US007225343B1

(12) **United States Patent**
Honig et al.

(10) **Patent No.: US 7,225,343 B1**
(45) **Date of Patent: May 29, 2007**

(54) **SYSTEM AND METHODS FOR ADAPTIVE MODEL GENERATION FOR DETECTING INTRUSIONS IN COMPUTER SYSTEMS**

(75) Inventors: **Andrew Honig**, East Windsor, NJ (US); **Andrew Howard**, Lansdale, PA (US); **Eleazar Eskin**, Santa Monica, CA (US); **Salvatore J. Stolfo**, Ridgewood, NJ (US)

(73) Assignee: **The Trustees of Columbia University in the City of New York**, New York, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 442 days.

(21) Appl. No.: **10/352,342**

(22) Filed: **Jan. 27, 2003**

Related U.S. Application Data

(60) Provisional application No. 60/351,913, filed on Jan. 25, 2002.

(51) **Int. Cl.**
H04L 9/00 (2006.01)

(52) **U.S. Cl.** **713/194**; 713/193; 713/189

(58) **Field of Classification Search** 713/194, 713/193, 189
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,278,901 A * 1/1994 Shieh et al. 726/22
5,448,722 A * 9/1995 Lynne et al. 706/49
5,621,889 A * 4/1997 Lermuzeaux et al. 726/22

OTHER PUBLICATIONS

U.S. Appl. No. 10/352,343, filed Jan. 27, 2003 claiming priority to U.S. Appl. No. 60/351,857, filed Jan. 25, 2001, entitled "Behavior Based Anomaly Detection For Host-Based Systems For Detection Of Intrusion In Computer Systems," of Frank Apap, Andrew Honig, Shlomo Hershkop, Eleazar Eskin and Salvatore J. Stolfo.

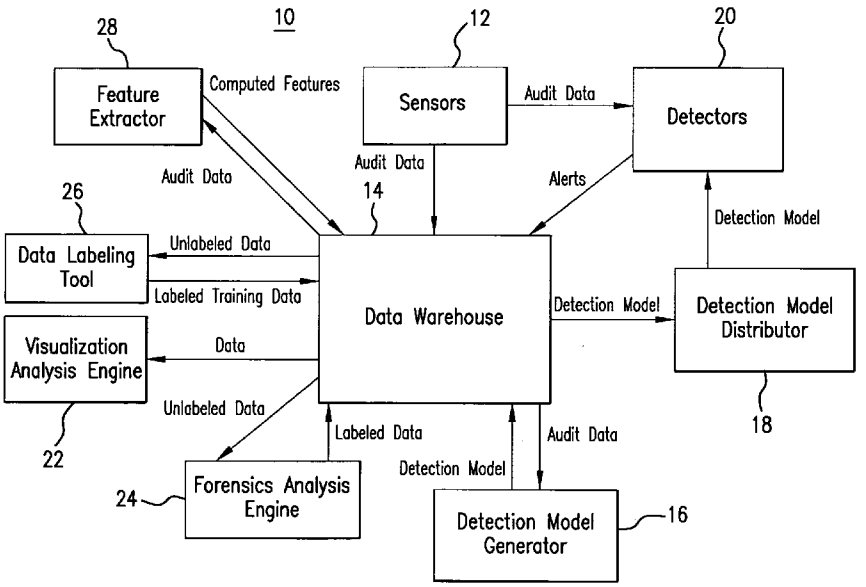
(Continued)

Primary Examiner—Thomas R. Peeso
(74) *Attorney, Agent, or Firm*—Baker Botts LLP

(57) **ABSTRACT**

A system and methods for detecting intrusions in the operation of a computer system comprises a sensor configured to gather information regarding the operation of the computer system, to format the information in a data record having a predetermined format, and to transmit the data in the predetermined data format. A data warehouse is configured to receive the data record from the sensor in the predetermined data format and to store the data in a database. A detection model generator is configured to request data records from the data warehouse in the predetermined data format, to generate an intrusion detection model based on said data records, and to transmit the intrusion detection model to the data warehouse according to the predetermined data format. A detector is configured to receive a data record in the predetermined data format from the sensor and to classify the data record in real-time as one of normal operation and an attack based on said intrusion detection model. A data analysis engine is configured to request data records from the data warehouse according to the predetermined data format and to perform a data processing function on the data records.

49 Claims, 3 Drawing Sheets



OTHER PUBLICATIONS

U.S. Appl. No. 10/327,811, filed Dec. 19, 2002 claiming priority to U.S. Appl. No. 60/342,872, filed Dec. 20, 2001, entitled "System And Methods for Detecting A Denial-Of-Service Attack On A Computer System" of Salvatore J. Stolfo, Shlomo Hershkop, Rahul Bhan, Suhail Mohiuddin and Eleazar Eskin.
U.S. Appl. No. 10/320,259, filed Dec. 16, 2002 claiming priority to U.S. Appl. No. 60/328,682, filed Oct. 11, 2001 and U.S. Appl. No. 60/352,894, filed Jan. 29, 2002, entitled "Methods of Unsupervised Anomaly Detection Using A Geometric Framework" of Eleazar Eskin, Salvatore J. Stolfo and Leonid Portnoy.
U.S. Appl. No. 10/269,718, filed Oct. 11, 2002 claiming priority to U.S. Appl. No. 60/328,682, filed Oct. 11, 2001 and U.S. Appl. No. 60/340,198, filed Dec. 14, 2001, entitled "Methods For Cost-Sensitive Modeling For Intrusion Detection" of Salvatore J. Stolfo, Wenke Lee, Wei Fan and Matthew Miller.
U.S. Appl. No. 10/269,694, filed Oct. 11, 2002 claiming priority to U.S. Appl. No. 60/328,682, filed Oct. 11, 2001 and U.S. Appl. No. 60/339,952, filed Dec. 13, 2001, entitled "System And Methods For Anomaly Detection And Adaptive Learning" of Wei Fan, Salvatore J. Stolfo.
U.S. Appl. No. 10/222,632, filed Aug. 16, 2002 claiming priority to U.S. Appl. No. 60/312,703, filed Aug. 16, 2001 and U.S. Appl. No. 60/340,197, filed Dec. 14, 2001, entitled "System And Methods For Detecting Malicious Email Transmission" of Salvatore J. Stolfo, Eleazar Eskin, Manasi Bhattacharyya and Matthew G. Schultz.
U.S. Appl. No. 10/208,432, filed Jul. 30, 2002 claiming priority to U.S. Appl. No. 60/308,622, filed Jul. 30, 2001 and U.S. Appl. No. 60/308,623, filed Jul. 30, 2001, entitled "System And Methods For Detection Of New Malicious Executables" of Matthew G. Schulz, Eleazar Eskin, Erez Zadok and Salvatore J. Stolfo.
U.S. Appl. No. 10/208,402, filed Jul. 30, 2002 claiming priority to U.S. Appl. No. 60/308,621, filed Jul. 30, 2001, entitled "System And Methods For Intrusion Detection With Dynamic Window Sizes" of Eleazar Eskin and Salvatore J. Stolfo.
Honig A et al., (2002) "Adaptive Model Generation: An Architecture for the Deployment of Data Minig-based Intrusion Detection Systems." To Appear in Data Mining for Security Applications, *Kluwer*.
Burroughs D et al., Apr. 2002, "Analysis of Distributed Intrusion Detection Systems Using Bayesian Methods" presented at *IPCCC*.
Eskin E et al., (2002) "A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data." *Technical report, CUCS Technical Report*.
Apap F et al., (2001) "Detecting malicious software by monitoring anomalous windows registry accesses." *Technical report, CUCS Technical Report*.
Eskin E et al., (2001) "Modeling system calls for intrusion detection with dynamic window sizes." In *Proceedings of DARPA Information Survivability Conference and Exposition II* (*DISCEX II*), Anaheim, CA.
Mahoney M et al., (2001) "Detecting novel attacks by identifying anomalous network packet headers." *Technical Report CS-2001-2*, Florida Institute of Technology, Melbourne, FL.
Portnoy L et al., (2001) "Intrusion detection with unlabeled data using clustering." In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security* (*DMSA-2001*).
Schölkopf B et al., (To appear in *Neural Computation*, 2001) "Estimating the support of a high-dimensional distribution." *Technical Report 99-87*, Microsoft Research, 1999.
Eskin E et al., (2000) "Anomaly detection over noisy data using learned probability distributions." In *Proceedings of the Seventeenth International Conference on Machine Learning* (*ICML-2000*).
Eskin E et al., Nov. 2000, "Adaptive Model Generation for Intrusion Detection Systems." *Workshop on Intrusion Detection and Prevention*, 7th ACM Conference on Computer Security, Athens, GR.
Lee W et al., Nov. 2000, "A framework for constructing features and models for intrusion detection systems." *ACM Transactions on Information and System Security*, 3(4)1-33.
Lane T et al., (1999) "Temporal sequence learning and data reduction for anomaly detection." *ACM Transactions on Information and System Security*, 2(3):295-331.
Warrender C et al., (1999) "Detecting intrusions using system calls: alternative data models." In *Proceedings of the 1999 IEEE Symposium on Security and Privacy, IEEE Computer Society*, pp. 133-145.
Lee W et al., Aug. 1998, "Mining Audit Data to Build Intrusion Detection Models" In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (KDD '98), New York, NY.
Lee W et al., (1998), "Data mining approaches for intrusion detection." In *Proceedings of the Seventh USENIX Security Symposium*.
Paxson V, (1998) Bro: A system for detecting network intruders in real time. In *7th Annual USENIX Security Symposium*.
Staniford-Chen S et al., Oct. 1998, "The common intrusion detection framework (cidf)." In *Proceedings of the Information Survivability Workshop*.
Lane T et al., (1997) "Sequence Matching and Learning in Anomaly Detection for Computer Security" AAAI Workshop: *AI Approaches to Fraud Detection and Risk Management* pp. 49-49.
Lee W et al., (1997) "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* pp. 50-56. Menlo Park, CA: AAAI Press.
Forrest S et al., (1996) "A sense of self for unix processes." In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pp. 120-128. IEEE Computer Society.
Cohen WW, (1995) Fast effective rule induction. In *International Conference on Machine Learning*, pp. 115-123.
Denning DE, (1987) An intrusion detection model. *IEEE Transactions on Software Engineering*, SE-13:222-232.

