a single source. This data can be aggregated and used more efficiently than if it was stored individually.

4 Model Generation Algorithms

There are three types of model generation algorithms that the AMG system supports. The first is misuse detection which trains on labeled normal and attack data. The second is supervised (traditional) anomaly detection which trains on normal data. The third is unsupervised anomaly detection which trains on unlabeled data.

4.1 Misuse Detection

Misuse detection algorithms train over normal and attack data. Using this data, these algorithms build a model that can discriminate between attack records and normal records. These models can then classify new records as either attack or normal. This approach has been very successful in the past [16]. The only major disadvantage of this type of system is that it requires labeled training data that contains labeled normal activity and labeled attacks. This data is very expensive to obtain, and it may not be portable from one system to another or from one network to another.

Misuse detection algorithms can be used as model generation algorithms in the adaptive model generation framework. The training data for misuse detection algorithms must consist of labeled normal and attack data often making the training data for this algorithm very expensive.

Using the AMG system, we can help minimize the cost of labeling the data. Once we deploy the sensors into a network, we can run simulated attacks and record the time stamps and other information about the attack. Since the sensors will be automatically sending the data to the data warehouse, the data for labeling is already aggregated into one location. Using the data labeling tool, we can label the attack data. This labeled data is now stored in the data warehouse and can be retrieved by the model generators. These models can also be distributed in to the detectors using the model distributors.

4.2 Anomaly Detection

Anomaly detection algorithms train over normal data to create a model of normal activity. These algorithms need to train over data that contains no intrusions. The training data needed for these algorithms is expensive because it is difficult to ensure that the data contains no intrusions. This can be done by either having an expert manually clean the data or by somehow ensuring that the data contains no intrusions to begin with. In general this is not as expensive as the training data necessary for misuse detection algorithm. However many anomaly detection algorithms require a very large amount of training data which can increase the cost.

Once an anomaly detection model is trained, it can then classify new data as normal or anomalous. These algorithms rely on the assumption that attacks are cause behavior that is different from normal.

The adaptive model generation framework supports the creation of anomaly detection models. Since sensors send data to the data warehouse, it is easy to aggregate the data for collection. Using the forensics analysis engine, we can help check to see if the data is clean and contains no intrusions. This can greatly decrease the cost of creating the training set since it speeds the process of verifying that the data is clean. The model generators can automatically generate anomaly detection models using the data from the data warehouse and deploy the detection models using the model distributor.

4.3 Unsupervised Anomaly Detection

Unsupervised anomaly detection algorithms examine unlabeled data and attempt to detect intrusions buried within the unlabeled data. Unsupervised anomaly detection algorithms makes the assumptions that intrusions are very rare compared to the normal data and they are also quantitative different. Because of this, intrusions are outliers in the data and can be detected. Unsupervised anomaly detection is discussed in depth in [6, 20, 7].

Since unsupervised anomaly detection can detect intrusions in an unlabeled data set, they are used inside the forensics analysis engines. Data from the data warehouse is sent to a forensics analysis engine where a unsupervised anomaly detection algorithm is applied. The forensics analysis engine can label the data which it determines to be an outlier.

Unsupervised anomaly detection algorithms can also be used to help label data that is collected by the system. This labeled data can then be used to train a misuse or anomaly detection model.

5 Model Generation Example: SVM

One specific type of model generation algorithm used by AMG is Support Vector Machines (SVMs). SVMs were first introduced by Vapnik [25] as a supervised machine learning algorithm. Vapnik's SVM algorithm is used in AMG for misuse detection. An unsupervised variant of the SVM algorithm was put forth by Schölkopf et. al. [22]. This algorithm can be used for both Unsupervised Anomaly Detection and normal Anomaly Detection.

5.1 SVM Algorithm

Vapnik's SVM algorithm is a binary classifier. The idea behind an SVM approach to intrusion detection is that we map our data to a *feature space.* Inside this feature space, use use the SVM and a set of labeled training data to determine a linear decision surface (hyperplane). This surface is then used to classify future instances of data. Data is classified based upon which side of the decision surface it falls.

