DESIGN AND IMPLEMENTATION OF A POLICY-BASED INTRUSION DETECTION SYSTEM – GENERIC INTRUSION DETECTION MODEL FOR A DISTRIBUTED NETWORK

By

AKHIL NARAYAN KARKERA

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2002

ACKNOWLEDGMENTS

I would like to thank Dr. Richard Newman for his guidance and the opportunity to work on this project. His brilliant analysis in our discussions on the topic has helped shape this thesis.

I would also like to thank Dr. Randy Chow, Dr. Douglas Dankel and Dr. Jonathan Liu for agreeing to serve on my committee and for their valuable suggestions.

I express my gratitude to Murugabhupathy Chengalvarayan and Matt Wilson for working with me on this project and their contributions. I also thank Amit Gandre for introducing me to the project.

Last but not least, I would like to thank my parents, brother and my friends Naren, Nikhil, Seema, Nandhini and Kaumudi for their constant support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

DESIGN AND IMPLEMENTATION OF A POLICY-BASED INTRUSION DETECTION SYSTEM – GENERIC INTRUSION DETECTION MODEL FOR A DISTRIBUTED NETWORK

By

Akhil Narayan Karkera

December 2002

Chair: Dr. Richard E. Newman
Major Department: Computer and Information Science and Engineering

Computer networks and computer systems on these networks are increasingly being subject to attacks either to gain access to systems or simply to deny service to legitimate users of those systems. Intrusion detection systems (IDS) to prevent or mitigate these attacks have been in development for the past twenty years now. Earlier systems were targeted towards specific types of attacks and this led to the development of network-based and host-based IDS systems. The past decade has seen these systems being integrated into one, but under single monolithic analysis components. Such systems do not scale well to a distributed setting. Furthermore the single central analysis component presents a single point of failure for the system.

The focus of this thesis is the design and implementation of a generic intrusion detection model (GIDEM) for a distributed network. This thesis extends and remodels the original GIDEM architecture to suit the needs of a large network of potentially thousands of hosts. Techniques to manage effectively the large number of hosts and ensure system survival are presented. The extension of the infrastructure to support knowledge exchange among different hosts on a network is also briefly mentioned.

CHAPTER 1
INTRODUCTION

1.1 Problem Definition

Computer networks have brought the world together, by bridging the information gap among people. Network technology has undergone a revolution with better and faster ways of sending information between computers. Unfortunately security systems and policies to govern these networks have not progressed as rapidly. With the birth of the Internet in 1969, computers and computer networks that were isolated from each other were suddenly interconnected. The Internet has grown at an explosive rate with innovations in communication and information technologies. While the Internet has made it easier to reach and communicate with people all over the globe, it has also made it easier to attack computer systems connected to it. With e-commerce and online banking being more and more employed these days, security for systems offering these services has become necessary. Furthermore, the requirement for ease of communication has forced companies and government organizations to open their computer networks to the Internet. To add to this, more and more operating system flaws, and network protocol features that can be exploited, are being discovered with each passing day.

The three most important aspects of security are confidentiality, integrity, and availability [5]. Confidentiality implies that only authorized entities can access and modify information and resources, while unauthorized entities are denied access. Integrity means that the information being accessed is consistent, accurate and can be changed only through authorized actions. Availability for data and services implies that legitimate users should have fair and timely access, the services should be usable and the capacity provided should meet needs. Ensuring that these three aspects are provided for is the primary challenge facing security professionals today.

1.2 Extended Generic Intrusion Detection Model

Intrusion detection started with network-based and host-based systems. Later there were attempts to bring these together into one system. In order to handle large networks, these systems were installed on different parts of the network. But most of these intrusion detection systems (IDSs) have central analysis components. This presents a single point of failure for the whole system. They do not scale well to a distributed network. In addition each intrusion detection system has its own way of reporting, which is usually not compatible with other systems.

The extended generic intrusion detection model (EGIDEM) attempts to get over these limitations by providing an infrastructure for handling intrusions on a distributed scale and for sharing information between different hosts, thus avoiding the need for central analysis. The focus throughout this thesis is on developing small and effective agents for intrusion detection and analysis. These agents can be incorporated on the fly into the system and are not critical for the system to function. Analysis is done on a distributed scale, by these agents and hence there is no single point of failure. The policy-based approach allows for most of the system intelligence to be present in the rule set, thus avoiding the need for complicated components.

1.3 Organization of the Thesis

Chapter 2 gives an introduction into Intrusion Detection Systems and discusses some of the work currently being done on these systems. Chapter 3 describes the architecture of our Intrusion Detection System and how it scales to a large distributed network. Chapter 4 briefly describes the sharing of information between hosts on the network, which is one of the primary applications of the architecture. Chapter 5 has the conclusion and comments on future work.

CHAPTER 2
INTRUSION DETECTION SYSTEMS

2.1 Introduction

An intrusion as defined in Balasubramaniyan et al. [4] is any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource. Intrusion detection is the technique of determining that an attempt has been made at compromising the resource, or worse the resource has been compromised. One point that needs to be made clear is that, intrusion detection systems (IDSs) do not detect intrusions; they detect evidence or manifestations of intrusions, either while the intrusion is in progress or after an intrusion has occurred.

2.2 Types of Intruders

Intruders are basically of two types: external and internal. External intruders do not have any authorized access to the system they attack. Internal intruders on the other hand have some access rights, but they seek to gain additional capability [22]. A surprisingly large number of security breaches are due to internal intruders. Their motivation to hack is either because they can hack, to cause destruction if they are disgruntled employees, or to profit from the company.

2.3 Approaches to Intrusion Detection

There have been two approaches to detecting intrusions: anomaly detection and misuse detection.

2.3.1 Anomaly Detection

Anomaly detection defines and characterizes correct static form and/or acceptable dynamic behavior of the system, and then detects wrongful changes or wrongful behavior [13]. The existing behavior of the system is defined as the normal, and then deviants from this behavior can be detected. Detecting deviants in the case of files is pretty straightforward, as the file contents are precisely known. However there is a fuzzy line between normal and anomalous when it comes to detecting system behavior.

Anomaly detection can be divided into static and normal anomaly detection. Static anomaly detection monitors part of the system that should remain constant. Static anomaly detectors can test system code and system data for variations. Tripwire is an example of a static anomaly detector.

Dynamic anomaly detectors define system behavior. This behavior is usually defined as a sequence of distinct events. Statistical distributions with means and standard deviations help to decide what is normal and what is anomalous. Typically these detectors start with a base profile of the system being monitored and then change the base profile over time.

2.3.2 Misuse Detection

Misuse detection on the other hand defines patterns for known ways to penetrate a system. It then tries to match system activity to these patterns and raises an error when a match is found. Patterns can be defined as strings or as a set of actions that are associated with a known attack.

The first generation of misuse detectors used rules to match patterns (MIDAS [21]). However as these rules grew in number, it became a problem to interpret and modify them. The second generation misuse detectors use model-based rules (IDES [8] and NIDES [1]) and state-transition representations (USTAT [12]). Under the model-based approach, each intrusion scenario is separately modeled so that the number of rules to be considered is less during modifications. The state-transition approach uses attribute-value pairs to characterize the system state. Actions that contribute to the intrusion scenario are defined as transitions between these states.

2.4 Types of Intrusion Detection Systems

There are two main types of IDSs: network intrusion detection and host-based intrusion detection.

2.4.1 Network Intrusion Detection

Network intrusion detection monitors the packets that are being transmitted through a network. They are also called packet-sniffers. They generally have a signature database against which they compare network packets. These systems have been incapable of operating in switched environments, encrypted networks and high-speed networks. Cisco's latest line of switches tries to get around these problems, by incorporating network intrusion detection directly into the switch.

2.4.2 Host-based Intrusion Detection

Host-based intrusion detection systems monitor activity on a host. They are best suited for internal threats because of their ability to monitor and react to specific user actions and file accesses on the host. They offer audit policy management centralization, supply forensics, statistical analysis, and evidentiary support.

2.4.3 Hybrid Intrusion Detection

Hybrid intrusion detection systems manage both network-based and host-based systems. They are kind of a central intrusion detection system and add a logical layer to NID and HID.

2.4.4 Network-Node Intrusion Detection

These systems are an improvement over the network-based intrusion detection systems. They monitor network packets that reach a particular host. They are installed on host-based intrusion systems as an add-on component usually. The disadvantage of this approach is that the whole network is not monitored.

2.5 Issues with Intrusion Detection System

There is a considerable overhead involved in collecting information. If every system activity is monitored it would require huge amounts of storage for that information and slower analysis by the IDS. The amount of system activity an IDS collects is thus a tradeoff between efficiency of storage and analysis, and effectiveness of the IDS in detecting attacks. Collecting information is expensive, and collecting the right information is important.

As far as detection techniques go, anomaly detection can detect previously unknown attacks. This is however very difficult in highly dynamic environments, where determining the baseline of system activity is difficult. By decreasing the alert threshold, the IDS will report more false positives. By increasing the alert threshold, several actual attacks may not be caught by the sensors. Misuse detection on the other hand throws very few false positives. But at the same time, it can detect very few attacks – only those for which the signature patterns are known beforehand.