* cited by examiner

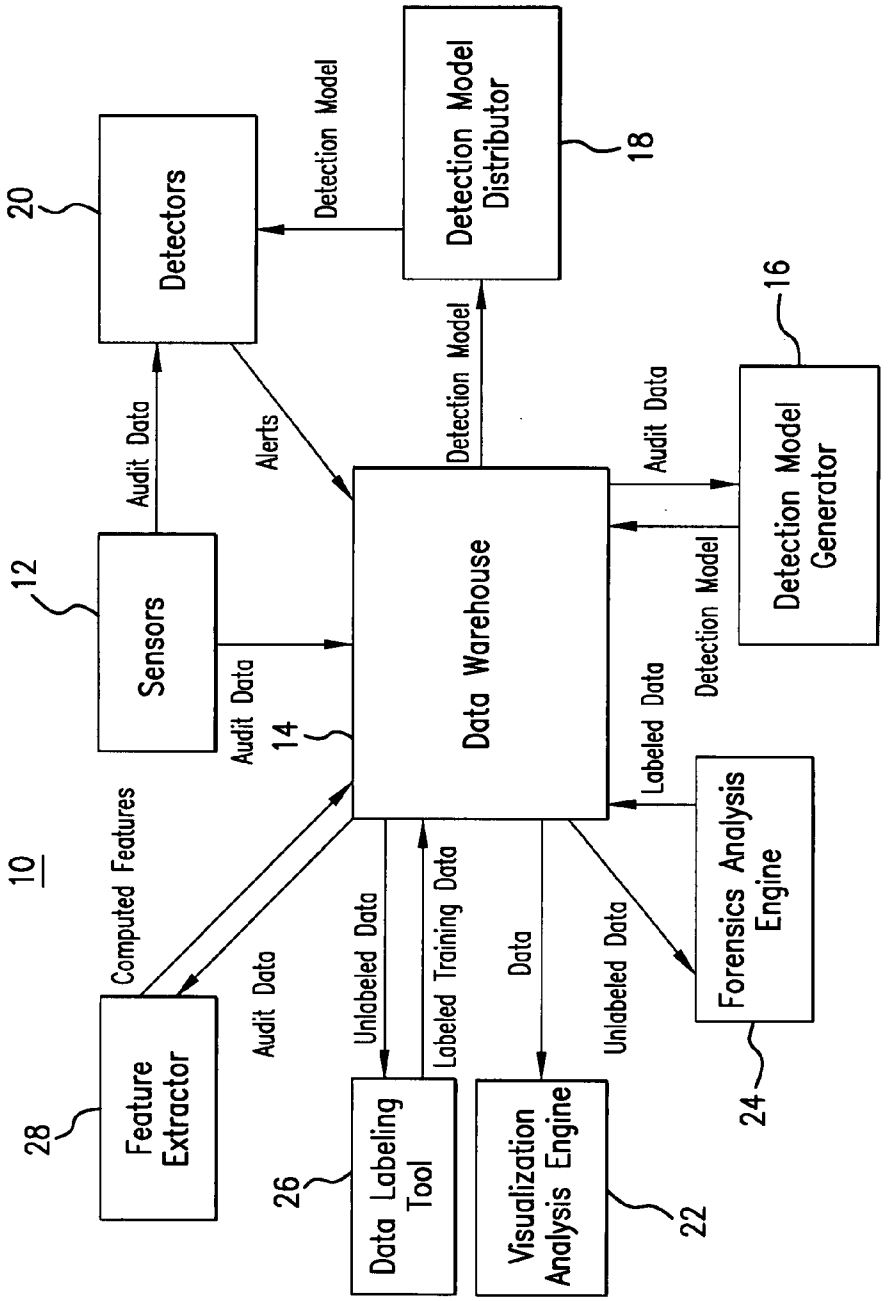

10
Sensors
12
Audit Data
Detectors
20
Alerts
Detection Model
Detection Model Distributor
18
Detection Model
Audit Data
Data Warehouse
14
Audit Data
Detection Model
Detection Model Generator
16
Computed Features
Feature Extractor
28
Audit Data
Unlabeled Data
Labeled Training Data
Data Labeling Tool
26
Data
Visualization Analysis Engine
22
Unlabeled Data
Labeled Data
Forensics Analysis Engine
24

FIG. 1

idsWatch

File Tools Help

Sensor Activity | IDS Alerts | HOBIDS | HAUNT | ASIDS | Query Tool

date	user	computer	action	cat	ty	string 1	string 2	string 3
Nov 27, 2000	SYSTEM	PEACH	515	1	8	Service Co...	I	I
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...	I	I
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	I	I
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...	I	I
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