Given a training set S consisting of m vectors and their labels (x_i, y_i) where $x_i \in \Re^n$ and $y_i \in \{\pm 1\}$, the algorithm generates a decision surface. The decision surface is a hyperplane of the form $\langle w, x\rangle + b = 0$ where w is normal to the hyperplane and b scalar that shifts the hyperplane. The decision surface that is chosen is determined by solving an optimization problem that determines the "best" hyperplane under a set of criteria which is described fully in [3].

The classification of a future instance $x \in \Re^n$ is made by the function

$$f(x) = \operatorname{sgn}(\langle w, x\rangle + b)$$

It is shown in [3] that solving the following optimization problem results in a solution the solution to the SVM optimization.

$$\text{maximize: } \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j\rangle$$

$$\text{subject to } 0 \leq \alpha_i, \sum_i \alpha_i y_i = 0$$

Setting $b = 0$, the solution is then:

$$w = \sum_i \alpha_i y_i x_i$$

All x_i with $\alpha_i \neq 0$ are called the support vectors. These are the vectors on the boarder of each class that determine the unique solution. If a support vector were removed it would change the resulting hyperplane. However, all non-support vectors are irrelevant to the solution. They can all be removed and the solution would not change.

This algorithm performs best when the data is linearly separable data. However in order to work for the non-linearly separable case, data must be mapped into a higher dimension feature space where it does become linearly separable. In addition, often intrusion detection data are not all vectors in $\Re^n$ so there is no natural definition of the dot products between the data elements.

Since the SVM algorithm is defined in terms of dot products, we can use kernel functions to define both the mappings of the elements to the feature space and the dot product within these space simultaneously. This fact can be exploited and a kernel function can be used in place of the dot product.

Let Φ be a feature map $\Phi : X \to F$. Φ maps the input space X into a dot product space called the feature space F. A kernel function K implicitly maps data into this feature space and takes the dot product in that space.

$$K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j)\rangle$$

An example of a kernel function is the Gaussian kernel.

$$K(x_i, x_j) = e^{-||x_i - x_j||^2/2\sigma^2}$$

Now the support vector machine optimization equation and classification equation can be rewritten in terms of kernels.

$$\text{maximize} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$\text{subject to: } 0 \leq \alpha_i, \sum_i \alpha_i y_i = 0$$

Substituting the formula for w into the classifier equation we get another dot product that can be converted to a kernel. Setting $b = 0$ the solution is then:

$$f(x) = \text{sgn}\left(\sum_{i=1}^{N_s} \alpha_i y_i K(s_i, x) + b \right)$$

where N_s is the number of support vectors and s_i is the i^{th} support vector

5.2 SVM for Misuse Detection in AMG

The standard support vector machine algorithm is used for misuse detection by AMG. Data in the form of vectors of real numbers are sent from the sensors to detectors. The detectors use a SVM model to differentiates between normal data and intrusion data.

To implement this in the AMG system, training data must first be generated. A system is monitored by sensors that send their observations to the data warehouse in the form of XML tagged data. Sporadically, different attacks are launched against the system. After enough training data has been generated, data is labeled in the data warehouse as either normal or attack. This labeled data is then sent via XML to the model generator. The model generator uses the SVM algorithm to create a model for misuse detection. A model, in this case, is the set of support vectors and their weights. This model is automatically sent to the data warehouse for storage and to all of the detectors that use this kind of model. Once the model is in place, sensors send data that they are monitoring to the for classification by the SVM classification rule.

SVMs could also be used in misuse detection to determine what kind of attack is being carried out against the system. This would require labeling training data with a more specific attack type label. Then a set of support vector machines can be training with each one trying to detect a specific attack. This basically equates to taking the intersection of these support vector machines. This would not add much additional complexity but it might interfere with the accuracy of the classification.

5.3 Unsupervised SVM Algorithm

The standard SVM algorithm is a supervised learning algorithm. It requires labeled training data to create its classification rule. Schölkopf adapted the SVM algorithm into an unsupervised learning algorithm. This unsupervised variant does not require its training set to be labeled to determine a decision surface.

