AN ARCHITECTURE FOR PROTECTION OF NETWORK HOSTS FROM DENIAL OF SERVICE ATTACKS

By

SEETHARAMAN BALASUBRAMANIAN

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

2000

Dedicated to my parents,
Balu and Dhaksha and
my sister, Vidya

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. Richard Newman for his constant encouragement and support and the opportunity to work on this research project. He has been a great source of inspiration and my mentor.

I would also like to thank Dr. Randy Chow for being my co-advisor and Dr. Abdelsalam Helal for willingly agreeing to serve on my committee and for giving their valuable suggestions and feedback.

I express my gratitude to officemates Paramtap Desai, Wenqiu Zhang and Anna Brown for their endless discussions on the subject.

I thank my parents and sister, for their inspiration and support throughout my academic career.

I would also like to thank all my friends, far and near for being there to help me always.

TABLE OF CONTENTS

page

LIST OF TABLES

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

AN ARCHITECTURE FOR PROTECTION OF NETWORK HOSTS FROM DENIAL OF SERVICE ATTACKS

By

Seetharaman Balasubramanian

August 2000

Chairman: Dr. Richard E. Newman
Major Department: Computer and Information Sciences and Engineering

As we prepare ourselves to take a joyride in the ubiquitous computing world of the 21st century, we must also be prepared to face the challenges from the netherworld of hackers, who have now more avenues to reach us. The recent "denial-of-service" attacks that infested the major Web sites are signs of an unsafe Internet. This thesis is a step to unify the forces of intrusion detection to combat the plague of attacks. Intrusion detection has become a significant focus of research in the security of computer systems and networks. A number of commercial and free intrusion detection systems (IDS) are available and more are becoming available all the time. Some products are aimed at detecting intrusions on the network (network based IDS); others are aimed at host operating systems (host based IDS), while others are aimed at applications.

The purpose of the thesis is to define an architecture that is able to interact with the various ID systems so that it is possible to realize the individual ID systems as a whole and issue policy-based responses. A rule-based coordinator has been developed for

the same. The policy-driven coordinator interacts with the various agents (analyzers or detectors) and issues responses. For this, an appropriate method to express the policy unambiguously is needed. The policy must address the issues relating to configuration of the agents (monitoring profile) and the countermeasures (response profile) taking into account the entity in which they are enforced (target behavior). A language that addresses these issues has been formulated. The data formats and exchange procedures for sharing information of interest among the entities are also discussed.

CHAPTER 1
INTRODUCTION

1.1 Problem Definition

"Attack of the Zombie PCs"--this is not a title of a movie under production. Rather, it is a strategy used in the Internet world, whose damage may exceed more than any Hollywood production. The recent "denial of service" problems that plagued the major Web sites were reminders of how much the Internet has become part of our lives--and of the dangers of relying on a technology still in its infancy. Intrusion detection (ID) systems have become an ineluctable component in the rapidly growing e-commerce world. ID systems use various strategies to suspect or detect attacks. Though an ID system is a cynosure in the security world, they are despised as the resource-consuming ogre because many ID systems work as a 24/7 watchdog. The thesis focuses on defining an architecture that can dynamically deploy highly distributed and independently tunable agents and issue policy-based responses. The larger goal is to develop a Generic Intrusion Detection model (GIDEM) and the paper attempts to take baby steps to achieve the same. Well, a goal properly set is halfway reached.

1.2 Motivation

February 2000 was probably a memorable month for hackers, for they were instrumental in flashing the CNN headlines “Amazon.com Floods out,” “Bye Bye Buy.com,” “eBay: Going Going Gone,” courtesy of a Distributed Denial-of-Service

attack, Stacheldraht (German for barbed wire). The Internet is becoming the Wild West of cyberspace and a fear grips us as to who will be the next victim.

When we refer to attacks, it is necessary to define what we are trying protect and against what. The resources that we want to protect are the ***processes, files*** and ***data in transit***, on computers and networks. Resource objects, such as databases or semaphores must also be protected.

For such resources, the main security elements are confidentiality, integrity and availability [1]. **Confidentiality** refers to protecting information from being read or copied by anyone who has not been explicitly authorized by the owner of the information. **Integrity** protects information from being deleted or altered in any way without the permission of the owner of that information. **Availability** refers to protecting one's services so they are not degraded or made unavailable without authorization.

Cohen [2] terms the attack against the security elements as **disruptions**, which he specifically calls ***leakage*** (opposed to confidentiality), ***corruption*** (opposed to integrity) and ***denial*** (opposed to availability)

The first step in the development of a comprehensive architecture to detect and respond to computer and network security attacks and incidents is to define computer security [3].