Refresh

52 54 56 58 60 62 64 66 68

FIG.2

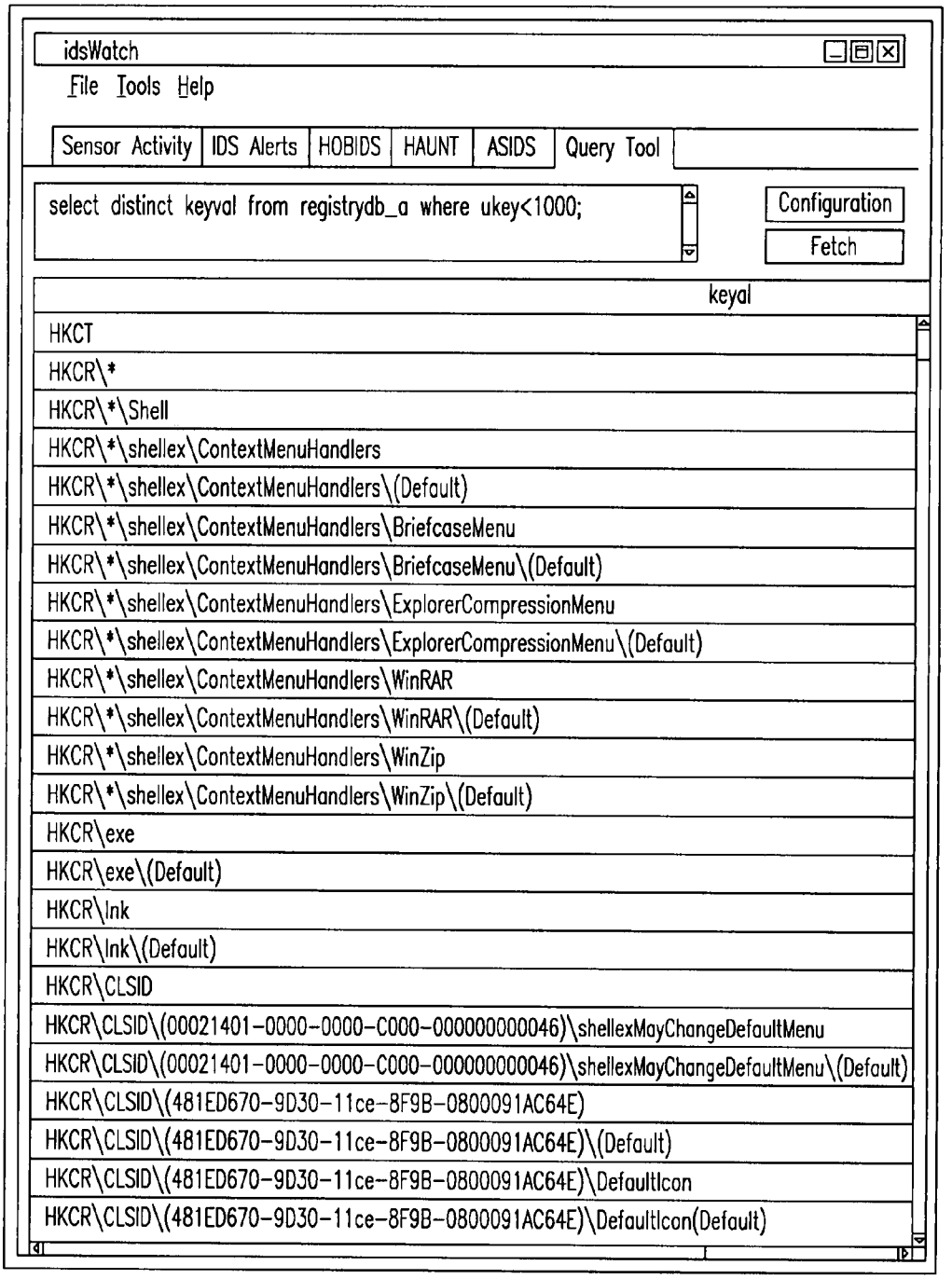

idsWatch
File Tools Help
Sensor Activity
IDS Alerts
HOBIDS
HAUNT
ASIDS
Query Tool
select distinct keyval from registrydb_a where ukey<1000;
Configuration
Fetch
keyal
HKCT
HKCR*
HKCR*\Shell
HKCR*\shellex\ContextMenuHandlers
HKCR*\shellex\ContextMenuHandlers\(Default)
HKCR*\shellex\ContextMenuHandlers\BriefcaseMenu
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)\shellexMayChangeDefaultMenu
HKCR\CLSID\(00021401-0000-0000-C000-000000000046)\shellexMayChangeDefaultMenu\(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)

FIG.3

SYSTEM AND METHODS FOR ADAPTIVE MODEL GENERATION FOR DETECTING INTRUSIONS IN COMPUTER SYSTEMS

CLAIM FOR PRIORITY TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Patent Application Ser. No. 60/351,913, filed on Jan. 25, 2002, entitled "Data Warehouse Architecture for Adaptive Model Generation Capability in Systems for Detecting Intrusion in Computer Systems," which is hereby incorporated by reference in its entirety herein.

STATEMENT OF GOVERNMENT RIGHT

The present invention was made in part with support from United States Defense Advanced Research Projects Agency (DARPA), grant nos. FAS-526617, SRTSC-CU019-7950-1, and F30602-00-1-0603. Accordingly, the United States Government may have certain rights to this invention.

COMPUTER PROGRAM LISTING

A computer program listing is submitted in duplicate on CD. Each CD contains a routines described in the Appendix, which CD was created on Jan. 24, 2003, and which is 406 kB in size. The files on this CD are incorporated by reference in their entirety herein.

COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by any one of the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to systems and methods for detecting anomalies in a computer system, and more particularly to an architecture and data format for using a central data warehouse and heterogeneous data sources.

2. Background

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.

Many widely used and commercially available IDSs are signature-based systems. As is known in the art, 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. For example, a signature-based method is designed to only detect attacks for which it contains a signature in the database. Therefore, the signature-based methods cannot detect unknown attacks since there is no signature in the database for them. Such unknown

attacks can be dangerous because the system is completely vulnerable to them. In addition to the expense in time and human expertise of manually encoding a signature for each and every known attack

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. 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. Machine learning and data mining algorithms are used on a large set of such data (e.g., "training data") to build detection models. New data (e.g., "sensor data") gathered by sensors is evaluated by a detector using the detection model. This model determines whether or not the sensor data is intrusive. These models have been proven to be very effective. (See, W. Lee, S. J. Stolfo, and K. Mok, "Data Mining in Work Flow Environments: Experiences in Intrusion Detection," *Proceedings of the* 1999 *Conference on Knowledge Discovery and Data Mining* (*KDD*-99), 1999; and Christina Warrender, Stephanie Forrest, and Barak Pearlmutter, "Detecting Intrusions Using System Calls: Alternative Data Models," *Proceedings of the* 1999 *IEEE Symposium on Security and Privacy*, pages 133–145. IEEE Computer Society, 1999).

These algorithms are generally classified as either misuse detection or anomaly detection. 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 (See, 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.) 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 learn a model of normal activity by training on a set of normal data. Anomaly detection models compare sensor data to normal patterns learned from the training 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. (See, e.g., D. E. Denning, "An Intrusion Detection Model," *IEEE Transactions on Software Engineering*, SE-13:222–232, 1987; T. Lane and C. E. Brodley, "Sequence Matching and Learning in Anomaly Detection for Computer Security," *Proceedings of the AAAI*-97 *Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 43–19. Menlo Park, Calif.: AAAI Press, 1997; Christina Warrender, Stephanie Forrest, and Barak Pearlmutter, "Detecting Intrusions Using System Calls: Alternative Data Models," *Proceedings of the* 1999 *IEEE Symposium on Security and Privacy*, pages 133–145. IEEE Computer Society, 1999; and T. Lane and C. E. Brodley, "Temporal Sequence Learning and Data Reduction for Anomaly Detection," *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 150–158, 1998.) Anomaly detection models are popular because they are seen as a possible approach to detecting unknown or new attacks. 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.

As discussed above, 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 proms 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.

The work most similar to adaptive model generation is a technique developed at SRI in the Emerald system. (See, e.g., H. S. Javitz and A. Valdes, "The NIDES Statistical Component: Description and Justification," *Technical Report, SRI International,* 1993.) 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. perform adaptive real time anomaly detection by using inductively generated sequential patterns. (See, H. S. Teng, K. Chen and S. C. Lu, "Adaptive Real-Time Anomaly Detection Using Inductively Generated Sequential Patterns," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 278–284, Oakland, Calif., May 1990.) Also relevant is Sobirey's work on adaptive intrusion detection using an expert system to collect data from audit sources. (See, M. Sobirey, B. Richter and M. Honig, "The Intrusion Detection System Aid, Architecture and Experiences In Automated Audit Analysis," *Proc. of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security*, pages 278–290, Essen, Germany, 1996.)

Many different approaches to building anomaly detection models have been proposed. A survey and comparison of anomaly detection techniques is given in Christina Warrender, Stephanie Forrest and Barak Pearlmutter, "Detecting Intrusions Using System Calls: Alternative Data Models," *Proceedings of the* 1999 *IEEE Symposium on Security and Privacy*, pp. 133–145, IEEE Computer Society, 1999. Stephanie Forrest presents an approach for modeling normal sequences using look ahead pairs (See, Stephanie Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaf, "A Sense of Self For UNIX Processes," *Proceedings of the* 1996 *IEEE Symposium on Security and Privacy*, pp. 120–128, IEEE Computer Society, 1996) and contiguous sequences (See, S. A. Hofmeyr, Stephanie Forrest, and A. Somayaji, "Intrusion Detection Using Sequences of System Calls," *Journal of Computer Security,* 6:151–180, 1998). Helman and Bhangoo present a statistical method to determine sequences which occur more frequently in intrusion data as opposed to normal data. (See, 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.) Lee et al. uses a prediction model trained by a decision tree applied over the normal data. (See, W. Lee and S. J. Stolfo, "Data Mining Approaches For Intrusion Detection," *Proceedings of the Seventh USENIX Security Symposium,* 1998; and W. Lee, S. J. Stolfo, and P. K. Chan, "Learning Patterns From UNIX Processes Execution Traces For Intrusion Detection," *Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56. Menlo Park, Calif.: AAAI Press, 1997.) Ghosh and Schwartzbard use neural networks to model normal data. (See, Anup Ghosh and Aaron Schwartzbard, "A Study in Using Neural Networks for Anomaly and Misuse Detection," *Proceedings of the Eighth USENIX Security Symposium,* 1999.) Lane and Brodley examine unlabeled data for anomaly detection by looking at user profiles and comparing the activity during an intrusion to the activity under normal use. (See, e.g., T. Lane and C. E. Brodley, "Sequence Matching and Learning in Anomaly Detection for Computer Security," *Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 43–19. Menlo Park, Calif.: AAAI Press, 1997; T. Lane and C. E. Brodley, "Temporal Sequence Learning and Data Reduction for Anomaly Detection," *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 150–158, 1998; and 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.)

