A System for Distributed Intrusion Detection

Steven R. Snapp[1], James Brentano[1], Gihan V. Dias[1], Terrance L. Goan[1], Tim Grance[2], L. Todd Heberlein[1], Che-Lin Ho[1], Karl N. Levitt[1], Biswanath Mukherjee[1], Douglass L. Mansur[3], Kenneth L. Pon[3] and Stephen E. Smaha[4]

1. Division of Computer Science, University of California, Davis, California 95616
2. United States Air Force Cryptologic Support Center, San Antonio, Texas 78243
3. Lawrence Livermore National Laboratory, Livermore, California 94550
4. Haystack Laboratories, Inc., Austin, Texas 78729

Abstract

The study of providing security in computer networks is a rapidly growing area of interest because the network is the medium over which most attacks or intrusions on computer systems are launched. One approach to solving this problem is the *intrusion-detection* concept, whose basic premise is that not only abandoning the existing and huge infrastructure of possibly-insecure computer and network systems is impossible, but also replacing them by totally-secure systems may not be feasible or cost effective. Previous work on intrusion-detection systems were performed on stand-alone hosts and on a broadcast local area network (LAN) environment. The focus of our present research is to extend our network intrusion-detection concept from the LAN environment to arbitrarily wider areas with the network topology being arbitrary as well. The generalized distributed environment is heterogeneous, i.e., the network nodes can be hosts or servers from different vendors, or some of them could be LAN managers, like our previous work, a network security monitor (NSM), as well. The proposed architecture for this distributed intrusion-detection system consists of the following components: a host manager (viz. a monitoring process or collection of processes running in background) in each host; a LAN manager for monitoring each LAN in the system; and a central manager which is placed at a single secure location and which receives reports from various host and LAN managers to process these reports, correlate them, and detect intrusions.

1. Introduction

Intrusion detection and *network security* are becoming increasingly more important in today's computer-dominated society. As more and more sensitive information is being stored on computer systems and transferred over computer networks, more and more *crackers* are attempting to attack these systems to steal, destroy or corrupt that information. While most computer systems attempt to prevent unauthorized use by some kind of access control mechanism, such as passwords, encryption, and digital signatures, there are several factors that make it very difficult to keep these crackers from eventually gaining entry into a system [2,4]. Most computer systems have some kind of security flaw that may allow outsiders (or legitimate users) to gain unauthorized access to sensitive information. In most cases, it is not practical to replace such a flawed system with a new, more secure system. It is also the case that it is very difficult, if not impossible, to develop a completely-secure system. Even a supposedly-secure system can still be vulnerable to insiders misusing their privileges, or it can be compromised by improper operating practices. While many existing systems may be designed to prevent specific types of attacks, other methods to gain unauthorized access may still be possible. Due to the tremendous investment already made into the existing infrastructure of *open* (and possibly insecure) communication networks, it is infeasible to deploy new, secure, and possibly *closed* networks [6]. Since the event of an attack should be considered inevitable, there is an obvious need for mechanisms that can detect outsiders attempting to gain entry into a system, that can detect insiders misusing their system privileges, and that can monitor the networks connecting all of these systems together.

Intrusion Detection Systems (IDS) are based on the principle that an attack on a computer system (or network) will be noticeably different from normal system (or network) activity. An intruder to a system (possibly masquerading as a legitimate user) is very likely to exhibit a pattern of behavior different from the nor-

CH2961-1/91/0000/0170$01.00 © 1991 IEEE

mal behavior of a legitimate user. The job of the IDS is to detect these abnormal patterns by analyzing numerous sources of information that are provided by the existing systems. The two major methods used to detect intrusions are *statistical analysis* and rule-based *expert system analysis*. The statistical method attempts to define *normal* (expected) behavior, while the expert system defines *proper* behavior. The expert system also searches for breaches of policy. If the IDS notices a possible attack using either of these methods, then a *System Security Officer* (SSO) is notified. The SSO may then take some action against the aggressor.

The audit trail records provided by a computer system are the main source of information regarding behavior for that system. The IDS must use these records in its analysis of the expected and proper behavior for that system. The two major problems with audit trail analysis are exactly what kind of activity should be audited (i.e., auditable events), and what to do with the huge amount of information that is produced. Since auditing was initially designed for accounting purposes rather than for system security, many desired auditable events may be unavailable to the security system. The audit trail is designed and provided by the vendor of the computer system in question. The amount of information generated by the audit trail mechanism for even a single user can be tremendous, possibly in excess of 10 megabytes of data per user per day [11]. Processing such a huge amount of information, viz. the collection, storage and timely analysis of the records, can become problematic, since overall system performance may become adversely affected. Another continuing source of disagreement is the idea of a so-called *canonical audit record*. The question is whether or not to standardize the kind of information that should be contained in an audit record.