Computer security is preventing attackers from achieving objectives through unauthorized access or unauthorized use of computers and networks.

In the world of security, intrusion and denial-of-service are common terms. To draw the thin line between intrusion and denial of service attack, we define the terms ‘Intrusion’ and ‘Denial-of-Service’.

An ***intrusion*** can be defined as any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource.

A ***denial-of-service*** (DoS) attack is an incident in which a user or organization is deprived of the services of a resource they would normally expect to have [4]. The major problem with DoS attacks is that they are easy to launch and detect but hard to prevent. The targets of such DoS attacks are usually users, computers or the networks. With the use of Internet and the attacks on the rise, it is necessary to equip us against these attacks.

1.3 Need for a Generic Intrusion Detection Model

As mentioned earlier, DoS attacks are easy to launch and they are targeted on a system's resources. Depending on signatures of the attacks, we name it Melissa, ILOVEYOU, etc. As new vulnerabilities and exploitations are found, new signature-based detectors are created and proper responses are suggested for the same. Hence, a need to embed the ability to integrate the new attack analyzers and relate them with meaningful responses is realized. Rather than trying to develop a centralized system that is capable of detecting all the attacks (which might be infeasible), it is better to develop a model that can control independent agents that have a well-defined analyzer to detect or to suspect an attack. It is to be noted that a resource hungry analyzer need not be more effective. After all, a more expensive tennis racket will not make you a better player.

It must be possible to tune the agents and made run on an "as needed" basis at the required level. Otherwise, all the analyzers run all the time and consume enough resources to perform the DoS on itself (suicide). At the same time, a global approach can effectively differentiate between an isolated attack attempt and a concerted attack. On detecting or suspecting attacks, meaningful responses may be issued. In other words, a

coordinator that can control such entities is desired. The coordinator may receive the findings of the analyzers and can perform high-level correlation. It is evident that the findings of the analyzers are heterogeneous depending on the data analyzed, their computation capacity or even vendor-specific. The key issues here are the message exchange format (between the various entities), monitoring and response profile. Hence a suitable language to express the message, attack and response is necessary. It would be desirable to express these with a single entity, rather than framing a format for each. The thesis focuses on developing architecture and a language for expressing the same.

1.4 Organization of the Thesis

Chapter 2 gives the classification of the well-known ID systems and explains the various strategies employed. Chapter 3 describes the overall architecture of the system. Chapter 4 discusses the implementation details. Chapter 5 presents the testing results and performance of the implementation. Chapter 6 gives concluding remarks and suggestions for future work.

CHAPTER 2
BACKGROUND AND PREVIOUS WORK

2.1 Introduction to Denial of Service and Intrusion

Denial-of-service, in a nutshell, is an effort to prevent users from being able to use their systems. Most of the denial-of-service attacks exhibit in the form of destruction (of resources), process degradation, storage degradation, process shutdown, system shutdown, etc. Attackers use the tools to access information and achieve their objective. It is necessary to identify who are the attackers, what are the tools they may use and what are the results they achieve in order to build a fortified system to prevent them. After identifying potential attacks, the responses may be issued based on security policy.

2.2 Sources of Attacks

The monitoring profile can be effectively tuned if the weak points of the system and the possible sources of the attacks are identified. Intruders who try to gain access or disrupt the use of the system can be classified in two categories: outsiders and insiders.

Outsiders are intruders from outside our network who may attack our system (mar web servers, forward spam through e-mail servers, etc.). They may also attempt to con the firewall to attack machines on the internal network. Outside intruders may come from the Internet, dial-up lines, physical break-ins, or from a partner (vendor, customer, reseller, etc.) network that is linked to the corporate network.

Insiders are intruders who legitimately use our internal network. These include users who misuse privileges (such as a social security employee who marks someone as deceased because they disliked that person) or who impersonate higher privileged users (such as using someone's terminal). A frequently quoted statistic is that insiders commit 80% of security breaches [3].

Intruders may also be characterized as **joy riders** who hack because they can; **vandals** are intent on causing destruction or marking up the web pages; **profiteers** are intent on profiting from their enterprise, such as rigging the system to give them money or by stealing corporate data and selling it.

In general, the primary ways an intruder can get into a system are physical, system and remote intrusions.

Physical Intrusion

If intruders have physical access to a machine (i.e., they can use the keyboard or take apart the system), they will be able to get in. Techniques range from special privileges the console has, to the ability to physically take apart the system and remove the disk drive (and read/write it on another machine). Even BIOS protection is easy to bypass: virtually all BIOSes have backdoor passwords.

System Intrusion

This type of hacking assumes the intruder already has a low-privilege user account on the system. If the system doesn't have the latest security patches, there is a good chance the intruder will be able to use a known exploit in order to gain additional administrative privileges.