In intrusion data representation, related work is the IETF Intrusion Detection Exchange Format project ("Internet Engineering Task Force: Intrusion Detection Exchange Format," http://www.ietf.org/html.charters/idwg-charter.html, 2000) and the CIDF effort (S. Staniford-Chen, B. Tung and D. Schnackenberg. "The Common Intrusion Detection Framework (CIDF)", *Proceedings of the Information Survivability Workshop*, October 1998).

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.

What is needed is an architecture to automate the processes of data collection, model generation and data analysis, and to solve many of the practical problems associated with the deployment of data mining-based IDSs.

SUMMARY OF THE INVENTION

It is an object of the invention to provide a system and method for detecting intrusions in a computer system.

It is another object of the invention to provide support for audit data received from a plurality of data sources.

It is a further object of the invention to provide a technique for performing intrusion detecting in real time and forensics off-line.

It is a still further object of the invention to provide an architecture which is extensible and scalable.

These and other objects of the invention, which will become apparent with reference to the disclosure herein, are accomplished by a system and methods for detecting intrusions in the operation of a computer system comprising a sensor configured to gather information regarding the operation of the computer system, to format the information in a data record having a predetermined format, and to transmit the data in the predetermined data format. A data warehouse is configured to receive the data record from the sensor in the predetermined data format and to store the data in a database. such as an SQL database. A detection model generator is configured to request data records from the data warehouse in the predetermined data format, to generate an intrusion detection model based on said data records, and to transmit the intrusion detection model to the data warehouse according to the predetermined data format. A detector is configured to receive a data record in the predetermined data format from the sensor and to classify the data record in real-time as one of normal operation and an attack based on said intrusion detection model. A data analysis engine is configured to request data records from the data warehouse according to the predetermined data format and to perform a data processing function on the data records.

According to a preferred embodiment, the predetermined data format is XML markup language. The data record may comprises information regarding a source of the information, as "meta-data." The detection model generator is configured to periodically generate the detection model. A detection model distributor is configured to receive the detection model from the data warehouse and to transmit the detection model to the detector. The sensor may be a network sensor. Alternatively, the sensor may be a host sensor.

According to one embodiment, the detection model is a probabilistic model. Accordingly, the detection model generator may be configured to generate a parametrization of the probabilistic model. The detector is configured to compute a probability associated with the data record. According to another embodiment, the detection model is a set of support vectors which correspond to a decision boundary in a feature space. The detection model generator may be configured to generate a set of support vectors. The detector may be configured to map a data record to the feature space and determine the location of the data record in the feature space with respect to the decision boundary.

According to one embodiment, the data analysis engine, e.g., a visualization analysis engine, may be configured to display the data in the data warehouse as selected by a request in the predetermined data format. Another data analysis engine, e.g., a data labeling tool, may be configured to label a data record as one of normal operation and an attack. According to another embodiment, the data analysis engine is further configured to append the label data to the data records in the database. Yet another embodiment of the data analysis engine is a feature extractor configured to extract a feature from a single data record or a plurality data records. This data analysis engine may be configured to append the feature data to the data records in the database.

A system for detecting intrusions in the operation of a computer system is disclosed herein which comprises a plurality of sensors configured to gather information regarding the operation of the computer system, to format the information into data records having a predetermined format, and to transmit the data in the predetermined data format. A plurality of detection model generators may be configured to request data records from the data warehouse in the predetermined data format. Each detection model generator is configured to generate a respective intrusion detection model based on said data records, and to transmit the respective intrusion detection model to the data warehouse according to the predetermined data format.

The invention provides a number of advantages, including the support of heterogeneous data, a central storage of data at the data warehouse, and a common data format enables the system to combine data from different sources very easily. 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. Accordingly, multiple different types of data sources, and alert streams, can be algorithmically combined to detect a wider range of attacks, or even coordinated attacks targeting multiple sites.

In accordance with the invention, the object described above have been met, and the need in the art for an effective intrusion detection architecture has been satisfied.

BRIEF DESCRIPTION OF THE DRAWINGS

Further objects, features, and advantages of the invention will become apparent from the following detailed description taken in conjunction with the accompanying figures showing illustrative embodiments of the invention, in which:

FIG. **1** is a block diagram illustrating the architecture of the system in accordance with the present invention.

FIG. **2** is an exemplary user interface in accordance with the present invention.

FIG. **3** is another exemplary user interface in accordance with the present invention.

Throughout the figures, the same reference numerals and characters, unless otherwise stated, are used to denote like features, elements, components, or portions of the illustrated embodiments. Moreover, while the subject invention will now be described in detail with reference to the figures, it is done so in connection with the illustrative embodiments without departing from the true scope and spirit of the invention as defined by the appended claims.

DETAILED DESCRIPTION OF EXEMPLARY EMBODIMENTS

The system in accordance with the present invention (also referred to as "Adaptive Model Generation" or AMG) is an architecture which provides and automates many of the critical processes in the deployment and operation of real time data mining-based intrusion detection systems. The system 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. The system 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.

The system 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. A component

of the data management capabilities of the system 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, the system has the following major capabilities: Automated data collection is provided, in which 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 capabilities is provided by a data warehouse component that stores data from all sensors. This data is used for training detection models, but also may be used to support various types of data analysis such as forensic analysis. Automated model generation trains detection models 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 permits the handling of data from different types of audit sources. This capability is allowed since any form of data can be represented using a general XML-based language.

Automated model distribution is provided. Once a new model is generated, it may be deployed to all of the detectors that subscribe to the particular detection models that have been generated. The system enables evaluation of archival records stored within the data warehouse to search for intrusions. The system includes generic visualization components that allow in a consistent fashion the visualization of data from different sources. The system also provides correlation support. Since data from multiple sensors is stored in the data warehouse, the system can perform analysis over the data from multiple sources and to train detection models which examine audit streams from multiple sensors.

The system is designed to be extensible and scalable, and hence any audit source, any supervised learning module, any unsupervised (anomaly detection) learning model may be easily inserted into the architecture. In addition, any correlation function or algorithm may be easily inserted to combine information from multiple sources to provide better coverage of attacks, be they coordinated distributed attacks, or clever new attacks against multiple targets within the protected LAN. Moreover, the architecture design also allows for scalability in large networks, and high bandwidth environments. The system architecture can easily be distributed to multiple processors to increase the speed of operation, and it can utilize a high capacity data warehouse (e.g., a parallel or distributed Relational Database Management System (RDBMS)).

An exemplary embodiment of the adaptive model generation system **10** is illustrated in FIG. **1**. System **10** automates the processes of data collection, data management, detection model generation and distribution, and provides various capabilities for data analysis. 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 (e.g., 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 challenge in automating these processes is the need to support these different types of data and different types of detection models. The methods for building these detection models as well as executing them in real time vary greatly for each type of detection model.

The system **10** is a system framework and architecture that can handle virtually any data type and detection model. The system **10** consists of four types of components: (1) real time components which include sensors and detectors, (2) a data warehouse component, (2) detection model management components which include adaptive model generators and detection model distributors, and (4) data analysis components which includes components for visualization, forensics, labeling data, correlation and extracting information from multiple records.

Sensors **12** gather information from an environment and send that data to the data warehouse **14**. The data in the data warehouse **14** is accessed by the detection model generators **16** which generate models that classify activity as either malicious or normal. Once a model is created, it is stored in the data warehouse **14**. A model distributor **18** deploys that model to a real-time detector **20**. The detector **20** receives the detection model from the detection model distributor **18** and also receives the audit data from the sensor **12**. Detector **20** then uses the model to evaluate the audit data from the sensor **12** to detect intrusions.

The data analysis engines **22**, **24**, **26**, and 28 retrieve data from the data warehouse **14**. 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 **14**, or displayed directly to the user. Data analysis engines **22**, **24**, **26**, and **28** allow the system **10** 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 system.

The system uses a distributed architecture that consists of autonomous components. By making these components independent, and linking them only with a communication protocol with very loose format restrictions, any component conforming to that protocol may interface with the system. The communication protocol uses an XML markup language which is substantially identical to the IDMEF specification in the IETF (See, "Internet Engineering Task Force. Intrusion Detection Exchange Format," as published on http://www.ietf.org/html.charters/idwg-charter.html, 2000). The system can easily be adapted to the IDMEF format or any other language format, such as, e.g., CIDF (See, S. Staniford-Chen, B. Tung, and D. Schnackenberg. "The Common Intrusion Detection Framework (CIDF)," *Proceedings of the Information Survivability Workshop*, October 1998.)