One means of detecting anomalous behavior is to monitor statistical measures of user activities on the system. A popular way to monitor statistical measures is to keep *profiles* of legitimate user activities [2,4]. These profiles may include such items as login times, CPU usage, favorite editor and compiler, disk usage, number of printed pages per session, session length, error rate, etc. (See [3] for a comprehensive list of possible measures.) The IDS will then use these profiles to compare current user activity with past user activity. Whenever a current user's activity pattern falls outside certain pre-defined thresholds, the behavior is considered anomalous. Legitimate behavior that is flagged as intrusive is defined to be a *false alarm*. A major problem with the statistical method is determining exactly what activities and statistical measures provide the highest detection rate and lowest false alarm rate for a particular system. Those statistics that detect an attack on a computer system may differ from system to system depending on the system and its environment; so the measures must be tailored for each particular system. It may also be the case that a particular activity may not be threatening by itself, but when aggregated with other activities, it may constitute an attack. These statistical profiles must be adaptive, i.e., they must be updated regularly, since users may be constantly changing their behavior.

Another means of detecting possible attacks on a computer system is by using a rule-based expert system analysis method. The expert system will analyze the audit trail records, and it will try to determine attacks based on the information contained in the rule base. The expert system is able to pose sophisticated queries to the rule base to answer conditional questions based on sets of events [11]. The main problem with this method is determining exactly what the rules should be, and what kinds of attacks can be detected using this method. As an example, the rule base may contain rules describing known attack methods and signatures, known system flaws, expected system behavior, and the site-specific security policy.

Network intrusions such as eavesdropping, illegal remote accesses, remote break-ins, inserting bogus information into data streams and flooding the network to reduce the effective channel capacity are becoming more common. While monitoring network activity can be difficult because of the great proliferation of (un)trusted open networks, the task is not impossible. The same type of system that can monitor a single host machine may also be able to monitor a *local area network* (LAN) just as effectively [6]. The key here, again, is to attempt to define normal network activity based on past performance or experience, as well as proper network behavior.

The remainder of this paper is a discussion of a proposed Distributed Intrusion Detection project currently under investigation. At the time this paper is published, we expect to have an initial prototype of the above distributed monitor up and running.

2. Distributed Intrusion Detection System (DIDS)

2.1. Introduction

Our previous work concentrated on the development of an intrusion-detection model and a prototype implementation of a network security monitor (NSM) for a broadcast local area network environment [6]. The NSM (adaptively) develops profiles of usage of network resources and then compares current usage patterns with the historical profile to determine possible

security violations. The goal of our proposed research is to extend our network intrusion-detection concept from the LAN environment to arbitrarily wider areas with the network topology being arbitrary as well.

The generalized distributed environment is heterogeneous, i.e., the network nodes can be hosts or servers from different vendors, or some of them could be LAN managers, like our NSM, as well. The proposed architecture for this distributed intrusion-detection system consists of three major components. First, there is a host manager in each host that is required to be monitored by the system. This manager is a collection of processes running in background in the host. Second, each LAN that is monitored has a LAN manager, which operates just like a host manager except that it analyzes LAN traffic. Finally, there is a central manager which is placed at a single secure location. The central manager receives reports from various host and LAN managers, and by processing and correlating these reports, it is expected to detect intrusions.

2.2. Design Goals

DIDS is a proposed distributed intrusion detection system that is intended to enhance the effectiveness and efficiency of the SSO in monitoring a network of computers. DIDS is designed to discover attacks on, and misuse of, individual hosts as well as the network which connects them. It looks for attempts to subvert or avoid authentication as well as other methods of gaining unauthorized access to, or privileges on, the monitored computers. It also attempts to detect masqueraders and insiders committing a variety of offenses, e.g., espionage, or data alteration. The system is based on both known and hypothesized abuses. It is designed to operate in near real time, providing for both general surveillance and focused investigation. The analysis portion of the system utilizes an inference network and incorporates learning algorithms so that it can deal with new forms of attack and abuse as they develop. The inference network is also supported by an explanation facility which lets the operator examine the system's chain of reasoning. DIDS incorporates, yet refines, various ideas from a number of its predecessors. In addition to refining these approaches, DIDS provides a new dimension to intrusion detection by facilitating the correlation and analysis of data from multiple sources. The target environment will consist of a single physical segment of a Local Area Network with approximately 10 hosts running at least 2 different C2-level[1] secure operating systems.