Performance is a big problem with Intrusion Detection Systems. There is a considerable overhead incurred if Intrusion Detection is employed. Software Intrusion Detection Systems cannot keep up with the pace of development of today’s networks. Their inability to keep up causes many packets to be dropped.

2.6 Other Intrusion Detection Systems

In this section, we take a look at some other IDSs that have been developed in the recent past and on the technologies being used in current IDSs.

2.6.1 AAFID (Autonomous Agents for Intrusion Detection)

The AAFID [4] system introduced the concept of using autonomous agents to handle intrusion detection. The tries to address some of the issues faced with monolithic analysis units in earlier IDSs with this approach. The AAFID system consists of three components: agents, transceivers, and monitors. Agents examine certain aspects of the host. They report all abnormal behavior to corresponding transceivers. Agents do not directly generate an alarm. The transceiver or monitor will generate an error based on the information that the agent provides. Transceivers handle alarms on a host level, while monitors handle alarms on a network level.

The AAFID architecture is shown in Figure 2.1. Any agent can be used with the AAFID system, as long as it reports information in the format that the transceiver understands. The transceivers handle control and data processing for the host. Every host that is part of the AAFID system needs to have a transceiver on it. Monitors can do higher level correlation. They also handle control and data processing, but on a network scale. User Interfaces to the AAFID system are plugged into the monitors. The monitors have an API that lets a Graphical User Interface (GUI) access the AAFID system.

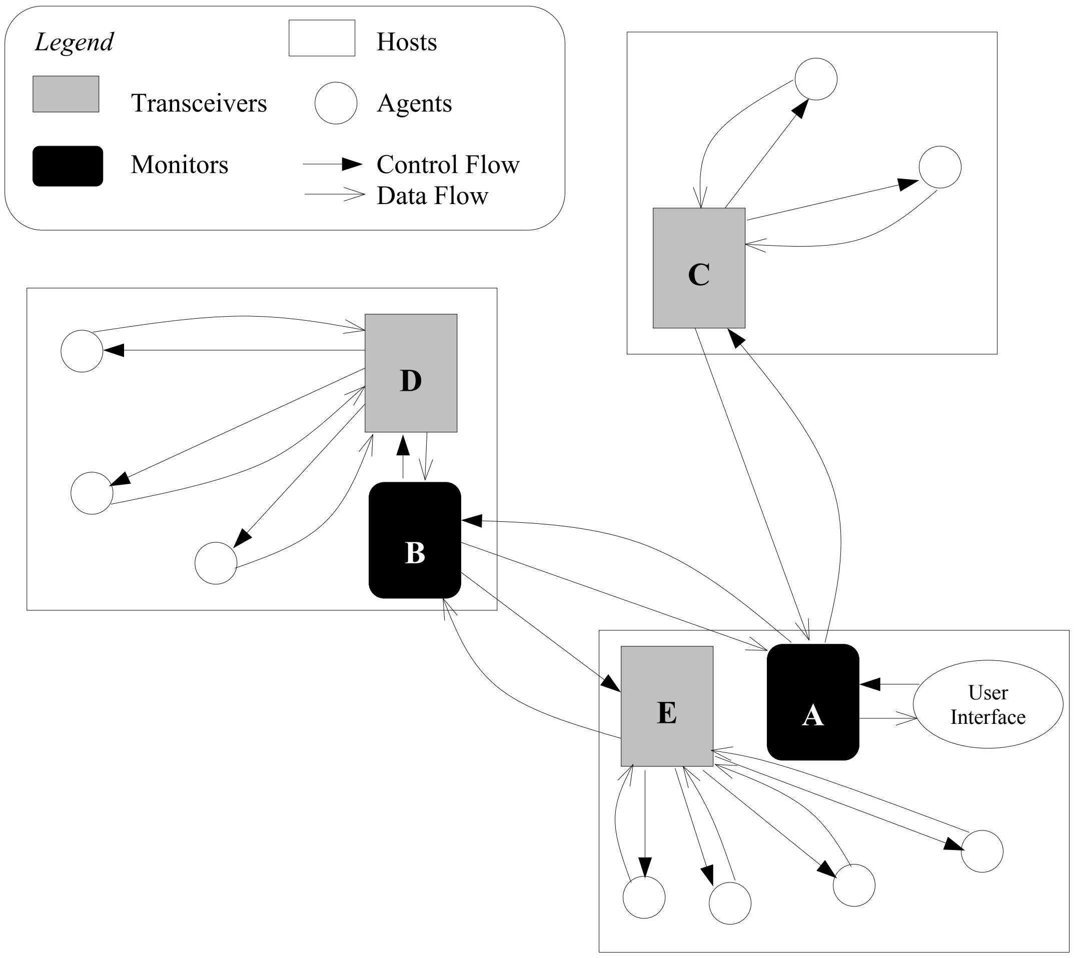

Figure 2.1 AAFID Architecture [4]

2.6.2 EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances)

The EMERALD [19] architecture uses a hierarchically layered approach to network surveillance. It is a distributed scalable tool suite that can handle large networks. The architecture is divided into three tiers: service analysis, domain-wide analysis, and enterprise-wide analysis. EMERALD has an application programmers' interface that allows for integration of third-party tools into the system. It also provides a framework for coordinating analysis from distributed monitors for global detection capabilities.

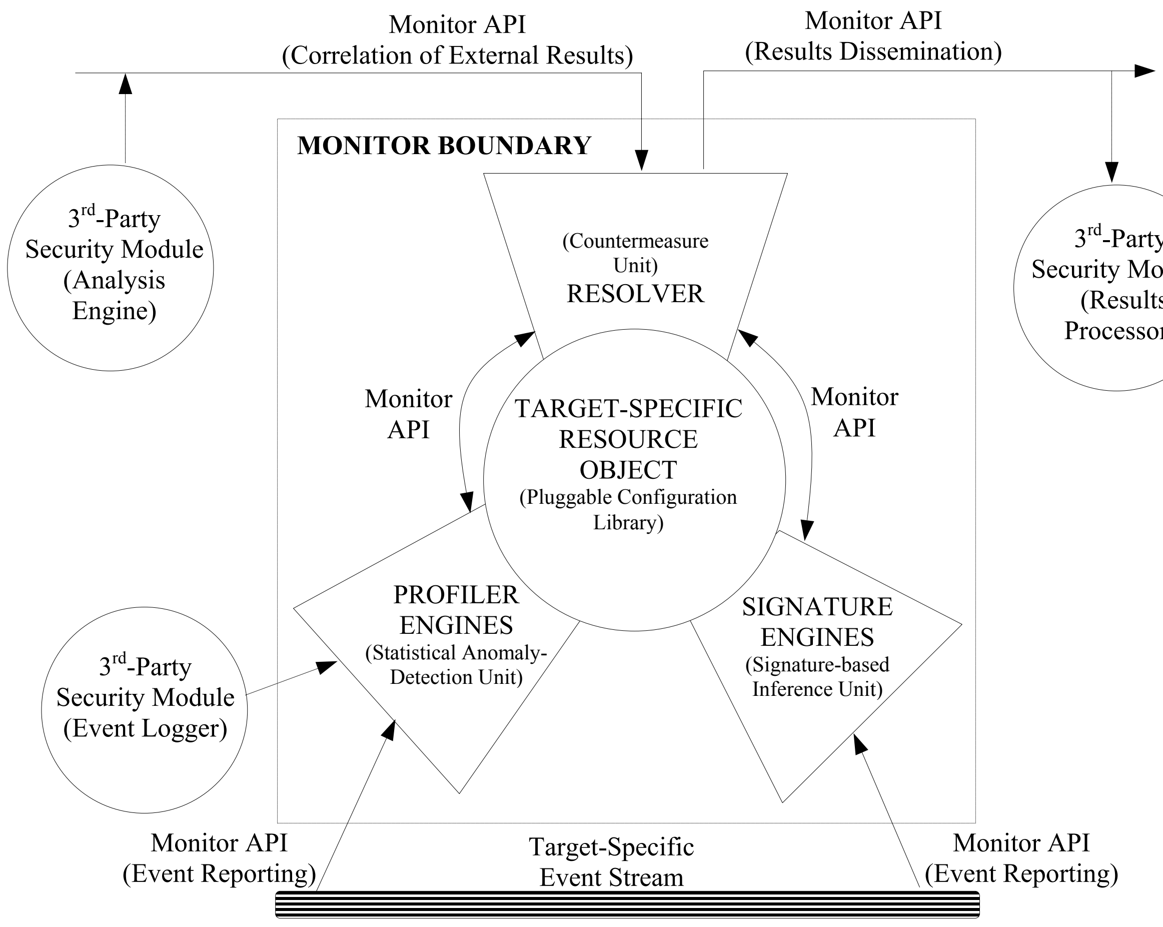
Monitor API
(Correlation of External Results)
Monitor API
(Results Dissemination)
MONITOR BOUNDARY
3rd-Party Security Module (Analysis Engine)
(Countermeasure Unit) RESOLVER
3rd-Party Security Module (Results Processor)
Monitor API
TARGET-SPECIFIC RESOURCE OBJECT (Pluggable Configuration Library)
Monitor API
PROFILER ENGINES (Statistical Anomaly-Detection Unit)
SIGNATURE ENGINES (Signature-based Inference Unit)
3rd-Party Security Module (Event Logger)
Monitor API (Event Reporting)
Target-Specific Event Stream
Monitor API (Event Reporting)

Figure 2.2 EMERALD Architecture [19]

Service analysis handles individual components and network services within a domain. Information that the service monitor gathers is forwarded to the other monitors using a subscription scheme. Client monitors subscribe to server monitors to receive information. This technique helps to keep the message passing overhead low.