The system **10** uses two basic components to provide real time detection capabilities: sensors **12** and detectors **20**. A sensor **12** 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 **20** 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 **20** generates an alert.

In this framework, a traditional IDS system consists of a sensor **12** and a detector **20** which in most cases are integrated together. For example, in an exemplary embodiment, the Snort system contains a sensor which monitors packets and a detector which evaluates signatures over the features extracted from the packets. (Snort is described in M. Roesch, "Snort—Lightweight Intrusion Detection For Networks," *Proceedings of Lisa* '99, 1999 and in U.S. application Ser. No. 10/327,811 filed Dec. 19, 2002 claiming priority to U.S. Provisional Application 60/342,872, filed

Dec. 20, 2001, entitled, "System and Methods For Detecting A Denial-of-Service Attack On A Computer System," which is incorporated by reference herein.) The system **10** separates these two components, thereby 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 system **10** over traditional system architectures is the integration of the sensors **12** and detectors **20** with other analysis and distribution sub-systems. In addition to sending the data to a detector **20**, the sensor **12** also sends the data to a data warehouse **14** which aggregates and stores the data. The detector **20** retrieves its detection models from the model distributors **18**. These detection models are created by other components of the system **10**. Sensors **12** may be lightweight processes that gather information from a real-time audit stream, format the information into an XML representation, and then send the formatted data to the detectors **20** for real time intrusion detection and to the data warehouse **14** for aggregation of data and storage. The sensors **12** can gather information from any source. For every different audit stream of data, a sensor **12** can be implemented to monitor and collect the data.

Typically, there are two major classes of sensors **12**, network sensors and host sensors. Network sensors monitor the packets moving through the network. In the exemplary embodiment, a network sensor such as HAUNT (See, 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) is implemented, and which is used to monitor network activity. The system **10** listens to network traffic and reassembles the packets in order to create data records which summarize connections. A network sensor in the system **10** can be created from existing network IDS systems such as, e.g., Snort and Bro (See, V. Paxson, "Bro: A System for Detecting Network Intruders in Real Time, "*7th Annual USENIX Security Symposium,* 1998) by wrapping the component in the XML-based form required for the system. (As is known in the art, "wrapping" refers to the process of taking input and output of database, and adding meta information to. For example, wrapping is performed by adding software which receives an XML command, translates it to be understood by a MYSQL database, and also operates in the opposite direction, e.g., both input and output from the database.)

In the exemplary embodiment, there are multiple sensors **12** for each host. This configuration may be implemented because there are several different streams of information that are useful for detecting intrusions. The system has been implemented with a variety of sensors for both WindowS™ and Linux systems. For a Windows™ system, there are Windows™ event log sensors, software wrappers, netatat sensors, and registry sensors. There are 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. Netatat sensors use the netatat 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 the system 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 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.

The sensors **12** 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 may be performed differently for streams of different types of data. Therefore, a separate BAM may be needed for each source of data that the system monitors. Packet sniffers and Win32 hooks, as known in the art, 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 works. This makes the system very flexible with respect to sensors.

The communication engine takes the data from the BAM, encodes the data into the system XML format and then sends that data to the data warehouse **14**. 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. The following is the raw data being read by sensor

Process: IEXPLORE
Key: HKCR\Applications\notepad.exe\shell
Query: Openkey
Result: Success
Response: OxE22FC4C0

The sensor **12** sends the following Data sent to Communication Engine:

Process: IMPLORE
Key: HKCR\Applications\notepad.exe\shell
Query: Openkey
Result: Success
Response: OxE22FC4C0
Time: Tue July 31 14:43:20 EDT 2001
ProcQuery: 1263.4353
KeyProc: 6784.93633
QueryKey: 6621.3521
KeyResponse: 4510.2431
KeyResVal: 8743.3245

In the exemplary embodiment, 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> OxE22FC4C0 </response> <procQuery> 1263.4353 </procQuery> <keyProc> 6784.9363 <keyProc> <queryKey> 6621.3521 </queryKey> <keyResponse> 4510.2431 </keyResponse> <keyResval> 8743.3245 </keyResVal>

Detectors **20** analyze audit stream data collected from a sensor and detect intrusions by using a detection model. A detector **20** 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 may have a different detector that implements model evaluation for that model type.

The detector **20** may 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 the exemplary embodiment, 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. (Further details of a decision-tree detection model are given in U.S. application Ser. No. 10/208,402 filed Jul. 30, 2002, entitled "System and Methods For Intrusion Detection With Dynamic Window Sizes", which is incorporated by reference in its entirety herein.) 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. (Further details of a probabilistic model are given in U.S. application Ser. No. 10/208,432, filed Jul. 30, 2002, entitled "System and Methods For Detection of New Malicious Executables," which is incorporated by reference in its entirety herein.) 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. Further details on the support vector machine model generation and detection are provided below and in U.S. application Ser. No. 10/320.259, filed Dec. 16, 2002, entitled "Methods of Unsupervised Anomaly Detection Using a Geometric Framework," which is incorporated by reference in its entirety herein.

Detectors **20** receive detection models from model distributors **18**, which distribute models stored in the data warehouse originally created by model generators. The detectors **20** receive real time updates from the model distributors **18**. This process keeps the detection models updated as soon as new models are available. Below is an example of a model that the detector **20** for the RAD system (described below and in U.S. application Ser. No. 10/352, 343 concurrently filed, entitled "System and Methods for Detecting Intrusions in a Computer System by Monitoring Operating System Registry Accesses," 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\Applications, . . . </values> </feature>
</encode> </model>

The encoding of this model is explained in greater detail herein. The evaluation of the record shown above with this model would result in a normal label for the record.

The data warehouse **14** is described herein. It serves as the central storage repository for all of the data collected by the sensors **12**. The model generators **16** access the data in the data warehouse **14** and create detection models using this data. The data warehouse **14** also stores these detection models. The analysis engines **22**, **24**, **26**, and 28 also access the data stored in the data warehouse **14**. These components give the system visualization and forensics capabilities, as described below.

The core of the data warehouse **14** is a Structured Query Language (SQL) database. This allows for easy manipulation of the data, which allows for the creation of training data sets to build data mining-based detection models. Since arbitrary subset of data can be retrieved using an SQL query, the data warehouse automates the process of manually creating these data sets. This flexibility is very important in practical deployments of the system **10**.

For example, if there are 40 Windows™ hosts in a network, in order to create an anomaly detection model over the application Event logs for each of the hosts in the system framework, the following steps must be performed. A sensor **12** is installed on each of the hosts. This sensor collects the data and stores it in the data warehouse **14**. If each host is typically used in the same way, a large data set may be created containing the combined event logs from each of the hosts. Alternatively, if each host is used differently, a separate training set may be created for each individual host. Since the data warehouse uses a SQL database, different data sets may be created by issuing different queries.

Storing the data in a single repository has several other advantages for correlating sensor outputs. For example, since the data is stored in a SQL database, "join" statements may be used to facilitate the linking of records from different sensors into single records. In addition, data may be obtained from two sensors relatively easily because all of the data is stored in the same database.

The data warehouse **14** 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, and is described in greater detail in the Appendix. The format for an insert transaction is displayed below.

<command>

<tablename>

<begin> <rec> <vari varitype> valueA </vari> <var2 var2type> valueA </var2> <var3 var3type> valueA </var3> . . . . . <varN varNtype> valueA </varN>

</rec>

<rec> <vari varitype> valueB </vari> <var2 var2type> valueB </var2><var3var3type> 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 technique greatly reduces the cost in many cases when there are many records being sent to the database **14** by a sensor. When the data warehouse **14** 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 a HAUNT network sensor.

<insert>

<nfri>

<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 **14** 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 nfri where the data is to be stored. Then it opens its first record with an opening <rec> tag. Each variable is sent in order 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, dst-bytes, also an integer **490**, is sent. Then each variable is sent sequentially until the entire record is sent to the data warehouse. For convenience, the types of data integer, float, and string are abbreviated with i, f, and str respectively. The exemplary coding of the data warehouse **14** is provided in routines dw.cfg, class1.java, DBAaccess.java, and wrapper threadx-.java, which are appended hereto, as discussed in the Appendix.

The system **10** manages the creation and distribution of detection models. The detection models are generated by the detection model generators **16** using data collected stored in the data warehouse **14**. They are distributed to the detectors **20** by the model distributors **18**, as described herein.