[1] The DOD defines Class (C2) as controlled access protection, i.e., finely grained discretionary access control that makes users individually accountable for their actions (see [1]).

2.3. Overview

In DIDS, the monitoring and analysis functions are distributed among several components. These components include a *central manager*, a collection of *host managers*, and at least one *LAN manager*. The host and LAN managers are primarily responsible for detecting single events and known attack signatures which have a high probability of being relevant to the security of a system; so they must constantly monitor their respective domains. The central manager is responsible for analyzing the events reported by the host and LAN managers, and therefore will have access to the distributed audit data gathered by the various managers. The central manager may then use these records in support of a directed investigation of a particular subject. The central manager communicates bidirectionally with the host and LAN managers to facilitate the transfer of data, and the processing of queries and responses. It is also able to correlate information obtained from the individual host and LAN managers. The central manager also supports the user interface.

The host manager is a process or collection of processes running in the background on each individual host, making the presence of the host manager essentially transparent to the users of the host. The LAN manager and the central manager are also processes running on single hosts on the network. Ideally, the central manager should have exclusive use of a host, since it is responsible for the high-level analysis of the data, and it will therefore require significant computing resources. Although the LAN manager, the host manager, and the central manager may physically reside on the same computer, they are logically independent (see Fig. 1).

The division of labor between the central and distributed components of DIDS is straight-forward. Correlation of the data from multiple sources must be done at the central manager. Platform specific transformations of information should clearly be done on the individually monitored hosts. However, there is a trade-off between doing some of the analysis on the distributed managers and doing it all on the central manager. If more work is done on the distributed managers, it will reduce the chance that the processing capability of the central manager will be a bottleneck, and it will also reduce the amount of data that must be sent across the network. On the other hand, increasing the amount of analysis that is done on the distributed managers will obviously hinder the performance of the individual hosts.

2.4. Host Manager

The design of the host manager is modular (see Fig. 2). The major components deal with capturing and translating audit data, simple analysis of the translated audit data, and communication with the other components of DIDS.

The host managers use the native audit data provided by the host's operating system, so that the operating system itself is not modified. The host manager captures each audit record from the operating system, which is then mapped into a common syntax called a *Host Audit Record* (HAR). This common syntax across operating systems provides the first level of abstraction from the native audit record format and improves portability and heterogeneity of the remaining portions of the host manager. Redundant records are also eliminated at this point. This provides a significant reduction in the number of records that the higher levels of the host manager needs to process (more than a 4:1 reduction has been observed in early tests).

The host manager incorporates three levels of analysis performed on the HARs. At the lowest level, the host manager scans each HAR for *notable events*. Notable events are transactions that may be of interest independent of their context (i.e., independent of previous HARs). Examples of notable events include any type of network activity, failed file accesses, accessing system files, and changing a file's access control. At the next higher level, the host manager looks for *sequences* of events which may be interesting. Known attack signatures or patterns of abuse are examples of sequences which would be of interest. We are developing a general purpose approximate pattern matcher which will be used for this purpose. Finally, the host manager looks for anomalous behavior by tracking user behavior patterns, such as number of programs executed, number of files accessed, etc. As noted above, there are performance tradeoffs to be considered when deciding how much analysis should be done at the host manager. When a host manager notes an interesting event or pattern, it alerts the central manager and forwards the relevant information. The lists of notable events, the templates for pattern matching, and the metrics for anomaly detection are maintained separately to allow for easy modification.

The audit data for each host is, in general, stored locally. Since our target platform is a C2 or higher rated system the audit data is already being kept as part of the C2 specification [1] so this imposes no additional burden on the host. Moreover, unlike some systems, we do not use network bandwidth in transmitting all the audit data to the central analyzer. In addition to storing the native audit data, the host keeps a short history of the translated records produced by the host manager. This history simplifies the process of providing information to the central manager. The number and duration of these records depends on both system resources and security needs.

Although the audit records provide a significant amount of information, there are still a few metrics that are, at present, best gotten by querying the host operating system directly (e.g., Unix's *ps*, Sun's *rusers*, etc.). To facilitate these queries, the host manager provides an *agent* through which the central manager can ask for information about the status of the host. The host manager also handles central manager queries of the stored HARs.