Remote Intrusion

This type of hacking involves a intruder who attempts to penetrate a system remotely across the network. The intruder begins with no special privileges. There are several forms of this hacking. For example, an intruder has a much more difficult time if there exists a firewall on between him/her and the victim machine. Note that Network Intrusion Detection Systems are primarily concerned with remote intrusion.

Of these, the physical intrusion can be eliminated with proper security measures. System and remote intrusions require serious considerations since it might possibly involve the world and the organization's policy might affect a larger community.

2.3 How Intruders Get into Systems

The vulnerabilities like the software bugs, design flaws, easily guessable passwords, etc may be used to break into the system. Some of these vulnerabilities are discussed below.

Software Bugs

Software always has bugs. System administrators and programmers may never track down and eliminate all possible holes. Intruders have only to find one hole to break in. Software bugs are exploited in the server daemons, the client applications, the operating system, and the network stack. Software bugs are usually caused by buffer overflows, un-handled and unexpected inputs. These factors are discussed below.

Buffer overflows

Most of the attacks may be attributed to this problem. It is not uncommon to notice the limited buffer size allocated for certain attributes (say login-name), e.g., char login[256]. A common assumption is that no one may use a name longer than 256 chars. A hacker, if he/she is able to identify such limitation may send a login-name with 350

characters and voila, he/she has broken into the system. These problems are more prevalent in C/C++, but rare in Java programs.

Unexpected combinations

A common hacking technique would be to enter something like "| mail < /etc/passwd". The input might be meaningless for one stage but meaningful to the underlying OS. By that, the OS intervenes when it encounters a '|' pipe stage and might mail the password file to the intruder.

Unhandled input

A valid input to an application is usually handled. Many application programmers do not consider what happens when an input that does not match the specification is given.

System Configuration

Many systems are vulnerable because of improper configuration. Part of this problem lies with the administrators, who rely on the default configurations or configure it incorrectly thereby creating a hole in the system. The following factors are responsible for causing vulnerabilities in the system.

Default configuration

Almost any UNIX or WinNT can be hacked in if the default configuration is not altered, because the 'easy-to-use' default configuration unfortunately is 'easy-to-break-in'.

Lazy administrators

Attackers are people and so are victims (utlimately). The bitter truth is that many of the attacks could be avoided if the administrators have properly configured the system. Even delays in setting a new password could be exploited.

Hole creation

All programs can be run in non-secure mode. Inadvertently, some administrators open a hole in a machine. Security auditing packages may be able to locate these holes (potential detectors in a ID system). Even the hacker is looking for a hole or trying to create one in your system.

Password Cracking

In the movie “Wrongfully Accused”, we would see Leslie Nielsen breaking into a health-care computer system by entering the login name as “login” and password as “password”. Though funny, it is a bitter truth that many users use weak passwords like “hello”, “welcome”, “system”, “password”, etc. In many cases, well-known and easily guessable public information of the user is used as the password. Such weak passwords are vulnerable to dictionary attacks and other brute force attacks.

Sniffing Unsecured Traffic

It is possible to break into client machines by passively observing the packets in “promiscuous” mode. With pcap (packet capture library) support and root access to a system, it is possible to monitor the packets in the Ethernet. Usually, this library might not be effective in switches.

Design Flaws

Sometimes the software in a system might be error-free, but the design itself might lack security features. Flaws in TCP/IP and the UNIX system pose significant challenge to trim down the vulnerabilities.

TCP/IP protocol flaws

ICMP echo reply, three-way handshake, IP fragments are effectively exploited for smurf, SYN flood, and teardrop attacks. The IP protocol was designed as “trusting”;

hackers can change the information (e.g., IP spoofing). IPSec has been developed to overcome this. An active network approach to deal with IP spoofing has been discussed by Halbig [5].

UNIX design flaws

The access control system is a chief problem. The possibility to break into the system with 'root' access creates an unsafe situation giving an opportunity for the users to abuse their privileges by trying to impersonate as 'root'.

2.4 Typical Intrusion Scenario

A Hacker is no voodoo priest able to control any system as a zombie at his/her will, outright. The hacker usually goes through a series of steps (though not strictly applicable) to gain entry to or disrupt a system service.

A common scenario in National Geographical Channel would be the moves of a tiger hunting a prey. The tiger initially hides behind the shrubs and looks for its potential prey (**outside reconnaissance**); after identifying a potential victim, it moves slowly towards its prey (**inside reconnaissance**) -- note that the prey is not yet harmed; when the victim is unsuspecting of the danger, the tiger paces and moves directly towards the prey (**exploit**); and before the victim could realize the danger, the carnivore grips (**foothold**) the victim and devours it.