Domain-wide analysis covers misuse over several service analysis components. It covers an entire domain. Domain monitors also handle system reconfiguration.

Enterprise wide analysis allows coordination of analyses of domain-wide analysis components. Thus information in one domain can be propagated to other monitors in different domains. This architecture does not require a central administration for the enterprise-wide monitors.

The EMERALD monitor architecture is shown in Figure 2.2. The architecture supports easy integration and deletion of analysis engines to the monitor boundary. The monitor performs both signature analysis and statistical profiling. The profiler engine performs statistical profile-based anomaly detection. The signature engine provides a focused and distributed signature analysis model. The normally central rule-base and inference engine are distributed and modularized into smaller, more focused signature engines. The resolver coordinates the analysis reports of the two engines and implements the response policy. The resource object is a pluggable library of target specific configuration data and methods. This lets the monitor code-base to remain independent of the target machine specifics. To integrate third-party modules into the system, EMERALD uses a standard interface specification for communication.

2.6.3 GrIDS (A Graph Based Intrusion Detection System for Large Networks)

GrIDS [23] is a graph based intrusion detection system. It collects data about activity on computers and network traffic between them. It then creates tree- like activity graphs from this information, which reveal the structure of network activity. The hosts form the nodes of the graph, while the activities between them are represented as branches between the nodes. The GrIDS components then add to the graph when the activity propagates to other hosts. The GrIDS algorithm has defined thresholds for the size of the graph, and when these thresholds are exceeded, an alarm is raised. If a graph is not being modified, it expires after sometime and is deleted. The GrIDS system can also implement organization policies. The main goal of the GrIDS system is to detect widespread attacks and to do so in near real time.

2.7 Recent Advances in Intrusion Detection

AAFID and EMERALD were frameworks that allowed intrusion detection systems scale up to the domain and enterprise levels. There have been other significant contributions to the field of intrusion detection that apply to specific components of intrusion detection.

2.7.1 Mobile Agent Technology

In keeping with the move away from centralized systems, mobile agent technologies for intrusion detection attempt to address the issues of having a single point of failure, scalability, and re-configurability of the system.

SPARTA (Security Policy Adaptation Reinforced through Agents) [15] is one such system. Each host on the system has a local event generator, a storage component, and a mobile agent platform installed. This mobile agent platform is responsible for moving the state and the code of the agents between different hosts and for providing an execution environment for them. The system also provides protection against security risks involved when utilizing mobile code. The SPARTA detection algorithm uses an attack pattern language called EQL. Definition of attack patterns using EQL allows the reduction of data transferred, while still retaining enough information for analysis. Mobile agents locally select interesting information and only move parts of the data across the network. For agent authentication, SPARTA uses a public key infrastructure.

IDA [3] is another mobile agent framework. IDA employs sensors on every host to monitor system logs. When an intrusion is suspected, the manager devises a tracing route and deploys a tracing agent to the hosts on that part of the network. When the tracing agent arrives at a host, it activates an information-gathering agent. This information gathering agent extracts information pertaining to the intrusion from the sensors. It then reports back to the manager. The tracing agent meanwhile jumps from host to host trying to discover more about the intrusion independent of the manager. Message boards are used on each host so that the agents on that host can exchange information between each other. Bulletin boards are placed on the manager machine and are used for integrating information reported by the mobile agents. The centralized manager in the IDA system is one of its weaknesses, making it a single point of failure.

Helmer et al. [11] proposes an intrusion detection system based on the concept of distributed knowledge networks and data warehousing techniques. Distributed knowledge networks use agents for information retrieval and extraction, data transformation, and knowledge discovery.

Data warehouse technologies are used for data and knowledge organization and assimilation from heterogeneous physically distributed data and knowledge sources. Figure 2.3 shows the architecture of the system. It consists of routers that read log files and parse them. This data is provided to data cleaners which process the data into homogeneous formats. Routers and the data cleaners have to be present at every host that is monitored by the IDS.

Mobile agents that reside above the data cleaning agents perform intrusion detection. These agents travel to each of their associated data cleaning agents and gather information, classify data, and determine if suspicious activity is taking place.

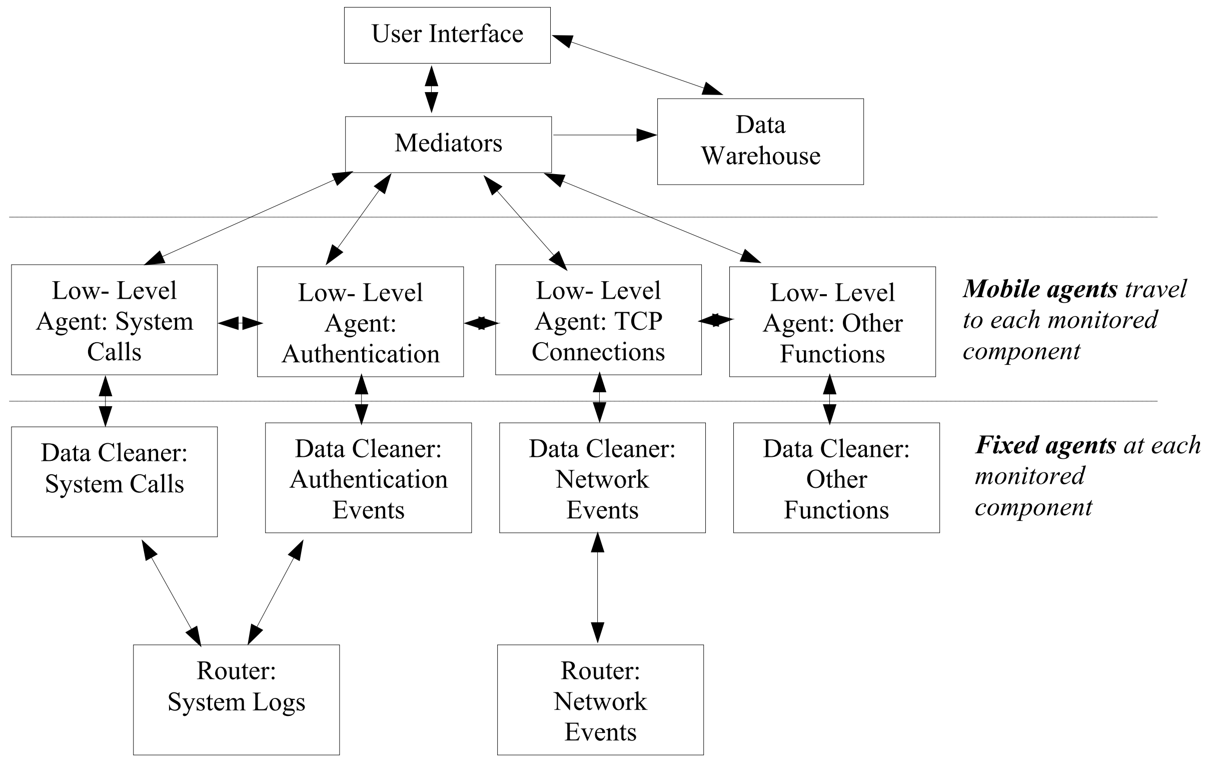
User Interface
Mediators
Data Warehouse
Low- Level Agent: System Calls
Low- Level Agent: Authentication
Low- Level Agent: TCP Connections
Low- Level Agent: Other Functions
Mobile agents *travel to each monitored component*
Data Cleaner: System Calls
Data Cleaner: Authentication Events
Data Cleaner: Network Events
Data Cleaner: Other Functions
Fixed agents *at each monitored component*
Router: System Logs
Router: Network Events

Figure 2.3 Architecture of the IDS [11]

Mediator agents at the top level, maintain the data warehouse by combining knowledge and data from the lower layer of agents. The mediator applies data mining algorithms to discover associations and patterns. The user interface helps to control the mediators, monitor the status of mobile agents, and access data warehouse features.

2.7.2 Other Technologies in Intrusion Detection

Data mining techniques can be used to discover new patterns and attack signatures. Audit data can be correlated to form association rules using an association-rules algorithm described in Lee and Stolfo [16]. These rules can then be merged with the existing rule base. Lee also discusses a frequent episode algorithm that tries to find a set of events that occur frequently within a sliding time window. This process of computation of new rules should be an integral part of the intrusion detection process.