2.5. LAN Manager

The DIDS LAN manager is built on the same foundation as UC Davis' Network Security Monitor [6]. Since there is no native LAN audit trail, the LAN manager is responsible for building its own. The LAN manager sees every packet on its segment of the LAN. From these packets, the LAN manager is able to construct higher level objects such as connections (logical circuits), and service requests. In particular, it audits host-host connections, services used, and volume of traffic. Like the host based manager, the LAN manager uses several levels of analysis to catch the most significant events, for example, sudden changes in network load, the use of security-related services, and network activities such as *rlogin*. As with the host manager, the LAN manager retains the audit data for analysis by the central manager. It also uses and maintains profiles of network behavior, which are updated periodically. Like the host manager, the LAN manager provides an agent for communications with the central manager. In addition to handling queries of the audit data from the central manager, this agent gives the central manager access to a number of network management tools, which are analogous to the host operating system services provided by the host manager.

2.6. Central Manager

The central manager consists of four major components: an interprocess *communications agent*, an *expert system*, a set of *tools*, and a *user interface*.

2.6.1. Communications Agent

The *communications agent* is responsible for the coordination of information and instructions between the central manager and each of the host and the LAN managers. In particular, the communications agent interprets requests from the expert system and relays them to the agents of the host and the LAN managers;

it also buffers the corresponding replies and converts them into the form that is expected by the expert system. The communications agent also processes the SSO's requests for the various managers, tools, and operating systems. For example, the SSO, through the user interface, might request a listing of all current logins on a particular host. That request would pass through the communications agent, which would form an appropriate call to the target OS. When a response is received from the host manager, the communications agent would then return that response to the user interface. Network protocols are provided so that the details of the communications between the various components of DIDS are invisible to the individual components (hosts and LANs), as well as to the SSO.

2.6.2. Expert System

The *expert system* is responsible for evaluating and reporting the security state of the monitored network. It receives the reports from the host and the LAN managers through the communications agent, and, based on these reports, it makes inferences about the security of users, hosts, processes, etc. The expert system reports its conclusions through the user interface. The expert system consists of three components: the *evidence*, which are facts about the world; the *inference rules*, which represent the knowledge which the expert uses in reasoning about the evidence; and the *inference mechanism*, which controls the process of reasoning.

The evidence takes the form of audit records residing on the individual hosts and on the LAN manager. In addition, the expert system has access to databases of (relatively) static information regarding the hosts, LANs, and users. This information could include such things as: user SMITH is on vacation, or host EREBUS is out of service. In its local fact base, the expert system has access to its own intermediate hypotheses as well as the notable events which have been reported by the LAN and the host managers.

Except for the information in the static databases, all of the facts are affected by the passage of time. The individual audit files are periodically purged of the oldest information. The intermediate hypotheses and notable events in the local fact base are also aged. Thus, the expert system recognizes the importance of temporal proximity without having to specify sequences of events, (see [9]).

The inference rules provide the ability to recognize a security incident. Each rule takes the general form of a conclusion, which is logically dependent on one or more antecedents. The individual rules are then linked together to form an inference network. That is, the antecedents of a rule are, in general, the conclusions of other rules. The rule base has a hierarchical structure based on a model which describes a security incident in terms of levels of abstraction from the evidence. Reasoning progresses up the hierarchy of abstractions using rules whose conclusions are at a higher level than their antecedents. The current version of the model has seven layers of abstraction. The model provides a framework for developing the rules themselves, as well as for providing the foundation for our argument that the rule base is comprehensive.

The principle issues related to the inferencing mechanism are handling uncertainty and training. There are few situations in which one can be completely sure whether a user's behavior represents an abuse. Therefore, a mechanism which attempts to evaluate security incidents must be capable of reasoning with uncertainty. To this end, rules have a *Rule Value* (RV) associated with them. This is similar to the certainty factor of EMYCIN [5]. The RV represents the degree to which the truth of the antecedents of a rule guarantees the truth of the conclusion. The RV is expressed as number between 0 (no faith in rule) and 100 (absolute faith in rule).

For each *intermediate hypothesis* (i.e., conclusion of some rule) the system maintains a *Hypothesis Value* (HV). The HV is expressed as a value between 0 (no support for the hypothesis) and 100 (the hypothesis is absolutely true). HVs automatically decrease over time as a perceived threat diminishes.