A hacker goes through similar phases trying to attack a victim. Without revealing his/her identity, the intruder tries to find public information of a normal user (**outside reconnaissance**). DNS utilities, web page information come handy for this. After gaining enough information, without harming the victim, the hacker scans for more information moving slowly closer to the system (**inside reconnaissance**). Utilities like ping sweep, TCP/UDP port scans, snmpwalk, etc., may be used to achieve this. After identifying a

weak point in the system, the intruder **exploits** the holes in the system, e.g., gains 'root' access after breaking into the system; Once the weak points are exploited, the intruder gains the required access and a **foothold** on the system and the intruder may profit by stealing confidential data, misusing the data or disrupting services.

Sometimes, intruders might not be attacking a specific site. For the fun of it, they might issue an attack (say SendMail DEBUG hole) to any vulnerable system in the Internet. If we analyze the possible tools that an intruder might use in the afore-mentioned scenario, we might realize that these include **UNIX utilities** like ping, traceroute, nslookup/dig, whois, finger, rpcinfo, showmount, SAMBA, telnet as well as **WinNT utilities** like nbstat, net view, which any normal user, might employ. It might look hard to separate a hacker from a normal user, but **hacking** utilities like sniffing (tcpdump), port scanners, ping sweepers, war dialers are difficult to conceal from effective ID systems.

2.5 Intrusion Signatures

It is possible to identify an intrusion even in the reconnaissance stage. However, there may be a potential increase in the number of false positives (cry wolf), the earlier we attempt to identify a potential attack.

In the **reconnaissance** stage, ping sweeps, DNS zone transfers, e-mail recons, TCP/UDP port scans, account scans, and CGI scripts walk-through is usually observed. In the **exploitation** stage, Web server attacks, Web browser attacks, IP spoofing etc., are prevalent. In the DoS attacks, the situation is worsened with attempts to crash a service (or machine), overload network links, overload CPUs or fill disk space. The common attacks like Ping-of-Death, SYN Flood, Land, WinNuke etc., might be observed.

2.6 Security Policy

Proper expression and enforcement of security policies are the key issues in any system. There are two basic philosophies behind a security policy (an information access policy). This can be used to design the information flow within the system and prevent sensitive information leak.

- **Prohibitive:** Anything not explicitly permitted is denied. It is used in system in which security is of utmost concern, e.g., military applications.
- **Permissive:** Anything not explicitly denied is permitted. A malicious user can exploit it to break in the system if not properly configured.

Any rule in white paper serves no purpose unless it is properly enforced.

2.7 Anti-Intrusion Approaches

Anti-intrusion techniques [6] consider the less explored approaches on the periphery of "intrusion detection" which are independent of the availability of a rich audit trail, better known as intrusion detection techniques. Less considered have been other complementary anti-intrusion techniques that can play valuable roles. Some of them are as follows.

- **Prevention** techniques (enforced internally or externally to the system) seek to prevent or at least severely handicap the likelihood of success of a particular intrusion, e.g., like hiring bodyguard in a rough neighborhood to prevent possible attacks. In a real world system this technique alone proves untenable and is unlikely to be implemented as a foolproof defense against intrusions.
- **Preemption** techniques strike offensively prior to an intrusion attempt to lessen the likelihood of a particular intrusion occurring later. Rather than the reactive defenses

offered by detection and countermeasures, preemption refers to proactive action against the source of as yet unlaunched intrusion.

- **Deterrence** seeks to make any likely reward from an intrusion attempt appear more troublesome than it is worth. Deterrents encourage an attacker to move on to another system with a more promising cost-benefit outlook.
- **Deflection** leads an intruder to believe that he has succeeded in an intrusion attempt, whereas instead he has been attracted or shunted off to where harm is minimized.
- **Intrusion Detection** encompasses those techniques that seek to discriminate intrusion attempts from normal system usage and alert the system.
- **Countermeasures** actively and autonomously counter an intrusion as it is being attempted.

Intrusion detection is more popular than the rest of the anti-intrusion approaches.

2.8 Detection Models

Various detection models have been recognized based on various characteristics. Some of the classifications are based on source information, intrusion, event generation, event processing, and the approaches. Some of these classifications are explained below.

Based on Source Information

The detection models are classified as network-based or host-based depending on the source information. Special devices might be necessary to monitor the information from some sources.

Network-based

Usually, the network-based intrusion detection system uses the raw network packets from a network adaptor as its data source. It can monitor the network traffic in promiscuous mode (passive protocol mapping or 'sniffing') and analyze its data either in

real time or in batch mode (using tcpdump and analyzing the same). Usually superuser privileges are required to sniff the packets from the ethernet.