The challenge facing these data mining approaches is the huge amount of audit data that is required to compute the profile rule sets. Lee proposes that the architecture should have two separate agents. One is the learning agent, and the other is the detection agent. The task of the learning agent is to compute very large amounts of data and generate rule sets. The detection agent is a generic and extensible entity that can work with a given rule syntax.

Kim and Bently [14] present a good analogy between the human immune system and a network IDS. There are several key features of the human immune system highlighted in this paper, that are applicable to a network IDS. It mentions the absence of a central coordinator for sustaining the appropriate level of immune responses. Each antibody set is unique and independent of the others. There is a gene library evolution that retains only the best of the antibodies and eliminates inappropriate ones. The immune system is also very lightweight. A vast number of antigens can be detected with a smaller number of antibodies, information about known antigens is reused efficiently, and numerous antibodies can be generated with a limited number of genes. Several similarities can be drawn between this view of the human immune system and the architecture proposed in this thesis.

With the demand for higher network performance, traditional shared networks are being migrated to switched networks. The problem with switched networks is that the IDS must be placed before the switch or must correlate information from separate IDS sensors on the different output lines of the switch. The IDSs are also unable to keep up with the pace of the switch when deployed in high performance networks. To address this issue, Cisco Systems the worldwide leader in networking for the internet has come up with the Catalyst 6000 [7] switch series. These switches integrate the IDS functionality directly into the switch itself. This provides real time intrusion detection and does not affect the switch performance, as the IDS only looks at copies of the packets being transferred. Furthermore the IDS module can be easily updated with the latest signatures. Cisco also has the IDS 4210/4235/4250 line of network sensors that monitor traffic from 45Mbps to gigabit environments. They also have solutions for host sensors, router sensors, and firewall sensors.

Intrusion detection systems and sensors have traditionally been developed mainly for UNIX systems. These days, due to the popularity of Windows, solutions are being developed for these systems too. Apap et al. [2] propose one such IDS that protects a Windows host. The algorithm proposed in this paper detects attacks on the host machine by looking for anomalous accesses to the Windows registry. The algorithm first defines a model of normal registry behavior on a Windows host, and then uses this model to detect abnormal registry access.

There have been some standardization efforts for intrusion detection event reporting and rules. Eckmann [9] describes an implementation of a translator that converts Snort rules to STATL scenarios. STATL is employed by NetSTAT [24] a popular network based IDS.

The limitation of an IDS is not in its ability to accurately detect misuse, but in its ability to suppress false alarms. Patton et al. [18] describe an alarming vulnerability which he calls “squealing.” Squealing is a technique of false IDS excitation that will render the IDS deaf to actual attacks. The hacker first attacks an IDS with a large number of packets having well-known attack signatures. This will cause the IDS to report many false positives. The actual attack can then progress concealed within these false positives. It can also render the system administrator to turn off the IDS components that produce the false positives, thus letting the attack progress undetected once the sensor is turned off. Squealing can also be used to alter statistical benchmarks used by anomaly detection sensors especially during their training periods. Once the benchmarks are updated, the actual attack can proceed without being detected as the excitation will be well below the new thresholds.

2.8 Trends in Intrusion Detection

The trend in intrusion detection is to distribute and use more specialized sensors. The result of this effort is hundreds of system components. All of these components should have an infrastructure to support them. A distributed Intrusion Detection System to support this many components is highly desirable.

There is a move to incorporate intrusion detection sensors into everyday computing environments. Intrusion detection should no longer be an optional component; it should be part of the operating system design. This will make tampering with the Intrusion detection system itself more difficult and make the process more efficient.

Data mining techniques are now being employed into intrusion detection sensors to catch previously unknown intrusions and to dynamically update sensor rules. Data warehousing techniques are also being used to effectively manage the huge amount of data being generated by the IDS.

For high performance networks, there is an effort to move intrusion detection into the network hardware. Cisco’s Catalyst 6000 is an example of a switch with intrusion detection built into the switch itself.

Intrusion detection systems have evolved independently for far too long now. There is a conscious effort by the Internet Engineering Task Force (IETF) to bring different intrusion detection systems together by trying to standardize the message exchange format and to define a generic event flow model for Intrusion Detection Systems.

2.9 Summary

This chapter looked at several intrusion detection systems and also some of the latest technologies in intrusion detection. There is a move towards implementing IDS infrastructure to scale up to meet the needs of large networks. Some of the works involve using mobile agents for intrusion detection. A lot of work is also being done to standardize of IDS event/signature formats and to improve IDS sensor capabilities.

CHAPTER 3
ARCHITECTURE

This chapter discusses the architecture of the Distributed Intrusion Detection System. This architecture is an extension of the Generic Intrusion Detection Model (GIDEM) discussed in Gandre [10]. The chapter starts with an overview of the architecture and then explains the individual components, the rule base, and the distributed setup.

3.1 Base Architecture

The base architecture has the following basic components/modules:

- A set of detectors
- A set of interfaces for the detectors
- Coordinator
- Rule Base
- Rule Base Parser
- Response Server
- A set of response agents

The modified GIDEM architecture for a single host in the distributed system is illustrated below. The basic event flow mechanism is the same as was proposed in Gandre [10]. However the interfaces between key components are much more simplified in this thesis. These simplifications help to avoid distinguishing between the network and host monitoring.

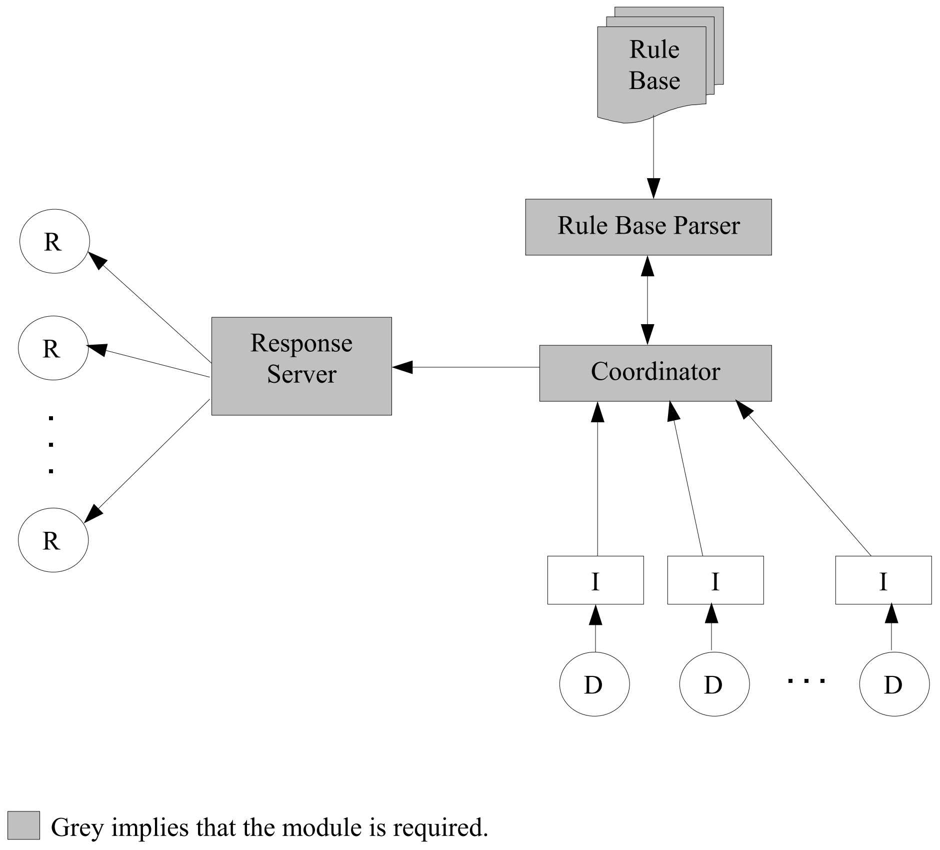

Figure 3.1 Architecture of a single host IDS

This basic architecture is duplicated on every host that needs to be monitored in the distributed system. For the single host IDS to scale up to a distributed system the basic architecture is augmented with two more modules: the heartbeat and the communicator. These modules allow the IDS on a host to communicate with neighboring hosts and to monitor their neighboring IDSs.

3.1.1 Detectors

The detectors (D in Figure 3.1) are the sensors used by the IDS to detect and monitor suspicious activity. They are implemented as autonomous software agents. An autonomous agent as defined by Balasubramaniyan et al. [4] with respect to their AAFID system is, a software entity that performs a certain security monitoring function at a host, runs continuously and autonomously in a particular environment, and is able to carry out activities in a flexible and intelligent manner. We have drawn from this idea of implementing sensors as autonomous agents and thus detectors in our system are also used in such a manner. They are not required for the rest of the system to function. Hence killing a detector does not hamper the IDS in any way other than rendering the IDS incapable of monitoring the particular security function that the detector was observing.