A simple algorithm is used for combining HVs and RVs. In applying rules whose antecedents are uncertain, the HV of the antecedent is multiplied by the RV of the rule to get an effective HV. The effective HV is then applied to the conclusion of the rule. In the case of rules with disjunctive antecedents, the maximum of the applicable HVs is used; in cases of conjunctive antecedents, the minimum of the applicable HVs is used; in the case in which multiple rules support a single hypothesis, the following formula is used:

$$HV_1 = HV_1 + \frac{(100 - HV_1) \times HV_2}{100}$$

where HV_1 is the HV of the hypothesis and HV_2 is the effective HV generated by the new application of a rule. This simple method avoids the special case calculations of MYCIN's certainty factors and the need for conditional probabilities as in the Bayesian approach [7].

While inductive learning has been successfully applied to the problem of recognizing anomalies [10], it does not apply to the problem of recognizing abuses. This is because there is no general criterion for recognizing an abuse as there is for recognizing an anomaly. On the other hand, it seems unlikely that one could specify a rule base using traditional "knowledge

engineering" methods. It is particularly unlikely that this approach could provide useful certainty values. As an alternative to either of these approaches, a method based on negative reinforcement is proposed. Essentially, the expert system will be taught to recognize what is *not* an abuse. The initial rule base is intentionally broad, with all of the rules initially given high values. The untrained system is thus overly sensitive and reports many security incidents which are false alarms. A human trainer is required to decide which of the reported incidents are valid. When the trainer ascertains that a report is invalid (i.e., a false alarm), he/she initiates a process which automatically reduces the RV of each rule that was used in inferring the security incident. This approach is fundamentally different from the inductive learning method used by other IDSs in that it makes no assumptions about the content of the training data. What is required is that the training proceed until the system makes acceptably-few mistakes. To avoid having the system become too desensitized, the RV of all active rules uniformly and gradually increases over time. This has the effect of keeping the system tuned.

This training method has the advantage that a general system can be created which will tailor itself to a particular installation or class of installations. Also, as a side effect of this training method, we get a reasonable approximation of anomaly detection. Since all of the rules are initialized with high RVs, and the RVs are only lowered when a rule is used in drawing a false inference, those rules whose antecedents are seldom seen will continue to have high RVs after the initial training.

2.6.3. Tools

We anticipate that a growing set of tools will be used in conjunction with the intrusion detection functions of DIDS. Among these will include incident handling tools. Depending on the level of expertise of the SSO, these tools could include online explicit instructions, e.g., "who to call" lists, automatic response procedures, etc. In addition to the incident handling tools, the user interface can provide access to standard network administration tools; in particular, we are evaluating the SNMP tools currently available.

2.6.4. User Interface

The central manager's user interface serves both the intrusion detection and the incident management components of DIDS. It is designed to take advantage of the graphic capabilities of the target platform (a Sun SPARCstation 1). The user interface is hierarchical in its presentation of information, allowing the SSO to display as much, or as little, of the information available from DIDS as he/she wants. In its most compact form, the interface is reduced to a state meter displaying a "measure" of the security state of the network as evaluated by DIDS. Other options include a control panel for access to system management tools, displays of summary or detailed monitoring of the net, of hosts or of users, and a query capability for the distributed audit data. The user interface is designed to be extensible, so that new functionality can be added without undue effort.

3. Conclusion

This paper presented an architecture for a Distributed Intrusion Detection System (DIDS). The target environment for DIDS is a heterogeneous network of computers that may consist of different hosts, servers, etc. The DIDS architecture consists of a collection of host managers, each of which monitors a single host computer; one or more LAN managers, each of which monitors its corresponding LAN traffic, including that flowing between a subset of the hosts; and a central manager, which receives reports from the various managers, correlates the data, and detects intrusions.

We believe that DIDS will be able to detect the same kind of single host intrusions that are flagged by other intrusion detection systems, such as IDES [2], Wisdom & Sense [10], and MIDAS [8]. DIDS should also be able to (1) detect attacks on the network itself, (2) detect attacks involving multiple hosts, (3) track tagged objects, including users and sensitive files, as they move around the network, (4) detect, via erroneous or misleading reports, situations where a host might be taken over by an attacker, and (5) monitor the activity of any networked system that doesn't have a host monitor, yet generates LAN activity, such as a PC.