Host-based

The common practice is to review the audit logs of the OS for suspicious activity. System, event, and security log in Windows NT or syslog in Unix environment typically records events of interest. A popular method of detecting intrusion is to check key system files and executables via checksums at regular intervals for unexpected changes (principle of Tripwire) [7]. The timeliness of the response is in direct relation to the frequency of the polling interval (an important factor in the monitoring profile).

A variant of host-based IDS is the **multi-host based** ID system. Here the data are collected locally in each host and the data are processed/analyzed centrally, e.g., configuring the 'syslog' in UNIX to report the events to another 'syslogd' in the powerful analyzing server machine. This feature is available in the NSTAT [8]. In a way, multi-host ID system, imitates the network-based ID system, to analyze the data in promiscuous mode (the real-time traffic of network) - except that the data analyzed is more host specific. It is possible to correlate the events from the multi-host traffic, e.g., the disk filled space in the /var/mail/ in all the hosts denote a "mail-bomb" targeting all the systems. In some cases the ordering of the events are critical, depending on the analyzing tool.

Strengths of network-based ID system

1. It lowers cost of ownership since it may be deployed on a single host.
2. It detects attacks that host-based systems miss, e.g., network-based SYN Flood, Teardrop, SMURF, Land, etc.

3. It is more difficult for an intruder to remove evidence. Live network traffic is captured in real-time and analyzed. It is not as easy as manipulating audit logs of an OS.
4. It is possible to realize real-time detection and response.
5. It can detect unsuccessful attacks and malicious intent.
6. It enjoys platform independence.

Strengths of host-based ID system

1. It can verify success or failure of an attack.
2. It can monitor system specific activities, e.g., deletion of /etc/passwd.
3. It detects host-specific attacks that the network-based ID system misses, e.g., data that do not cross network.
4. It usually requires no additional hardware (cost-effective).
5. It attempts to provide a near real-time detection and response.

Need for both network-based and host-based detection

It is desirable that the next generation IDS include integrated host and network components. A combination of the two approaches will improve network resistance to attacks and misuse, enhance enforcement of security policy and provide greater flexibility in deployment options.

Based on Intrusion Classification

Intrusion may be classified into two main classes: misuse and anomaly.

Misuse intrusions are well-defined attacks exploiting the known weak points of a system. They can be detected by watching for certain actions being performed on certain objects. This detection technique is also called knowledge-based intrusion detection.

Anomaly intrusions are based on observing a deviation from normal or expected behavior of the system or the users. There is a need here to define normal behavior (or self) and distinguish it from abnormal behavior (or non-self) [9].

To define normality, a profile of the system being monitored is first built and any significant deviation from this profile is detected and flagged as suspicious. Anything not seen previously may be deemed suspicious - this detection technique is also called behavior-based intrusion detection. In a nutshell, this is the paranoid approach.

Misuse ID systems know what they are looking for and can perform effectively, e.g., analyze events for signatures of a SYN Flood attack or 'fingerd' and 'sendmail' bug exploitation. Since they are focused on identifying well-defined signatures, they have no role to play for identifying unknown future intrusions. Hence, they need to be updated to identify new vulnerabilities and exploitations to take preventive or corrective action. Knowledge about attacks is very focused, dependent on the operating system, version, platform, and application. The resulting intrusion detection tool is therefore closely tied to a given environment.

Anomalous ID systems can detect attempts to exploit new and unforeseen vulnerabilities. They can even contribute to the (partially) automatic discovery of these new attacks. They have the inherent 'fuzziness' to define normality. The problem encompasses identifying the proper feature set (unnecessary information must not be emphasized and critical information must be not be neglected or given less importance), proper encoding (quantify the thresholds), which aid in building the profile. Further, an anomaly may be a symptom of a possible intrusion. Anomalous intrusions are hard to detect because the analyzers are not searching for something specific (unlike misuse IDS).

Hybrid detectors adopt some complementary combination of the misuse and anomaly detection approaches run in parallel or serially.

Based on Event Generation/Gathering

Depending on how (note, not the source) the ID system is able to gather or generate event, the IDS may be classified as event-based and poll-based systems.

- **Event-based:** This is also known as passive mapping. The detector watches the event stream continuously, e.g., sniffs the network (in promiscuous mode from the network adaptor). A packet for that entity usually identifies the creation of a new connection or service for the first time. Each packet in a network stream may be considered as an event.
- **Polling:** This may usually be referred as active mapping. The detector may look for conditions at regular events. It usually runs a test program or queries a variable at regular intervals. It is inherently suitable only for certain types of monitoring, e.g., resource monitoring. In some cases, it might not be effective or efficient, e.g., to check whether a specific user logs into the system, it is possible to issue 'who' and check for the user - but the frequency of polling interval directly maps to the effectiveness of the system. A regular check on system's resources might be an expensive operation.