The algorithm is similar to the standard SVM algorithm in that it uses kernel functions to perform implicit mappings and dot products. It also uses the same kind of hyperplane for the decision surface. The solution is only dependent on the support vectors as well. However, the support vectors are determined in a different way. This algorithm attempts to find a small region where most of the data lives and label it as class $+1$. Everywhere else is labeled as class -1. This is accomplished by finding the hyperplane that maximizes the distance from the origin while still capturing the majority of the data. The support vectors define that hyperplane.

Given a training set S consisting of m vectors $x_i \in \Re^l$

$$\text{minimize: } \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j K(x_i, x_j)$$

$$\text{subject to: } 0 \leq \alpha_i \leq \frac{1}{vl}, \sum_i \alpha_i = 1$$

where $0 < v < 1$ is a parameter that controls the trade off between maximizing the distance from the origin and containing most of the data in the region created by the hyperplane.

The classification equation is:

$$f(x) = \text{sgn}\left(\sum_{i=1}^{N_s} \alpha_i K(s_i, x) - b\right)$$

where N_s is the number of support vectors and s_i is the i^{th} support vector

For this algorithm b cannot be set to 0, it must be found explicitly.

$$b = \sum_{j=1}^{N_s} \alpha_j K(s_j, s_i)$$

5.4 Unsupervised SVM for Unsupervised Anomaly Detection

The standard SVM algorithm is not useful for Unsupervised Anomaly Detection as it requires labeled data. The unsupervised SVM variant proposed by Schölkopf can be used for UAD. This approach was used in [7] to perform UAD using the AMG framework. The algorithm differentiates between normal data and anomalous data. Anomalous data is thought to be intrusion data because intrusions are mush different than normal system use.

Like the misuse detection algorithm, UAD requires training data. During a training period a system is run normally with no attacks. Sensors monitoring the system send their observations via XML to the data warehouse. Although no attacks are intentionally run, if some unknown attacks were to occur, there would be no problem. The algorithm can tolerate some noise (unknown attacks) and still generalize well. Once enough training data has been accumulated it is sent from the data warehouse to the model generator via XML. There is no need to label data. The model generator then uses the unsupervised SVM algorithm to generate a model of normal activity. This model is made up of the set of support vectors and their associated weights. The model is then sent to the data warehouse for storage and to the appropriate detectors. Once the detection model is in place, sensors send data to the detector for classification.

Supervised Anomaly Detection can also be performed using the same method. However, all of the training data would have to be guaranteed to be attack free.

6 System Example 1: Registry Anomaly Detection

The AMG framework can support a variety of different intrusion detection systems. One example of an IDS system that is integrated into the AMG system is the *Registry Anomaly Detection* (RAD) system. The RAD system is a host-based IDS system which runs on the Microsoft Windows platform. RAD monitors the accesses to the windows registry on a host and detects anomalous registry accesses that correspond to attacks. It uses an anomaly detection algorithm to make models of normal registry accesses and compares in real time, monitored accesses to that model. Details on the system as well as details on experiments showing the effectiveness of the RAD system are presented in [1].

6.1 The RAD Data Model

The RAD system uses 10 features to characterize each registry access. Five of these are basic features that come directly from the registry accesses, and five are composite features which are made from the combination of two of the basic features. The basic features are `Key`, `Process`, `Query`, `Response`, and `ResultValue`. The advanced features are `Process/Query`, `Key/Process`, `Query/Key`, `Response/Key`, and `ResultValue/Key`. The first five features are derived directly from the registry accesses.

The registry is stored in a tree structure, and all information is stored in a data structured called a key. The name of the location where the information is stored is the '`Key` basic feature. The `Process` feature is the name of the process that is performing the registry access. The `Query` feature represents the type of access being made, such as `QueryValue`, `CreateKey`, `SetValue`, etc. The `Response` feature is the outcome of the query, such as `success`, `not found`, `access denied`, etc. The `Result` feature is the value of the key being accessed. These five features provide all the necessary information about any single registry access.

6.2 The RAD Sensor

The RAD sensor consists of two parts. The first part connects to the Windows operating system and monitors the accesses to the registry. This part is implemented as a basic auditing module(BAM). The BAM includes a hook into the audit stream which is the windows registry. The architecture of the system is taken from the commercial software Regmon produced by SysInternals[24]. The BAM uses Win32 hooks to listen for all reads and writes to registry.