The system **10** is designed to work with any model generation algorithm. Thus, the model generator components **16** can be viewed as modular components that are "plugged" into the architecture. 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 the architecture of system **10**, the model generators **16** may select any data through the use of general or specific queries. Accordingly, the architecture is robust enough to handle any type of model generation algorithm.

The model generation modules **16** request the data from the data warehouse **14** when they want to create a model. They form their request based on the information that the model needs to train on. The generator **16** then runs and creates a model. This model is then encoded into XML and sent to the data warehouse **14**. Model generators **16** also signal the model distributor **18** 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\Applications, ... </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 which follows the version information. The exemplary 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.

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. In the exemplary embodiment, detectors **20** may not continuously check for updates in the data warehouse **14**, because this may be inefficient, and the real-time requirements of the system **10** as a whole depend on the detectors **20** being lightweight components. The model distributors **18** are used to automatically send model updates to the detectors **20** whenever the model generators **16** create them. The exemplary coding of the detection model distributor **18** is provided in routines constants.java, DWclient.java, and MDcommunicator.java, which are appended hereto, as discussed in the Appendix.

An analysis engine is any component that takes as its input a set of data from the database **14** and performs analysis over this data. The analysis engines have the capability of inserting the results of the analysis into the database. In the exemplary embodiment, the analysis engines **22**, **24**, **26** and **28** query the data warehouse **14** for a set of data, and then insert new information into the data warehouse **14** using the SQL interface. This can be useful for several purposes. For example, the data analysis engines use the same XML format that the rest of the system uses, with some specific tags designed specifically for data analysis. In the exemplary embodiment, four types of analysis engines

have been implemented: a visualization client **22**, a forensics tool **24**, a data labeling tool **26**, and a feature extractor **28**.

The visualization analysis engine **22** provides a system administrator with a mechanism to view all of the data in the data warehouse **14**. An example of a user interface **50** provided by visualization agent implemented by the adaptive model generation system is displayed in FIG. **2**, which indicates the date **52**, user **54**, computer **56**, the operating system action which is being logged **58** (e.g., **592** a new process has been created; **528** successful logon; or **529** logon failure, unknown user name or bad password), category **60** (a code indicative of 'success' or 'failure'), type **62**, and strings of data **64**, **66**, and **68**. Action **58**, category **60** and type **62** are defined by the Microsoft™ logger which writes to the registry

The visualization analysis engine **22** is integrated with the database **14** which allows the use of SQL queries to filter the data to be viewed. An example of this interface and a SQL query is shown in FIG. **3**, which illustrates an interface to the underlying database and asks how many different values of 'keys' accessing the registry during the first 1000 records.

The visualization tool 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, she 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.

For example, efficient sensor networks for detecting tasks may be designed. The correlation and visualization capability might find, for example, that certain types of attacks are best detected by one source of data (e.g., a host-based detector) while another set of attacks are best detected by another (e.g., a network based sensor/detector). Hence, this architecture also provides the means of analyzing the overall detection task to architect a detection system combining components (sensors/detectors) in a manner that is both efficient and effective.

Another type of data analysis is forensic analysis. A forensic system **24** retrieves a set of historical data from the data warehouse **14**. Typically, the data of interest is a set of data which is suspected to contain intrusions. The tool must retrieve a specific set of data appropriate to the algorithm in question. Once the data set is retrieved, the forensics analysis engine **24** may apply a detection 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 **14** 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> <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> <arc str>128.59.22.69 </src> <dst str> 12.59.22.121 </dst> <ip-overlap str> 0</ip-overlap> </rec>

The output sent back to the data warehouse **14** 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> <arc 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> <arc str> 128.59.22.69 </src> <dst str> 12.59.22.121 </dst> <ip-overlap str> 0 </ip-overlap> <label str>normal </label></rec>

Forensics can be done with misuse detection models if there is a learned detection model for that data. If a learned detection model exists, it can be run over the data to 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, there is no appropriate model to perform forensics. In these cases, an unsupervised anomaly detection algorithm may be used 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 greater detail below.

Another data analysis engine is the data labeling tool **26**. The data labeling tool **26** takes the list of known attacks and uses that information to label all of the records in the database **14** which corresponds to these attacks. The labeling tool **26** is used to create labeled training data. The list of known attacks could be, e.g., 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 **26** is a significant improvement over the difficult manual labeling of records in a database. The manual labeling of data is a significant cost for deploying a data mining-based intrusion detection system. This cost is cut significantly through the use of this data labeling tool **26**.

The data labeling tool **26** is implemented using SQL joins with the sensor data in the data warehouse **14** and the attack list. For example, given a table full of Windows™ host-based information from the application log, all actions in the application log are stored in the data warehouse **14** with all available information from the log, including process name. Given an attack list of all process names corresponding to attacks, the labeling tool **26** can automatically insert that attack list into the data warehouse **14** 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. Labeling collected data is necessary to create training data for data mining-based detection models. 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> . . . queryKey </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>

. . . . .

Features are important discriminating attributes derived from raw audit data that are employed in detection models. A feature extractor **28** 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 is a more general version of the forensic analysis engine.

Many features may be computed by using information that spans several individual records. Frequently, records by themselves are not meaningful, but in combination with other records they could represent an attack. The data warehouse **14** has the capability to provide the feature extractor **28** 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 **14**. A column may be 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 example shows features extracted from three records. Features could also be extracted from any number of records. This example shows 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> <arc str>128.59.22.69 </arc> <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>

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. Models learned over well-computed features are generally far superior to those computed over raw pieces of information.

Feature extractors **28** can be seen as data analysis engines by the system. They retrieve data from the data warehouse **14** and then perform computations on that data. Once these computations are completed the new data is sent back to the warehouse **14** 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 configuration requires that a specialized feature extractor must be made for each sensor. 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.

Another 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 system which is a network intrusion detection system. The deployment of the HAUNT system in the system framework is described in greater detail below.

The system **10** described above has a distributed architecture to allow for the automation of the data collection and data warehousing. In the system framework, simply deploying a sensor will automatically collect and aggregate that sensors data in the data warehouse. This is desirable in order to aggregate the data. For example, the data analysis may include 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.

The distributed architecture of system **10** allows the system to gather data from heterogeneous systems. A set of standard guidelines in a flexible format are placed on sensor data, as described in the Appendix. There are many different types of information that IDSs use, such as e.g., 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 in system **10** because all of the data gathered by the system is transmitted to the data warehouse **14** using the XML mark up language. The system is flexible enough to store all types of information.

In a typical network environment, the system permits the deployment of many models throughout the network. These models can also become out of data. Accordingly, the system has a mechanism for the creation and the management of detection models. The models are created using the detection model generators **16**, described above. The detection models are then stored in the data warehouse **14**. The data warehouse **14** is robust enough to handle any types of models, and therefore the system can be used with any types of models. The data warehouse **14** is also stable enough that failure of model storage is not a concern while the protected machine is under attack. The use of model distributor **18** allows the system **10** to update and alter models on the fly with a minimal computational overhead. This is very advantageous because it allows the system **10** to be deployed for a long period of time without the need for maintenance by an administrator.

The flexibility of XML is used here for the management functions. All data (models, sources, etc.) are described easily in the XML framework by "meta-data" that describes the source and its relationship to other sources and components. This framework is used to provide management functionality. This coupled with a visualization tool, e.g. visualization analysis engine **22**, greatly aids the analysis of data.

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. By eliminating dependencies between sensors, model generators, and detectors, the system has enabled correlation algorithms to be constructed the same as any other algorithm. The distributed architecture and the data warehouse allow implementation of correlation algorithms with no additional implementation overhead. The data warehouse allows for the retrieval 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.

For example, a correlation algorithm operates in the following manner: given a training set of labeled data, each of detection models is run over the data. A probability table can be created to show which detection model has a probability of being correct given the outputs of the other models. For example, an exemplary system uses ten detection models, which are looking for an attack A in a given operation or process. In this case, models **1**–**9** of the detection models may label the operation or process as an attack, but model **10** does not label the operation or process as an attack. Accordingly, it is possible to calculate the probability that model **10** is correct. As a result, when model **10** is used to predict an attack, there is a statistic to predict the degree to which the model is believed to be accurate.

There are three exemplary types of model generation algorithms that the system **10** 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.

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. 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 system, the cost of labeling the data can be minimized. Once the sensors are deployed into a network, simulated attacks can be run and the time stamps and other information about the attack recorded. 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, the attack data can be labeled. This labeled data is now stored in the data warehouse and can be retrieved by the model generators. These models can also be distributed into the detectors using the model distributors.

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 operate on the principle that attacks are 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, it is possible 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.