In a generic model, a poll-based monitor is an event-based collector triggered by a timer event.

Based on Event Processing

Depending on when a generated event is been processed, the ID system is classified as real-time and batch mode.

- **Real-time:** This is usually the favored mode of operation. This is characterized by a faster response time.
- **Batch:** The events are collected over a time-period and then analyzed at regular intervals. Events whose impacts are not time-critical can be processed like this. Though a less favored option, it is preferred for its lesser resource consumption.

Based on Approaches

Various strategies for intrusion detection not unique to the field, but rather derived from applications established by other fields: knowledge based expert systems, pattern-recognition algorithms, statistical profiling techniques, neural networks, Bayesian statistics, information retrieval algorithms, state transition models, Petri-net techniques, and so forth.

Broadly these approaches may be classified as statistical, expert system based and formal language-based approaches. These approaches are discussed below.

Statistical

This approach is based on the statistical intrusion detection model of Dorothy Denning [10]. This approach is known as the 'comportmental model'. It tries to respond to the question “Is the user's behavior normal according to the past?” The statistical model is relevant in anomaly detection to build a profile of the subject using statistical measures. The significant deviation from the built model is helpful in identifying anomalous events. A simple model would be to derive a set of threshold to define severity states (say, critical/error/warning/normal). NOCOL [11] encourages the use of threshold measures to assign severity states.

An extension of the same could be realized by building short-term and long-term profiles for a subject based on the CPU, memory, usage, etc (as in the NIDES [12]

statistical component). This profile is matched with the current activity and checked for significant deviation that denotes the intensity of the possible attack. NIDES has four components, **activity intensity** (to determine whether the volume of data generated is normal), **audit record distribution measure** (to determine whether the recent activity is flagged normal), **categorical** and **ordinal** measures (to determine whether a particular activity is significantly different from its past, e.g., a file access). The key issues in building the short and long-term profiles are the number of samples (measures) used, the stress on the recently acquired values over the past thresholds to assign a severity state (red - critical / yellow - warning) and the number of times these states were crossed in the past.

In some cases, a Bayesian probabilistic model is built for a subject and the new activity is checked for deviation from the predicted bounds.

In general, the statistical approach leads to some problems. The choice of parameters is tricky. Further, the statistical model leads to a flow of alarms in the case of noticeable environment modification (inability to adapt quickly to legitimate changes in user's behavior). The formidable challenge to be faced in this approach is that it is possible for a user to slowly change his/her behavior to cheat the system ("slow poisoning approach").

Expert systems, AI, GA

AI is the simulation of human intelligence processes by machines, especially computer systems. These processes include learning (the acquisition of information and rules for using the information), reasoning (using the rules to reach approximate or definite conclusions), and self-correction. In a lighter vein, it is remarked, "a year spent in artificial intelligence is enough to make one believe in God." A particular application of

AI is expert systems. AI provides significant benefit to ID through data reduction, the ability to analyze a collection of data to identify the most important components, and classification, the process of identifying intruders. An **expert system** is a computer program that simulates the judgment and behavior of a human or an organization that has expert knowledge and experience in a particular field. Typically, such a system contains a knowledge base containing accumulated experience and a set of rules for applying the knowledge base to each particular situation that is described to the program. EMERALD [13] p-BEST is an expert system shell with a set of rules that enable to identify or suspect an attack in the system. The rules are continuously checked when the audit record changes to note the state of a possible intrusion. With respect to ID, a fact maps to an event that is recorded an evaluated by expert system. p-BEST [14] uses a forward chaining system.

The idea of **Genetic Algorithms** is to simulate the way nature uses evolution. The GA approach uses survival of the fittest with the different solutions in the population. The good solutions reproduce to form new and hopefully better solutions in the population, while the bad solutions are removed. Three genetic operators achieve the iterative process of population creation: selection (selects the fittest individuals), reproduction/crossover (promotes exploration of new regions of search space by crossing over of individuals) and mutation (protects population from an irrecoverable loss of information). GASSATA [15] explores the events in an audit trail that present the greatest risk in the system using the security audit trail analysis by GA approach.

Formal language-based compiler/interpreter technology

The latest approach is based on compiler/interpreter technology. It is based on the pattern matching techniques. The set of audit events can be seen as an alphabet, each

audit event as a character, the audit-trail a main string and the scenarios as sub-strings to locate in this main-string. Thus the attack is represented as a regular expression and pattern matching can do detection. A state transition approach (which may be boiled down to a regular expression) as used in STAT[16], USTAT[17], NetSTAT [18], Graph-based detection trying to build a graph with particular events, as used in GrIDS (Graph-based Intrusion Detection System) [19], and Petri-Nets approaches are variants of the same.