The second part of the RAD sensor is the communication component which translates this data into our XML format and sends it to the data warehouse. The communication module can supports multiple BAMs at the same time. This is done so that all sensors running on a host can be sent through a single source. Then the communication engine can send the data from all these sources to the data warehouse for storage.

The RAD BAM is not trivial to implement. The most direct method to implement the RAD BAM is through the Windows Event Log. However, the Windows Event Log is unable to handle the amount of traffic generated by the registry which required the use of more sophisticated Win32 hooks.

The five composite features that are used by the RAD system are examples of feature extraction. This is the simplest type of feature extraction possible, the combination of two basic fields. The RAD system uses these composite features in order to better classify activity. This is an example of the feature extraction capabilities of the adaptive model generation system. This is one of the cases where the feature extractor is very lightweight and therefore a part of the sensor.

6.3 The RAD Classification Algorithm

These ten features are used to classify each registry access as either normal or anomalous. In order to do this we implemented an anomaly detection algorithm first introduced by Phil Chan and Mathew Mahoney in the PHAD system[18]. This algorithm was first used to detect anomalies within packet headers in order to look for malicious behavior. The adaptive model generation algorithm allows us to use this algorithm even though it was created for Packet Headers. The algorithm trains over normal data to develop models of what normal activity is like and how to classify abnormal activity. The algorithm is based on keeping statistics about the data and using those statistics to determine anomalous features.

Each feature is individually evaluated to be either normal of anomalous. Then the statistics we gathered are used to score these anomalies. This score is based on how likely we think it is that the value of this feature will be different then values seen in the past. These scores are then added together and if they are over a threshold then the access is considered to be malicious, otherwise it is classified as normal. This algorithm is however not important to the adaptive model generation system, in reality any algorithm could have been used and it would not have effected the overall architecture. Also from the point of view of the classification algorithm the sensor is not important. This algorithm could have been used on any data without any changes in architecture.

6.4 The RAD Detector

In order to detect anomalies in real time, a detector was implemented for the RAD system. This detector was implemented specifically for the RAD system but it could be used to evaluate any model that was created by the classification algorithm described in the previous section. The first requirement of the detector is that is must receive data from the sensor in real time. This is necessary to evaluate models in real time. The detector must also decode the model and have the capability to receive real time updates to this model. The RAD detector would retrieve the model from the data warehouse, decode it, and then evaluate each record that it was sent from the sensor. This is all done in real time and consequently the system is successful in detecting malicious activity in real time.

The multiplicity of this system can easily be increased with the adaptive model generation system. With no changes in architecture the system can support any number of host machines and sensors. Without the adaptive model generation architecture increasing the multiplicity would require major changes in the structure of a system. This is because the central data collection is automated in AMG. This means that data from multiple machines is gathered in the same place and can be analyzed from that central location.

Since the focus of this paper to highlight the RAD system as an instance of AMG we are not reporting the results in this paper. The results of this system can be found in [1].

7 System Example 2: HAUNT

The *Heuristic Audit of Network Traffic* (HAUNT) system is a network based intrusion detection system that classifies network data as either normal or attack. Previous research has shown that network packet information can be useful in detecting intrusions. The majority of commercial intrusion detection systems use network data to detect attacks. This is because many attacks are remote attacks and they can be seen in the network data. However these commercial systems are signature-based due to the high cost of deploying a data mining based network intrusion detection system.

Again, the focus of this description of the HAUNT system is to describe how it is integrated into the AMG system. Details as well as experiments evaluating the performance of HAUNT are presented in detail in [5].

7.1 HAUNT Sensor

The HAUNT sensor is designed to gather information from a network stream. It listens to all network data, formats it, and sends that data directly to the data warehouse. The network sensor does not use a communication engine because it does not run on a host, so there is no need to aggregate information before it is sent to the data warehouse. The HAUNT sensor is implemented by utilizing the commercial products NFR [12] and Snort [21]. They use an abstract feature definition structure and a feature exchange protocol to extract information from the NFR and Snort systems. The HAUNT system only uses packet header information to extract features. This is done for efficiency purposes and because the system can be effective and inexpensive using just this information.