Detectors report suspicious activity in the form of events to the coordinator. It is not necessary for the detectors to report every event to the coordinator. The detector can perform analysis and then report only select events to the coordinator. Reporting every event helps in keeping the rest of the system knowledgeable, but at the same time, increases message traffic and protocol load. The architecture gives us the ability to tune or configure the interface of the detectors and thus decide what kinds of events are reported by the detectors. This can also be controlled by specific response agents that may react to excessive messages from a detector and consequently rewrite the detector/interface configuration files, to make them stop sending those particular messages.

3.1.2 Interfaces

The interfaces (I in Figure 3.1) are small software wrappers that help the IDS understand the detectors. They were introduced in AAFID [4], as a means of integrating detectors into the AAFID system. Interfaces translate the events reported by the detector into the common Event Reporting Language that is used by the rest of the IDS. An interface is necessary only for third party Detectors. They can be directly programmed into the ones that are developed in-house. The use of interfaces helps to integrate any detector into the IDS. It can also be used to integrate entire third-party IDSs into the system. This has been done for Snort [20], which is a freely available Intrusion Detection System.

3.1.3 Coordinator

The coordinator is the controller of the IDS. All events that are reported by the detectors go through the coordinator. Responses that are sent by the rule base are also routed through the coordinator. There are several advantages of having all traffic go through the coordinator. Logging of messages can be done at one point in the system. An extension of logging is maintaining the state of the system. This also can be done at the coordinator, with backups of the log files being taken at regular intervals. Another advantage is that event priorities can be assigned in the case that the IDS is swamped with events. The coordinator can then decide on an order of events that are sent to the rule base parser. The same advantages apply to the responses being sent by the rule base parser to the response server via the coordinator.

3.1.4 Rule Base

The rule base is a database that has policy decisions for events reported by the detectors. The decisions are responses that are enacted using response agents. The rule base is where the entire intelligence of the system is present. The remaining system modules can be lightweight. These modules can then be effectively grouped and triggered by the rule definitions. With the onus of intelligence on the rule base, the system can evolve over time, by slowly developing a very rich rule set. The rules can also be used to implement security and system behavior policies very effectively. The rule syntax is defined in Section 3.4.

3.1.5 Rule Base Parser

The rule base parser does the job of matching the events sent by the coordinator to the rules in the rule base. By having the coordinator handle event priorities, the job of the rule base parser is simplified. After matching the rules to the events, it reconstructs the action mentioned in the rule by substituting parameters into the action wherever necessary. The rule base parser then sends the events back to the coordinator, so that they can be sent in turn to the response server.

3.1.6 Response Server

The response server handles the triggering of actions that are sent by the coordinator. Some of the actions involve, sending parameters to already running response agents. In other cases, the response agents may not be running and have to be initialized by the response server. For simple tasks such as operating system shell commands, the response server may itself execute the actions without calls to particular response agents. The response server provides a good way of coordinating and controlling the response agents on the system. Ensuring that the response agents are authentic, and modification of their configuration files can all be done through the response server.

3.1.7 Response Agents

The response agents help the response server to execute a particular action. Each response agent handles only a specific action or set of actions. This helps to keep the response agents small and effective. Response agents can be made to email the administrator about an event, rewrite the configuration rules of the detectors, or start and stop detectors to prepare for a possible attack.

3.2 Extended Architecture

For the GIDEM architecture to scale up to a fully distributed model, two new components have been added:

- The Communicator
- The Heartbeat

These components are illustrated in the modified architecture diagram (Figure 3.2). The Communicator handles all inter-host communication, while the Heartbeat ensures that the IDS is alive on a particular host of the network. In a distributed system, the extended GIDEM architecture is replicated on most if not all hosts of the network. The components of the IDS that are not detectors, interfaces, or response agents are required by the IDS for basic functionality. These components are shown in grey. Each IDS will have its own detectors/response agents. These detectors/response agents need not be the same on every installation of the IDS.

As the core IDS modules are simple, they have a very low memory and processor footprint. Hence installing these on every host of the system will not affect system performance. Processing power is mainly used by the detectors and analysis components. These are not required on every host. For hosts that cannot afford running detectors, detectors residing on other hosts can do the monitoring for them. This flexibility lets the system be modeled differently, depending on the resources available. Thus the system can thus be configured according to needs, avoiding unnecessary installations. If a detector is performing remote monitoring however, it may not be very effective, as it does not have complete access to the system, and there could be substantial delay.

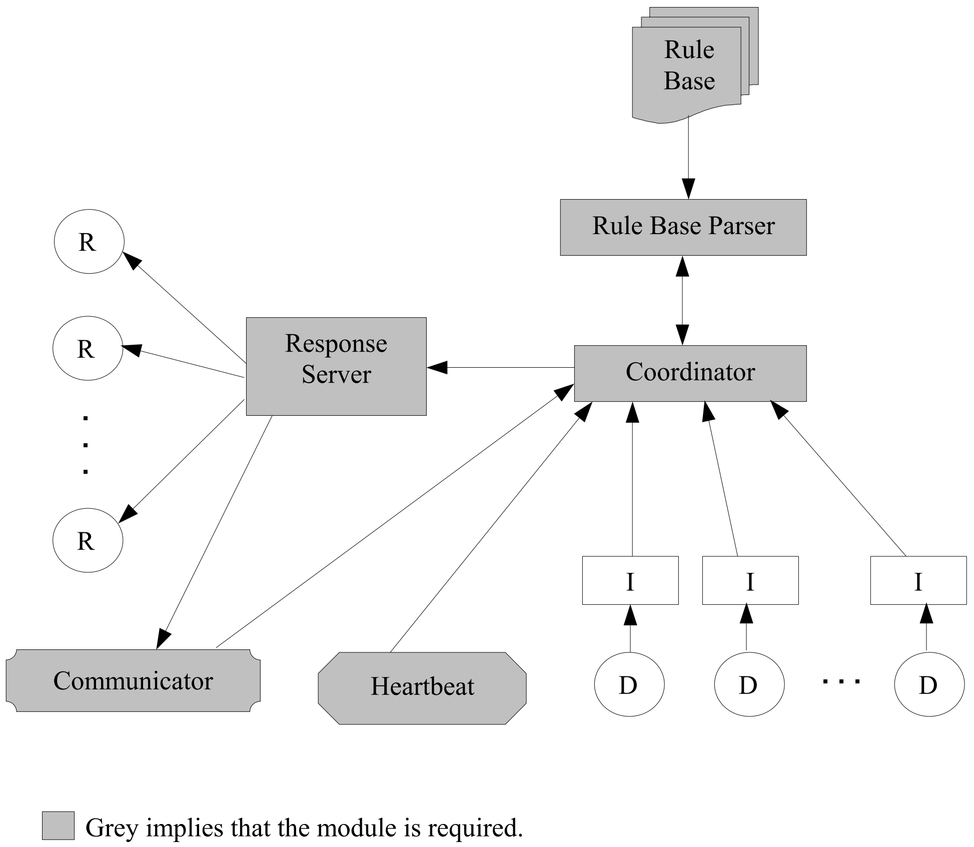

Figure 3.2 Extended Architecture

If a denial of service attack is being carried out against a host, then the host will not be able to respond to external detector probes. Similarly, this may hinder responses to attacks. It may be more advantageous in this case to simply run a copy of the IDS on that host and have the IDS download response agents/detectors to the host and run them on the fly.

3.2.1 Communicator

The communicator is a special module that handles all of the communication between the hosts of the network. The communicator acts both as a response agent and as a detector. If the rules in the rule base demand that a message be sent out to a remote host for further analysis, or if a remote detector/response agent needs to be activated, these are sent to the communicator first.

As part of the parameter list that the communicator receives, the remote IP address is also passed. The communicator then sends the message to the remote communicator on that IP address. It acts as a response agent in this case.

Messages received by the local communicator (from other remote IDSs) are reported as events to the local coordinator. The message in turn gets sent to the rule base parser and is processed as a normal event. Hence the communicator behaves as a detector in this case. This entire communication mechanism is shown in Figure 3.3. The sequence of messages is numbered in the figure.

By isolating inter-host communication to this one module, we can add onto the communication mechanism easily. Adding security for communications between hosts was made possible due to this. Another feature that can be added easily is concatenation different short messages into one message before they are sent out.

3.2.2 Heartbeat

The heartbeat is a special agent that is responsible for sending out periodic "alive" messages to the coordinator on the neighboring IDS. It also checks for the same messages being sent by the heartbeat agents of its neighbors. This lets the neighboring IDS know that another IDS is alive. If the message is not sent in a timely manner, the neighboring IDS generates an alert message. This alert message can be used to trigger an automatic restart of the local IDS or response agents that scan the local host for intrusions. The heartbeat implements the "Godfather-Child Policy" that is discussed in Section 3.3.