Thus, these classifications try to define the various dimensions of an intrusion detection system, though the agents composing the ID system are a hybrid derived from these classifications.

2.9 IETF (Internet Engineering Task Force) Model

A typical intrusion detection system consists of the following components (illustrated in Figure. 2.1) as defined by the IETF [20].

The raw information that an intrusion detection system uses to detect suspicious **activity** is collected from a **data source**. Common data sources include (but are not limited to) raw network packets, operating system audit logs, and system-generated checksum data. The **sensor** collects instantiations of data source (activity) and presents it as a formatted event to the **analyzer**. The analyzer processes the events and alerts the **manager** when an unauthorized or undesired or unusual activity has occurred. The manager notifies the operator for response or issues automated response for the **alerts**. The Security policy framed by the administrator governs the activity of the sensor, analyzer and the manager.

In the architecture, the following points do not matter

- Whether the sensor and the analyzer are integrated or separate.

- Whether the analyzer and manager are isolated, or embedded in some large hierarchy or distributed mesh of components.
- Whether the manager actually notifies a human, takes action automatically, or just analyzes incoming alerts and correlates them.
- Whether a component might act as an analyzer with respect to one component, while also acting as a manager with respect to another.

Thus modules representing these components might be distributed in the system as threads or processes. It is essential to maintain the information flow among the various entities. Thus the IETF model summarizes the requirement of a generic ID system.

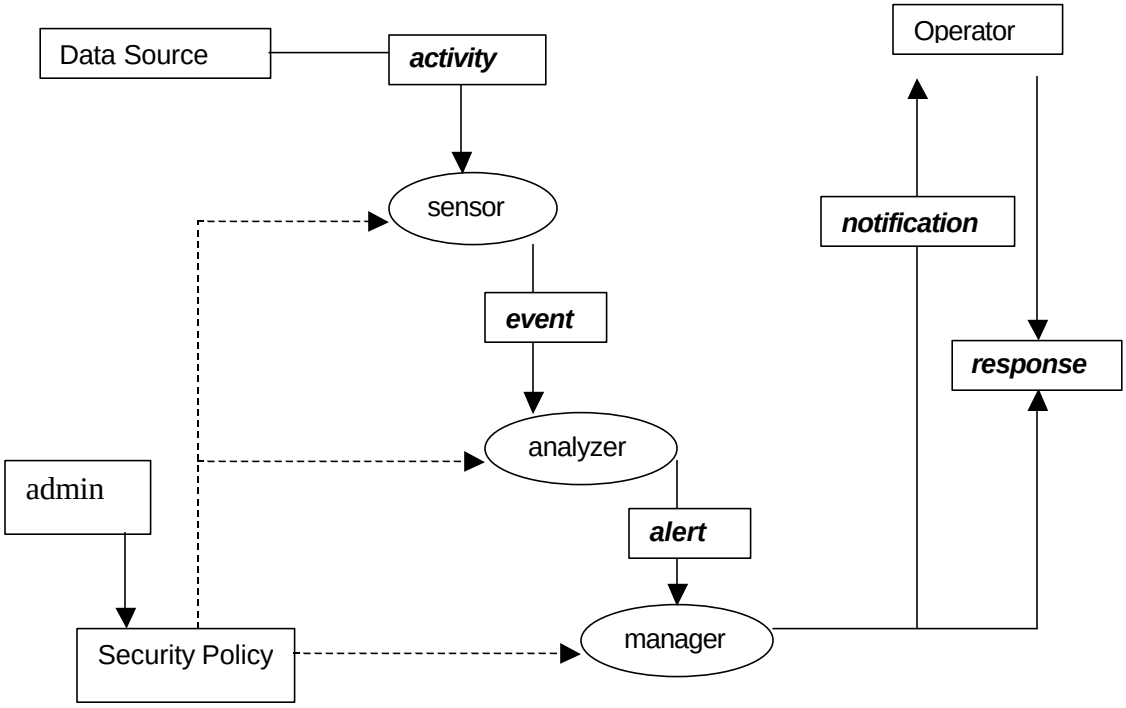

Figure 2.1 IETF model [20]

2.10 Intrusion Detection Systems

Intrusion detection systems attempt to detect and prevent intrusions. Many commercial and free ID systems are available [21,22]. Some ID systems are platform dependent (NOCOL is an UNIX-based system); others are portable models (EMERALD). New technologies such as the agent technology and the data-mining techniques are also employed nowadays for ID systems. Some of the ID systems and approaches are dealt in this section.