Unsupervised anomaly detection algorithms examine unlabeled data and attempt to detect intrusions buried within the unlabeled data. Unsupervised anomaly detection algorithms operate under the principle that intrusions are very rare compared to the normal data and they are also quantitatively different. Because of this, intrusions are outliers in the data and can be detected.

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 an 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.

One specific type of model generation algorithm used by the system is Support Vector Machines (SVMs). This algorithm can be used for both Unsupervised Anomaly Detection and normal Anomaly Detection, and is described in greater detail in U.S. application Ser. No. 10/320,259 filed Dec. 16, 2002, entitled "Methods of Unsupervised Anomaly Detection Using a Geometric Framework," which is incorporated by reference above. The SVM algorithm is a binary classifier. The operating principle for an SVM approach to intrusion detection is that data is mapped to a feature space. Inside this feature space, the SVM and a set of labeled training data are used 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 known in the art (See, e.g., N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.)

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

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

Solving the following optimization problem results in a solution the solution to the SVM optimization.

maximize:

$$\sum_i a_i - \frac{1}{2} \sum_{i,j} a_i a_j y_i y_j \langle x_i, x_j \rangle$$

subject to:

$$0 \le a_i, \sum_i a_i y_i = 0$$

Setting b=0, the solution is then:

$$w = \sum_i a_i y_i x_i$$

All x_i with $a_i \neq 0$ are called the support vectors. These are the vectors on the border 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, kernel functions may be used 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 $\Phi: X \rightarrow F$. 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.

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

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

maximize:

$$\sum_i a_i - \frac{1}{2} \sum_{i,j} a_i a_j y_i y_j K(x_i, x_j)$$

subject to:

$$0 \le a_i, \sum_i a_i y_i = 0$$

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:

$$f(x) = \mathrm{sgn}\left(\sum_{i=1}^{N_3} a_i y_i K(s_i, x) + b\right)$$

where N_3 is the number of support vectors and s_i is the ith support vector.

The standard support vector machine algorithm is used for misuse detection by the system. Data in the form of vectors of real numbers are sent from the sensors **12** to detectors **20**. The detectors **20** use a SVM model to differentiate between normal data and intrusion data.

To implement this in the system, training data must first be generated. A system is monitored by sensors **12** that send their observations to the data warehouse **14** 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 **14** as either normal or attack. This labeled data is then sent via XML to the model generator **16**. The model generator **16** 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 **14** for storage and to all of the detectors **20** that use this kind of model. Once the model is in place, sensors **12** send data that they are monitoring to the detector **20** for classification by the SVM classification rule.

The standard SVM algorithm is a supervised learning algorithm. It requires labeled training data to create its classification rule. An unsupervised variant does not require its training set to be labeled to determine a decision surface (See, e.g., 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.

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 claiss +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 $x_i \in \Re^l$ minimize:

$$\frac{1}{2}\sum_{i,j} a_i a_j K(x_i x_j)$$

subject to:

$$0 \le a_i \le \frac{1}{vl}, \sum_i a_i = 1$$

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:

$$f(x) = \mathrm{sgn}\left(\sum_{i=1}^{N_3} a_i K(s_i, x) - b\right)$$

where N_3 is the number of support vectors and s_i is the ith support vector. For this algorithm b cannot be set to 0, it must be found explicitly.

$$b = \sum_{j=1}^{N_3} a_j K(s_j, s_i)$$

The unsupervised SVM variant proposed by Schölkopf can be used for unsupervised anomaly detection. This approach was described in U.S. application Ser. No. 10/320,259, filed Dec. 16, 2002, entitled "Methods of Unsupervised Anomaly Detection Using a Geometric Framework," incorporated by reference above, perform unsupervised anomaly detection using the system framework. The algorithm differentiates between normal data and anomalous data. Anomalous data is thought to be intrusion data because intrusions are much different than normal system use.

Like the misuse detection algorithm, unsupervised anomaly detection requires training data. During a training period, a system is run normally with no attacks. Sensors **12** monitoring the system send their observations via XML to the data warehouse **14**. 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 **14** to the model generator **16** via XML. There is no need to label data. The model generator **16** 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 **14** for storage and to the appropriate detectors **20**. Once the detection model is in place, sensors **12** send data to the detector **20** for classification.

The system framework can support a variety of different intrusion detection systems. One example of an IDS system that is integrated into the system is the Registry Anomaly Detection (RAD) system, which is described in greater detail in U.S. application Ser. No. 10/352,343, concurrently filed, entitled "System and Methods for Detecting Intrusions in a Computer System by Monitoring Operating System Registry Accesses," incorporated by reference above. 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.

The RAD system uses several 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 structure 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.

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 (as described in "SysInternals. Regmon for Windows NT/9×," Online publication, http://www.sysinternals.com/ntw2k/source/regmon.shtml,2000). 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 the XML format and sends it to the data warehouse **14**. The communication module can support 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 five composite features that are used by the RAD system are examples of feature extraction. The RAD system uses 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.

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 as described in U.S. application Ser No. 10/352,343 concurrently filed, entitled "System and Methods for Detecting Intrusions in a Computer System by Monitoring Operating System Registry Accesses."

Each feature is individually evaluated to be either normal or anomalous. Then the statistics we gathered are used to score these anomalies. This score is based on how likely it is that the value of this feature will be different than 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. Any algorithm could be used without changing 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. Accordingly, the above is provided as an exemplary embodiment of the present invention.

In order to detect anomalies in real time, a detector **20** was implemented for the RAD system. Although this detector was implemented specifically for the RAD system, it could be used to evaluate any model that was created by the classification algorithm described above. The first requirement of the detector is that it 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 **14**, 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 novel system described herein. With no changes in architecture, the system can support any number of host machines and sensors. With a conventional system architecture, increasing the multiplicity would require major changes in the structure of a system. According to the invention, the central data collection is automated in the system. This means that data from multiple machines is gathered in the same place and can be analyzed from that central location.

Another exemplary implementation of the system is the Heuristic Audit of Network Traffic (HAUNT) system, which 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.

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 **14**. 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, Network Flight Recorder Inc. Network flight recorder, 1997, described at http://www.nfr.com, and Snort, described above. 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.

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 cannot confidently make a classification, 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.

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 model generation program, as is known in the art. 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.

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 a 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, to calculate the total number of tcp connections, the following command could be used:

num_tcp_connections=SUM (protocol==tcp)

The SUM(protocol==tcp) returns the total of number of records of which the condition service==http is true. To calculate the total number of tcp connections going to port **2301**, the following command is used:

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, the following command could be used:

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 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.

It will be understood that the foregoing is only illustrative of the principles of the invention, and that various modifications can be made by those skilled in the art without departing from the scope and spirit of the invention.

We claim:

1. A system for detecting intrusions in the operation of a computer system comprising:

(a) a sensor configured to gather information regarding the operation of the computer system, to format the information in a data record having a predetermined format, and to transmit the data in the predetermined data format;

(b) a data warehouse configured to receive the data record from the sensor in the predetermined data format, aggregate the data, store the data in a SQL database, and to store an intrusion detection model;

(c) a detection model generator configured to request data records from the data warehouse in the predetermined data format, to generate the intrusion detection model based on said data records, and to transmit the intrusion detection model to the data warehouse according to the predetermined data format;

(d) a detector configured to receive a data record in the predetermined data format from the sensor and to classify the data record in real-time as one of normal operation and an attack based on said intrusion detection model; and

(e) a data analysis engine configured to request data records from the data warehouse according to the predetermined data format and to perform a data processing function on the data records.

2. The system according to claim **1**, wherein the sensor is a network sensor.

3. The system according to claim **1**, wherein the sensor is a host sensor.

4. The system according to claim **1**, wherein the data record comprises information regarding a source of the information.

5. A system for detecting intrusions in the operation of a computer system comprising:

(a) a sensor configured to gather information regarding the operation of the computer system, to format the information in a data record having a predetermined format, and to transmit the data in the predetermined data format;

(b) a data warehouse configured to receive the data record from the sensor in the predetermined data format and to store the data in a SQL database;

(c) a detection model generator configured to request data records from the data warehouse in the predetermined data format, to generate an intrusion detection model based on said data records, and to transmit the intrusion detection model to the data warehouse according to the predetermined data format;

(d) a detector configured to receive a data record in the predetermined data format from the sensor and to classify the data record in real-time as one of normal operation and an attack based on said intrusion detection model; and

(e) a data analysis engine configured to request data records from the data warehouse according to the predetermined data format and to perform a data processing function on the data records,

wherein the data analysis engine is configured to label a data record as one of normal operation and an attack.