7.2 HAUNT Classification Algorithm

The HAUNT system uses a multiple model cost-sensitive approach to improve efficiency. The system is designed to minimize the computational cost of an intrusion detection system. The system first attempts to make a classification based on a simple rule and the basic data gathered from the sensor. If the system can not confidently make a classification at that point the system will perform more calculations in order to make a better decision. The system accomplishes this by implementing multiple models to classify the data. The difference between the models is that some are more accurate at the price of being more computationally expensive. The system does not evaluate the more expensive models unless it has to in order to make a classification. The more expensive models are more expensive in large part due to the fact that they require more data. These expensive models require derived features from the packet information. Some of these features are very expensive to calculate and therefore they are only calculated when needed by the more expensive models.

7.3 HAUNT Detector

The HAUNT system uses a special type of detector called JUDGE that implements multiple model evaluation.

The JUDGE system was implemented as a part of the HAUNT system in order to accomplish the evaluation of the multiple models. The JUDGE system is the system that decides whether to calculate more expensive features and evaluate more expensive models. The JUDGE models are models generated from the RIPPER[2] model generation program. RIPPER generates rule sets for evaluation by the JUDGE system. These rule sets come in one of two different types. The first type is ordered rule sets. When evaluating ordered rule sets JUDGE goes through each rule one by one until one of the rules can make a classification and then that rule makes the decision. The second type of rule set is unordered rule sets. When evaluating unordered rule sets each rule in the set is evaluated and the rule with the most precise ruling makes the ruling. The unordered rule sets are more precise because they are always labeled by most precise classifying rule. However ordered rule sets are faster because in many cases JUDGE does not have to evaluate every rule in the rule set.

7.4 HAUNT Feature Extraction

The HAUNT system uses a feature extractor to discover features that are useful for detecting attacks. The algorithms for performing this feature discovery are described in [14].

The HAUNT system uses feature descriptor in order to define the features that it uses for classification. These features are defined using arithmetic and logic expressions to combine primitive features. The logic expressions implemented by this system are SUM, AND, and UNIQUE. These features can be used to create a wide variety of important features. The sum feature could be used to calculate the total number of times something has happened. For example if we wanted to calculated the total number of tcp connections we could use

num_tcp_connections = SUM(protocol==tcp)

The SUM(protocol==tcp) which return the total of number of records of which the condition service==http is true. If we wanted to calculate the total number of tcp connections going to port 2301 we could use.

num_tcp_connections_to_port_2301 = SUM((protocol==tcp) AND destination_port==2301))

The AND operator is used to take the AND of two conditional expressions the same way it is normal used. The final logical operator is the UNIQUE operator. The UNIQUE operations takes in two parameters, a conditional, and a feature. The operator will return the number of unique values that feature has had when the condition is true. For example to get the number of different ports accessed by tcp protocol we would use.

num_tcp_ports = UNIQUE(protocol==tcp, destination_port)

These logical functions along with arithmetic functions such as multiplication and addition are all the HAUNT system needs to define all of the features it uses. The feature extraction provided by these tools can be seen as a data analysis engine by the adaptive model generation system. Feature extraction is an important part of many intrusion detection systems. The HAUNT system is an example of the feature extraction capabilities of the adaptive model generation system can be extremely useful in enabling an intrusion detection system. Without feature extraction capabilities the HAUNT system would not be realizable.

Since the focus of this paper to highlight the HAUNT system as an instance of AMG we are not reporting the results in this paper. The results of this system as well as further implementation details can be found in [5].

8 Conclusion

In this paper we presented adaptive model generation, a method for automatically and efficiently creating models for an intrusion detection system. The system uses data collected by sensors to create a model and to search for attacks. It uses an anomaly detection algorithm to create models to classify data. The system updates models in real time so that over time performance can be improved. Adaptive model generation can significantly reduce the cost of deploying an intrusion detection system because it streamlines the process of gathering training data, creating models, and evaluating data.

The system uses a distributed architecture with autonomous components to increase the flexibility of the system. This additional flexibility has allowed us to create many modules and tools that greatly reduce the cost of deploying an intrusion detection system. Automated data collection and storage saves the time and effort to manually gather data. The data labeling tool streamlines the process of labeling the data. This greatly reduces cost because labeling data requires a security expert to check each record individually, which in some cases could be millions of records. Automated model generation and distribution saves the cost of manually updating the models as they become outdated. Visualization capabilities allow an administrator to be involved with the intrusion detection in a manner that would not be possible with traditional black box intrusion detection. This reduces the deployment cost dramatically by circumventing the slow and lengthy process of gather and labeling training data.