References

1. *Department of Defense Trusted Computer System Evaluation Criteria*, Dept. of Defense, National Computer Security Center, DOD 5200.28-STD, Dec. 1985.
2. T.F. Lunt, et al., "IDES: A Progress Report," *Proc. Sixth Annual Computer Security Applications Conference*, Tucson, AZ, Dec. 1990.
3. T.F. Lunt, et al., "A Real-Time Intrusion-Detection Expert System (IDES)," Interim Progress Report, Project 6784, SRI International, May 1990.
4. D.E. Denning, "An Intrusion-Detection Model," *IEEE Trans. on Software Engeneering*, vol. SE-13, no. 2, pp. 222-232, Feb. 1987.

5. R. Forsyth, ed., *Expert Systems, Principles and Case Studies*, Chapman & Hall, London, 1989.

6. L.T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor," *Proc. 1990 Symposium on Research in Security and Privacy*, pp. 296-304, May 1990.

7. K.C. Ng and B. Abramson, "Uncertainty Management in Expert Systems," *IEEE Expert*, Apr. 1990.

8. M.M. Sebring, E. Shellhouse, M.E. Hanna, and R.A. Whitehurst, "Expert Systems in Intrusion Detection: A Case Study," *Proc. 11th National Computer Security Conference*, pp. 74-81, Oct. 1988.

9. H.S. Teng, K. Chen, and S.C-Y Lu, *Adaptive Real-time Anomaly Detection Using Inductively Generated Sequential Patterns.*

10. H.S. Vaccaro and G.E. Liepins, "Detection of Anomalous Computer Session Activity," *Proc. Symposium on Research in Security and Privacy*, pp. 280-289, Oakland, CA, May 1989.

11. J.R. Winkler and W.J. Page, "Intrusion and Anomaly Detection in Trusted Systems," *Proc. 5th Annual IEEE Computer Security Applications Conference*, pp. 39-45, Dec. 1989.

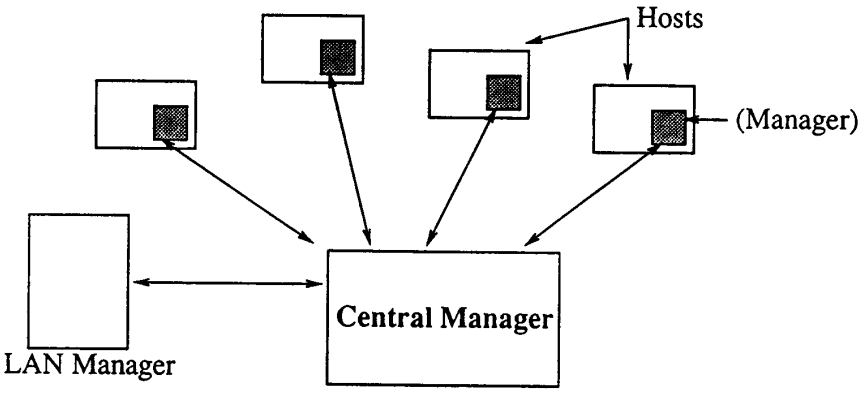

Fig. 1. Architecture of the Distributed Intrusion Detection System

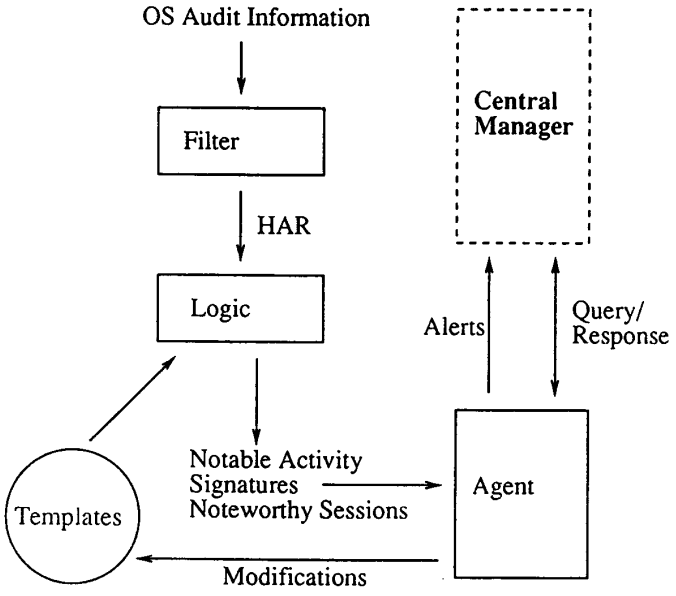

Fig. 2. Host Manager Architecture