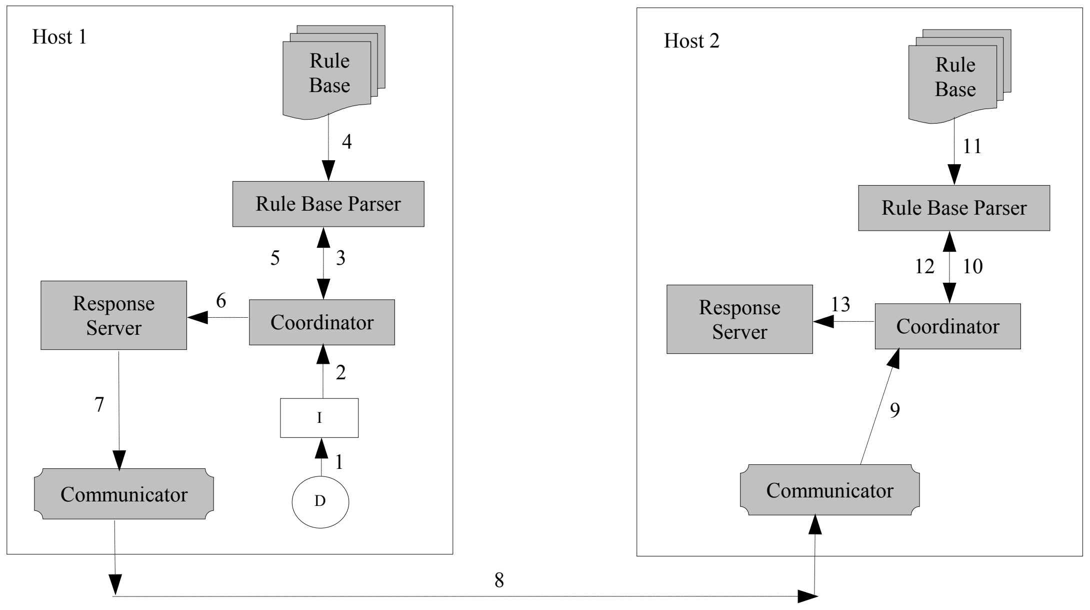

Figure 3.3 Inter-Host Communication

The heartbeat also acts as the timing agent of the system. It can be easily configured to send out periodic messages for events that are supposed to occur at certain intervals. One example is the request for knowledge (events occurring on other hosts). Certain special hosts on the network that are called knowledge stores use this message to trigger a knowledge request event that results in event digests being downloaded from neighboring knowledge stores. Another example would be a periodic self-check message. This can be used to fire of a response agent that performs integrity checks on the IDS itself. Periodic log file backups can also be triggered by the heartbeat. Hence, the heartbeat can be compared to the cron daemon on UNIX systems [17].

3.3 Division into Cells

The previous section described the extended GIDEM architecture and mentioned that it is advantageous to install the core modules on every host of the system. This section describes a way to arrange the distributed IDS system into cells, for improving

IDS cooperation and ensuring survivability. The cellular architecture arrangement is shown in Figure 3.4.

3.3.1 Determining Cell Members

There is no hard and fast rule that helps to determine cell members. There are however two goals that give us a direction on how to select the hosts that form a cell.

- It is necessary that if someone wants to bring the distributed IDS down then, the attack should not be allowed to proceed in any ordered manner. Any attack on the IDS should be felt on the whole system. If someone goes about killing the IDS on hosts, then the whole system should be able to “feel” it and react to it.

- The neighbors in a cell should be able to cooperate with each other for detecting attacks and taking appropriate response actions.

The key to achieving the first goal is to avoid isolation of hosts or cells in the system. This forces the attack to proceed on a distributed scale – killing the IDS instances almost simultaneously on all the hosts on the system (which is virtually impossible). Figure 3.5 shows an isolated cell.

The IDSs on this cell can be easily killed without the rest of the system knowing about the attack, as there is no interaction between it and the rest of the system. It is also important to select a good mixture of near and far hosts. This helps to take care of the situation wherein a distributed attack is carried out on a set of hosts that are physically close to each other.

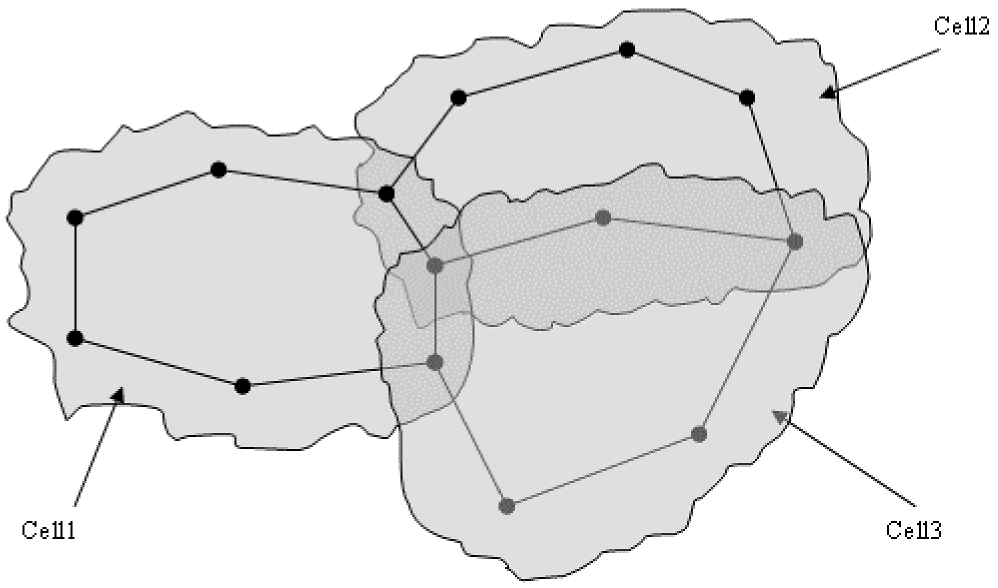

Figure 3.4 Cellular Architecture

Having a mixture of near and far hosts implies that the messages sent between cell members now go over several network segments before reaching their destination. This may not be desirable in case the IDSes are configured to report a large number of events to each other, and if the messages pass through routers with heavy traffic.

Achieving the second goal implies that a functional grouping must be performed. For example, a set of web servers can be grouped together. They can have specialized detectors and response agents running on them. It would make more sense for them to cooperate with each other, as they would experience similar kinds of attacks. Hence deciding which hosts form neighbors is a tradeoff among survivability, message traffic, and relevance of grouping.

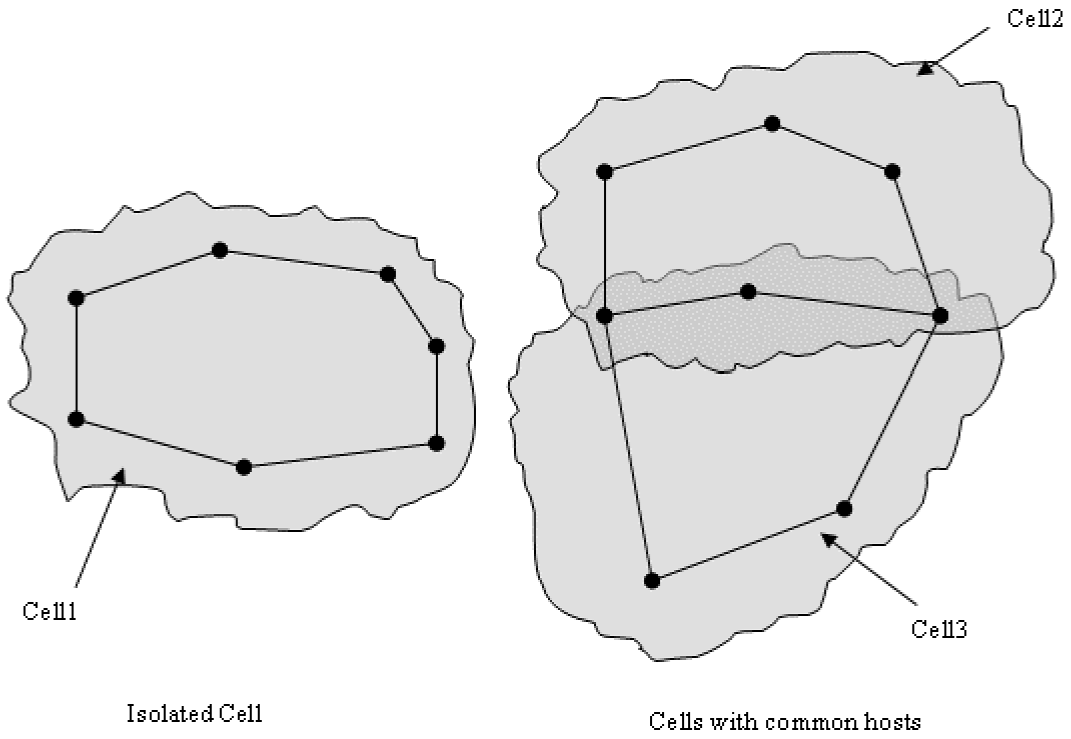

Figure 3.5 Isolated Cells

3.3.2 Godfather-Child Policy

Every host selects one of its neighbors as its godfather. Whenever the IDS on the host dies it is the godfather's responsibility to bring it back to life. To tell the godfather that it is awake, the host periodically sends out "alive" messages to the godfather. The godfather expects to receive these messages at regular intervals. In case it does not receive the "alive" message the godfather will timeout. After timing out, it will try to bring the host back to life, and raise an alert with its local coordinator. The godfather – child policy is implemented by the heartbeat module of the EGIDEM architecture.