6. The system according to claim **5**, wherein the predetermined data format is XML markup language.

7. The system according to claim **5**, wherein the detection model generator is configured to periodically generate the detection model, the system further comprising a detection

model distributor configured to receive the detection model from the data warehouse and to transmit the detection model to the detector.

8. The system according to claim **5**, wherein the detection model is a probabilistic model.

9. The system according to claim **8**, wherein the detection model generator is configured to generate a parametrization of the probabilistic model.

10. The system according to claim **9**, wherein the detector is configured to compute a probability associated with the data record.

11. The system according to claim **5** wherein the detection model is a set of support vectors which correspond to a decision boundary in a feature space.

12. The system according to claim **11**, wherein the detection model generator is configured to generate a set of support vectors.

13. The system according to claim **12**, wherein the detector is configured to map a data record to the feature space and determine the location of the data record in the feature space with respect to the decision boundary.

14. The system according to claim **5**, wherein the data analysis engine is configured to display the data in the data warehouse as selected by a request in the predetermined data format.

15. The system according to claim **5**, wherein the data analysis engine is further configured to append the label data to the data records in the SQL database.

16. The system according to claim **5**, wherein the data analysis engine is configured to extract a feature from a plurality data records.

17. The system according to claim **16**, wherein the data analysis engine is configured to append the feature data to the data records in the SQL database.

18. A distributed system for detecting intrusions in the operation of a computer system comprising:

(a) a plurality of sensors configured to gather information regarding the operation of the computer system, to format the information in a data record having a predetermined format, and to transmit the data in the predetermined data format;

(b) a data warehouse configured to receive the data record from the sensor in the predetermined data format, aggregate the data, store the data in a SQL database, and to store an intrusion detection model;

(c) a plurality of detection model generators configured to request data records from the data warehouse in the predetermined data format, each detection model generator configured to generate the respective intrusion detection model based on said data records, and to transmit the respective intrusion detection model to the data warehouse according to the predetermined data format;

(d) a plurality of detectors configured to receive a data record in the predetermined data format from the sensor and to classify the data record in real-time as one of normal operation and an attack based on a respective intrusion detection model; and

(e) a data analysis engine configured to request data records from the data warehouse according to the predetermined data format and to perform a data processing function on the data records.

19. The system according to claim **18**, wherein the sensor is a network sensor.

20. The system according to claim **18**, wherein the sensor is a host sensor.

21. The system according to claim **18**, wherein the data record comprises information regarding a source of the information.

22. A distributed system for detecting intrusions in the operation of a computer system comprising:

(a) a plurality of sensors configured to gather information regarding the operation of the computer system, to format the information in a data record having a predetermined format, and to transmit the data in the predetermined data format;

(b) a data warehouse configured to receive the data record from the sensor in the predetermined data format and to store the data in a SQL database;

(c) a plurality of detection model generators configured to request data records from the data warehouse in the predetermined data format, each detection model generator configured to generate a respective intrusion detection model based on said data records, and to transmit the respective intrusion detection model to the data warehouse according to the predetermined data format;

(d) a plurality of detectors configured to receive a data record in the predetermined data format from the sensor and to classify the data record in real-time as one of normal operation and an attack based on a respective intrusion detection model; and

(e) a data analysis engine configured to request data records from the data warehouse according to the predetermined data format and to perform a data processing function on the data records,

wherein the data analysis engine is configured to label a data record as one of normal operation and an attack.

23. The system according to claim **22**, wherein the predetermined data format is XML markup language.

24. The system according to claim **22**, wherein one of the detection model generators is configured to periodically generate the respective detection model, the system further comprising a detection model distributor configured to receive the respective detection model from the data warehouse and to transmit the detection model to a respective detector.

25. The system according to claim **22**, wherein one of the detection models is a probabilistic model.

26. The system according to claim **25**, wherein one of the detection model generators is configured to generate a parametrization of the probabilistic model.

27. The system according to claim **26**, wherein one of the detectors is configured to compute a probability associated with the data record.

28. The system according to claim **22**, wherein one of the detection models is a set of support vectors which correspond to a decision boundary in a feature space.

29. The system according to claim **28**, wherein one of the detection model generators is configured to generate a set of support vectors.

30. The system according to claim **29**, wherein one of the detectors is configured to map a data record to the feature space and determine the location of the data record in the feature space with respect to the decision boundary.

31. The system according to claim **22**, wherein the data analysis engine is configured to display the data in the data warehouse as selected by a request in the predetermined data format.

32. The system according to claim **22**, wherein the data analysis engine is further configured to append the label data to the data records in the SQL database.

33. The system according to claim **22**, wherein the data analysis engine is configured to extract a feature from a plurality data records.

34. The system according to claim **33**, wherein the data analysis engine is configured to append the feature data to the data records in the SQL database.

35. A method for detecting intrusions in the operation of a computer system comprising:

(a) gathering information regarding the operation of the computer system at a sensor and formatting the information into a data record having a predetermined data format;

(b) transmitting the data record in the predetermined data format to a data warehouse, aggregating the data, and storing the data in a SQL database;

(c) generating an intrusion detection model comprising requesting data records from the data warehouse according to the predetermined data format, transmitting the intrusion detection model to the data warehouse in the predetermined data format, and storing the intrusion detection model at the data warehouse;

(d) classifying a data item in real-time as one of normal operation and an attack based on the intrusion detection model comprising receiving the data records from the sensor; and

(e) requesting a data record from the data warehouse according to the predetermined data format and performing a data processing function on the data record.

36. The method according to claim **35**, wherein the step of gathering information regarding the operation of the computer system at a sensor comprises providing a network sensor.

37. The method according to claim **35**, wherein the step of gathering information regarding the operation of the computer system at a sensor comprises providing a host sensor.

38. The method according to claim **35**, wherein the step of gathering information regarding the operation of the computer system at a sensor comprises formatting the data record with information relating to a source of the information.

39. A method for detecting intrusions in the operation of a computer system comprising:

(a) gathering information regarding the operation of the computer system at a sensor and formatting the information into a data record having a predetermined data format;

(b) transmitting the data record in the predetermined data format to a data warehouse and storing the data in a SQL database;

(c) generating an intrusion detection model comprising requesting data records from the data warehouse according to the predetermined data format and transmitting the intrusion detection model to the data warehouse in the predetermined data format;

(d) classifying a data item in real-time as one of normal operation and an attack based on the intrusion detection model comprising receiving the data records from the sensor; and

(e) requesting a data record from the data warehouse according to the predetermined data format and performing a data processing function on the data record, wherein the step of requesting data from the data warehouse comprises labeling a data record as one of normal operation and an attack.

40. The method according to claim **39**, wherein the step of formatting the data according to a predetermined data format comprises formatting the data according to XML markup language.

41. The system according to claim **39**, further comprising-periodically generating the detection model and transmitting the detection model from the data warehouse to a distributor and transmitting the detection model from the distributor to the detector.

42. The system according to claim **39**, wherein the step of generating an intrusion detection model comprises generating a probabilistic model.

43. The method according to claim **42**, wherein the step of classifying a data record as normal operation or as an attack in real-time comprises computing a probability associated with the data record.

44. The system according to claim **39**, wherein the step of generating an intrusion detection model comprises generating a set of support vectors in a feature space.

45. The method according to claim **44**, wherein the step of classifying a data record as one of normal operation and an attack in real-time comprises mapping a data record to the feature space and determining the location of the data record with respect to the decision boundary.

46. The system according to claim **39**, wherein the step of requesting data from the data warehouse comprises displaying the data record stored in the data warehouse as selected by a request according to the predetermined data format.

47. The method according to claim **39**, comprising appending the label to the data records in the SQL database.

48. The system according to claim **39**, wherein the step of requesting data from the data warehouse comprises extracting a feature from a plurality data records.

49. The method according to claim **48**, further comprising appending the feature data to the data records in the SQL database.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 7,225,343 B1
APPLICATION NO. : 10/352342
DATED : May 29, 2007
INVENTOR(S) : Andrew Honig et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Col. 10, line 58, "KeyProc:6784.93633" should be -- KeyProc:6784.9363 --

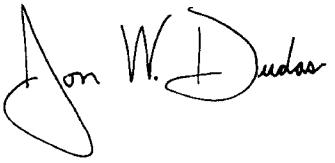
Col. 22, line 50, equation " $K(x_i, x_j) = e^{-\|x_i - {}_j\|^2/2\sigma}$ "

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

Col. 26, line 43, "definition structure and a feature exchange. protocol to" should be -- definition structure and feature exchange protocol to --

Signed and Sealed this

Twentieth Day of January, 2009



JON W. DUDAS
Director of the United States Patent and Trademark Office