The support of heterogeneous data and central storage of data enables the system to combine data from different sources very easily. In traditional systems combining data between sensors is not possible. The distributed architecture of the adaptive model generation system allows machine learning algorithms use data from multiple sources just as easily as data from one source. This allows correlation algorithms to be integrated into the adaptive model generation framework which could potentially increase the performance of an intrusion detection system.

Unsupervised anomaly detection algorithms eliminate the need for specialized training data. A system using an unsupervised anomaly detection algorithm could be deployed without labeled training data. This system can train over raw audit data gathered from sensors. This substantially reduces the deployment cost of an IDS. This property of unsupervised anomaly detection algorithm makes them very effective in the deployment of forensic analysis systems. The adaptive model generation implements data analysis components which can analyze data in the data warehouse and append information to the data. This in combination with an unsupervised anomaly detection will yield a forensic analysis system that can detect attacks within a data set without prior knowledge of the data set.

Future work includes expanding the correlation abilities of the system. Since all data of the system is stored in a central location we can combine data from different sensors in order to obtain a better picture of activity on a host machine or network. This approach will allow for the building of more complicated and more effective models. This approach will also allow us to study the relationship between data about the same attack that is gathered from different sources. An understanding of this relationship would aid greatly in future work on intrusion detection systems.

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. The adaptive model generation framework allows us to do this linking at a very small cost.

References

[1] Frank Apap, Andrew Honig, Shlomo Hershkop, Eleazar Eskin, and Sal Stoflo. Detecting malicious software by monitoring anomalous windows registry accesses. Technical report, CUCS Technical Report, 2001.

[2] William W. Cohen. Fast effective rule induction. In *International Conference on Machine Learning*, pages 115–123, 1995.

[3] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.

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

[5] Paolo De Dios, Raka El-Khalil, Kyri Sarantakos, Matthew Miller, Eleazar Eskin, Wenke Lee, and Salvatore Stolfo. Heuristic audit of network traffic: A data mining-based approach to network intrusion detection. Technical report, CUCS Technical Report, 2001.

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

[7] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Salvatore Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. Technical report, CUCS Technical Report, 2002.

[8] Eleazar Eskin, Wenke Lee, and Salvatore J. Stolfo. Modeling system calls for intrusion detection with dynamic window sizes. In *Proceedings of DARPA Information Survivabilty Conference and Exposition II (DISCEX II)*, Anaheim, CA, 2001.

[9] 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.

[10] 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.

[11] Shlomo Hershkop, Frank Apap, Eli Glanz, Tania D'alberti, Eleazar Eskin, Sal Stolfo, and Johnee Lee. Hobids: A data mining approach to host based intrusion detection. Technical report, CUCS Technical Report, 2001.

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

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

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

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

[16] 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.

[17] 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.

[18] M. Mahoney and P. Chan. Detecting novel attacks by identifying anomalous network packet headers. Technical Report CS-2001-2, Florida Institute of Technology, Melbourne, FL, 2001.

[19] V. Paxson. Bro: A system for detecting network intruders in real time. In *7th Annual USENIX Security Symposium*, 1998.

[20] Leonid Portnoy, Eleazar Eskin, and Salvatore J. Stolfo. Intrusion detection with unlabeled data using clustering. In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*, 2001.

[21] M. Roesch. Snort - lightweight intrustion detection for networks. In *Proceedings of Lisa '99*, 1999.

[22] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. Technical Report 99-87, Microsoft Research, 1999. To appear in *Neural Computation*, 2001.

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

[24] SysInternals. Regmon for Windows NT/9x. *Online publication*, 2000. http://www.sysinternals.com/ntw2k/source/regmon.shtml.

[25] V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition [in Russian]*. Nauka, Moscow, 1974. (German Translation: W. Wapnik & A. Tscherwonenkis, *Theorie der Zeichenerkennung*, Akademie–Verlag, Berlin, 1979).

[26] 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.