There are a few rules for selecting the godfather for a host.

- If A is the godfather of B then B cannot be the godfather of A.
- Small cycles of godfathers should be avoided as far as possible. A cycle is desirable in case it spans the whole network. This measure helps to make sure that bringing down a select set of IDSs at a time does not proceed without an

alert being triggered.

- There can be only one godfather for a particular host. This avoids contention between godfathers in case a child is down. If more godfathers for a host, are required to improve security, then an appropriate protocol has to be implemented to handle contention.

- There can be multiple children for a godfather.

- A hierarchical arrangement of godfathers should be avoided. A hierarchical arrangement of godfathers would let an attack proceed in an ordered manner, by starting at the hierarchical top of the system and then moving down through the children.

The EGIDEM architecture consists of a cell configuration file. This file determines which hosts on the network are parts of the same cell. It also marks the host that is the godfather of the current host, and the host(s) that are the children of the current host.

The message format in the original GIDEM Architecture used the carat sign for parameter separation. The Extended Architecture message format uses spaces for the same.

3.4 Rule Syntax

A rule is a mapping between the events reported by intrusion detection sensors and actions that need to be taken when those events are reported. Actions can be as simple as mailing the system administrator, or may be more complicated, in which several agents are fired to further monitor the system, or launch a response. The Extended GIDEM architecture supports dynamic rule base modifications, in which the actions can specify updating the rule base with rules from remote hosts. The rule syntax for the Extended GIDEM Architecture is the same as the original rule syntax described in Gandre [10].

The rules follow the Event-Condition-Action model (ECA). The event is specified by a set of name-value pairs. These name-value pairs are reported by the sensors of the system. The condition checks these name-value pairs against predefined values in the rule, and if satisfied, it triggers the corresponding actions. All events can be successfully reduced to name-value pairs. Hence the ECA model provides the best way for reacting to sensor events.

The advantages of the ECA model as highlighted in Gandre [10] are that the representation is simple yet efficient, the rules are simple to create, and there is more control over the response due to the presence of conditions.

Agents in the IDS report events in the following format (Gandre [10]):

BEGIN pid host key agent_name tag event_type event_subtype name_1 value_1 name_2 value_2 … END;

The **pid** is the process id of the agent reporting the event. The **host** is the host on which the agent resides. The **key** is an element used by the agent for communication purposes. The **agent_name** is the name of the agent reporting the event. The **event_type** and **event_subtype** define the kind of event being reported. Following the **event_subtype** is the **name-value** pair list. All event parameters are passed in this list. In case there are more classifications of the kind of event, they can also be passed as name-value pairs.

The rule for such an event in the rule base will have the following format:

ON;

pid ‘host’ ‘detector name’ ‘event type’ ‘event subtype’

CONDITION;

```
if ('boolean expression');
ACTION;
begin;
action 1;
action 2;
…
end;
END;
```

The boolean expression compares the name-value pairs against defined thresholds. The expressions use normal relational operators for comparison. For simplicity in parsing, these relational operators have been replaced by equivalent keywords. For example "gt" replaces the greater than comparison operator. Simple expressions can be combined to form more complex expressions with the help of the "and" and "or" operators. If the condition is satisfied, then the list of actions is fired. There can be multiple actions for a given rule.

This rule format and the corresponding event reporting syntax, provides a simple but powerful way of expressing response actions to given events.

3.5 Analysis of EGIDEM Architecture

The EGIDEM architecture meets most of the requirements of an IDS for large networks. The desirable characteristics of an IDS as listed in Balasubramaniyan et al. [4] are as follows.

1. *It must run continually with minimal supervision*. This is the case with our architecture.
2. *It must be fault tolerant in the sense that it must be able to recover from system crashes and re-initializations*. This has been implemented. The heartbeat signals are one of the mechanisms of detecting an IDS crash on another host. In response to a crash, the IDS

on another host can restart the IDS. Furthermore, the crash of an IDS on a host, does not affect the functioning of the IDSs on neighboring hosts

3. *It must resist subversion. It must be able to monitor itself.* While this has not been implemented currently. It will not be difficult to have a detector periodically match the IDS code against a one way hash stored in a secure location and raise an error when a discrepancy is found.

4. *It must impose minimal overhead on the system where it is running.* Our system allows for this by making the core IDS components lightweight. Processing power is required mainly by the detectors and response agents. The architecture does not require the presence of all detectors and response agents. Hence the system can be tailored according to needs.

5. *It must be able to be configured according to the security policies of the system.* This is not currently implemented, as there is no way for security policies to be transformed into rules that can be used by the IDS.

6. *It must be able to adapt to changes in system and user behavior over time.* The knowledge sharing mechanism allows for data and information about attacks and new signatures to be propagated to the whole system.

7. *It must be able to scale to monitor a large number of hosts.* This is done easily with the EGIDEM architecture.

8. *It must provide a graceful degradation of service.* Bringing the IDS down on one system does not affect the rest of the system. Also the detectors and response agents can be easily turned on or off without affecting the rest of the IDS.

9. *It must allow dynamic reconfiguration.* The core IDS components have configuration files that can be changed while the IDS is running. The IDS components periodically scan these files for changes and accordingly adjust themselves. As far as the detectors are concerned, by building interfaces for them, we can have complete control over the detector functionality.

3.6 Summary

This chapter describes the EGIDEM architecture. It shows how the model can be scaled to a distributed system. It explains the reasons for the EGIDEM design decisions. It also compares our system against the general requirements for an IDS.

CHAPTER 4
KNOWLEDGE SHARING

In the previous chapter we took a look at the distributed architecture for the IDS. This chapter presents a brief overview of the techniques for sharing information gathered at the various IDSs. The design and implementation of this information sharing model is explained in detail in Chengalvarayan [6].

4.1 Design of Knowledge Sharing Mechanism

Every cell in the distributed system has a dedicated host called the Knowledge Store. This host logs all events reported by detectors in its cell. Knowledge stores in different cells form cells between themselves and transfer information about cell activities between themselves. Thus the events happening in one cell are propagated to all other cells in the system.

By this sharing of information the IDSs on different cells learn about how to react to attacks. They do this by dynamically updating their rule base with the rules of the host that experienced the attack. Thus they know what agents to execute in response to a new attack. By propagating information on attacks to other parts of the network, it is possible to analyze network activity on a global scale. This scheme also prevents information about an attack from being lost.

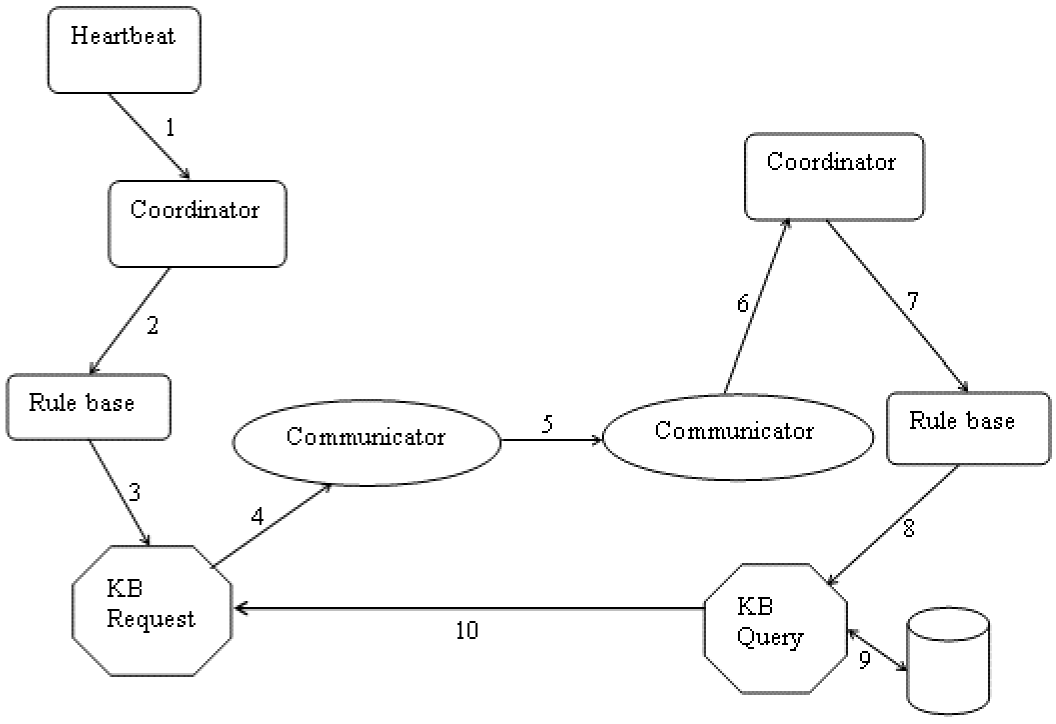

Figure 4.1 Information Flow in the Knowledge Store Architecture [6]

In case the IDS on a particular host is killed, another IDS can take over the host and retain the same state, by looking at the most recent events that were reported by that host to the knowledge store. Even if the knowledge store is attacked, the other stores in the network can contribute to the recovery process, with the information that they have about that cell.

This design also allows for new rules to be propagated to all hosts of the network. Thus it is easier to update the rule base on each IDS. In addition to this, knowledge stores can be queried for more information about an attack, by detectors and response agents. This allows for specialized sensors to formulate new patterns of attacks, by looking at events in different cells. Another good feature that arises from this design is that IDSs in remote parts of the network can learn about new attacks and prepare themselves by turning on their sensors and updating their rule base to handle the attack.

To support information sharing, the architecture was augmented with two agents called the Knowledge Request (KR) agent and the Request Service (RS) agent. These agents are present on each knowledge store. The Heartbeat agent sends out periodic events to the coordinator that trigger the KR agent. The KR agent then sends out a query to the remote knowledge store. On the remote knowledge store, the RS agent responds to this query by sending the appropriate matching events from its database to the requester agent. The KR agent then updates the local database with the events that are sent.

As mentioned in Chapter 2, IDSs generate a huge amount of information. In our design, the events reported by one cell get transferred on the network segment on which the cell members reside. Having all of this information present on a common database for every 5 – 10 hosts that form a cell will not cause much of a performance hit. But propagating it to all other hosts in the network is definitely a waste of network bandwidth. It will also allow a rather simple denial of service attack to be carried out on networks that our IDS protects. All the attacker would have to do is to excite one of the IDS hosts. The excessive overhead of event propagation will render the rest of the system to useless.

Hence we decided to send only a digest of the events reported to other knowledge stores. This digest is a mere fraction of the events present on the knowledge store. In case a particular cell member on a remote cell needs to know more about an attack, it can query the local knowledge store directly for more information. This can either be done using the KR and RS agents discussed above or directly by the detector that needs the information.

4.2 Summary

This chapter presents the design of an information sharing mechanism between the IDSs in our distributed architecture. We saw that information transfer is done using two special agents designed for this purpose. This is a periodic event triggered by the Heartbeat agent. We also transfer only a digest of the events on each information request to improve performance.

CHAPTER 5
CONCLUSION AND FUTURE WORK

This thesis proposes an architecture for distributed intrusion detection. It extends the Generic Intrusion Detection Model (GIDEM) architecture to scale up to a distributed setting. The extended architecture addresses a number of issues faced in today's intrusion detection systems when applied across a large network with hundreds of hosts. This architecture has been implemented in PERL. However, the implementation still has room for the following improvements:

1. Security is required for the communication mechanism between hosts.
2. There must be a technique for authenticating detectors and system components to each other. This should be practical enough to be applied to a distributed system.
3. The system needs a network management tool in the form of a Graphical User Interface for the administrator to effectively manage the IDS.
4. We also require state management for the IDS on different hosts, so that they can start from where they were before the attack.
5. A graphical interface to show system activity and alerts will also be a good addition.
6. The latest detectors have to be integrated into the system.
7. The rule base can be modified to support the Common Intrusion Detection Format (CIDF), so that the EGIDEM system can follow current system standards.
8. Mobile agents infrastructure can be built in to improve efficiency where having the IDS present on every host is not feasible.

The above improvements will let the EGIDEM implementation to be used in a practical scenario. Once these improvements are implemented, we can create a distribution package and have it tested on larger networks. This thesis has attempted to contribute to standardizing IDS technology and building a framework for easy integration of newer technologies.

LIST OF REFERENCES

[1] D. Anderson, Th. Frivold and A. Valdes. Next-generation intrusion detection expert system (NIDES): A summary. SRI-CSL-95-07, SRI International, Menlo Park, CA, May 1995

[2] F. Apap, A. Honing, S. Hershkop, E. Eskin and S. Stolfo. Detecting malicious software by monitoring anamalous Windows registry accesses. In *Proceedings of Recent Advances in Intrusion Detection*, pages 36-53, Zurich, Switzerland, October 2002.

[3] M. Asaka, S. Okazawa, A. Taguchi and S. Goto. A method for tracing intruders by use of mobile agents. In *Proceedings of INET*, pages 87-99, San Jose, California, June 1999.

[4] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff and E. Spafford, D. Zamboni. An architecture for intrusion detection using autonomous agents. In *Proceedings of the Annual Computer Security Applications Conference, IEEE Computer Society*, pages 13-24, Scottsdale, Arizona, December 1998.

[5] S. Balasubramanian. An architecture for protection of network hosts from denial of service attacks. Master of Science Thesis. Computer and Information Science and Engineering. University of Florida, Gainesville, FL, 2000.

[6] M. Chengalvarayan. Knowledge sharing in distributed generic intrusion detection model. Master of Science Thesis. Computer and Information Science and Engineering. University of Florida, Gainesville, FL, 2002.

[7] Cisco Systems Inc. Cisco Catalyst 6000 intrusion detection system module, 1992. WWW page at http://www.cisco.com/warp/public/cc/pd/si/casi/ca6000/prodlit/6kids_ds.htm (accessed August 2002)

[8] D. E. Denning and P. G. Neumann. Requirements and model for IDES – A real-time intrusion detection expert system, technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, 1985

[9] S. T. Eckmann. Translating snort rules to STATL scenarios, 2001. WWW page at: http://www.raid-symposium.org/Raid2001/papers/eckmann_raid2001.pdf (accessed September 2002).

[10] A. Gandre. Implementation of a policy-based intrusion detection system - generic intrusion detection model (GIDEM version 1.1). Master of Science Thesis. Computer and Information Science and Engineering. University of Florida, Gainesville, FL, 2001.

[11] G. Helmer, J. Wong, V. Honavar and L. Miller. Automated discovery of concise predictive rules for intrusion detection. Department of Computer Science, Iowa State University, Ames, IA, 2001.

[12] K. Illgun, R. A. Kemmerer and A. Porras. State transition analysis: A rule-based intrusion detection approach. In *IEEE Transactions on Software Engineering*, Vol. 23, No.3, pages 181-199, March 1995.

[13] A. K. Jones and R. S. Sielken. Computer system intrusion detection: A survey. Department of Computer Science Technical Report CS-99-17, University of Virginia, Thornton Hall, Charlottesville, VA 1999.

[14] J. Kim and P. Bently. The human immune system and network intrusion detection. Department of Computer Science, University College London, Great Britain, 1999.

[15] C. Krugel and T. Toth. Sparta – A security policy reinforcement tool for large networks. Distributed Systems Group, Technical University Vienna, Argentinierstrasse 8, A-1040 Vienna, Austria. 2001.

[16] W. Lee and S. J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, USENIX, pages 79-84, San Antonio, TX, January 1998.

[17] E. Nemeth, G. Snyder, S. Seebass and T. R. Hein. UNIX system administration handbook, 3rd Edition, Upper Saddle River, New Jersey, Prentice Hall.

[18] S. Patton, W. Yurcik, D. Doss. An Achilles' heel in signature-based IDS: squealing false positives in Snort, 2001. WWW page at: http://www.raid-symposium.org/raid2001/papers/patton_yurcik_doss_raid2001.pdf (accessed August 2002).

[19] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493. WWW page at http://www.sdl.sri.com/projects/emerald/emerald-niss97.html. (accessed August 2002)

[20] M. Roesch. Snort: The open source network intrusion detection system, 2000. WWW page at http://www.snort.org (accessed January 2002)

[21] M. M. Sebring, E. Sellhouse, M. E. Hanna, R. A. Whitehurst. Expert system in intrusion detection: A case study. In *Proceedings of the 11th National Computer Security Conference*, pages 74-81, Baltimore, MD, October 1988.

[22] Security Focus, 1999. WWW page at http://online.securityfocus.com/ids (accessed August 2002)

[23] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip and D. Zerkle. GrIDS – A graph based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, Vol I, pages 361-370, October 1996.

[24] G. Vigna, R. A. Kemmerer, NetStat: A network-based intrusion detection approach. In *Proceedings of the Annual Computer Security Applications Conference, IEEE Computer Society*, pages 25-38, Scottsdale, Arizona, USA, December 1998.

BIOGRAPHICAL SKETCH

Akhil Narayan Karkera was born in Udipi, India. He received his Bachelor of Engineering Degree in computer engineering from the University of Mumbai, India, in May 2000. He will receive his Master of Science degree from the University of Florida, Gainesville, Florida, in December 2002. His research interests include computer networks, network security, and distributed systems.