2.10.1 NOCOL (Network Operations Center On-Line)

NOCOL [11] is a system and network monitoring software that runs on Unix platforms and monitors reachability, ports, routes, system resources, etc. The interesting feature in NOCOL is its modularity, the ability to start, stop the detectors without affecting one another. This is also related to the extensibility of the system by the addition of new detectors. In fact, there is a perlnocol module that aids in the creation and configuration of new detectors in the system.

NOCOL features

- NOCOL assigns severity states by using thresholds for avoiding false alarms. Currently, three thresholds are used to categorize a variable into one of the four severity states.
- It employs **incremental** data storage that acts like a differential detector that stores only when the severity of an event changes. NOCOL detectors beep when severity of an event changes. Thus this feature also helps to detect the persistence of a severity state since the time field is updated when the severity state is changed.

- The users are able to view events from non-graphical interface. Currently, NOCOL supports a curses, Tk and a Web interface (webnocol). It is notable that the information is not replicated to support these UIs.

The user interface is not involved with the collector or the analyzer. It receives its input from the stored data.

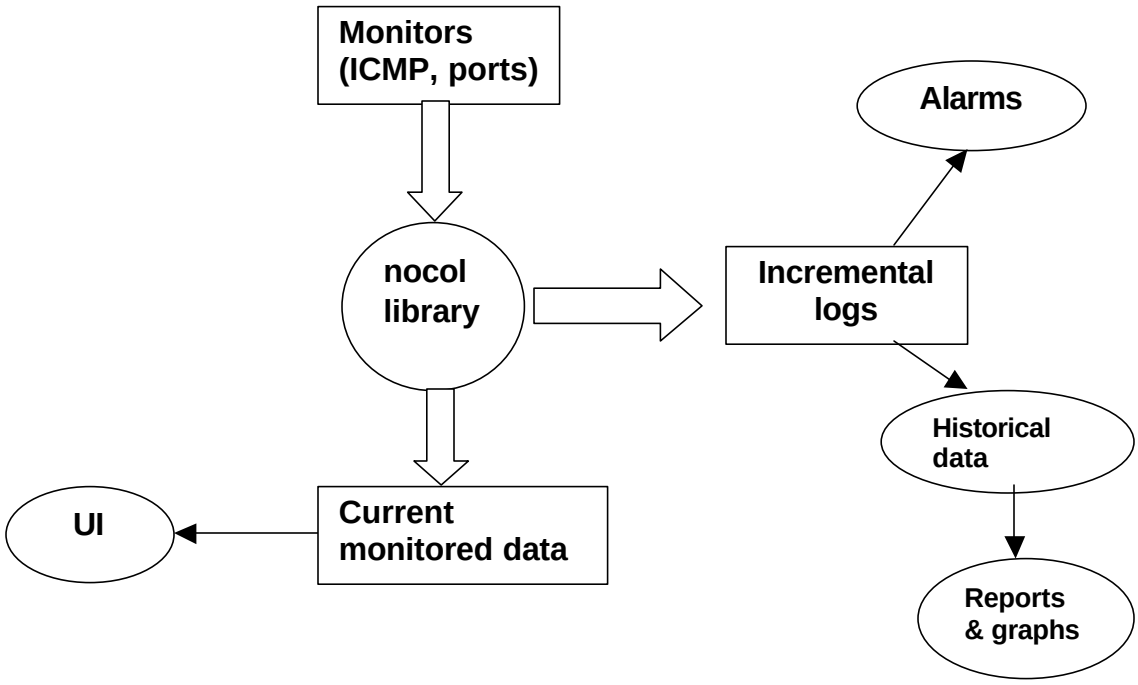

Figure 2.2 NOCOL architecture [11]

The striking feature of NOCOL is its simplicity in its architecture (Fig. 2.2). It is composed of monitors that poll at regular intervals to gather events (data value of a variable). The monitors are sensors, analyzers and managers of NOCOL. The data value is compared with threshold and assigned a severity state (critical/error/warning/normal). The event is represented as a tuple (device name + device address + variable name, value, severity state, threshold, timestamp), the first entity being the unique key. Notification

(SMS pager, send email, perhaps even run some automated tests or open a trouble ticket) may be user-configured for an event.

Some of the available **monitors** are RPC portmapper, Ethernet load, TCP ports, Nameserver, Syslog messages, Mailq, NTP UPS (APC) battery, Unix host perf, BGP peers, SNMP variables, Data throughput, etc and the list is extensible via C/Perl modules.

It is also possible to allow distributed, platform-specific data collection and remote processing. The collected raw data may be sent to a remote monitoring server host that processes the data, e.g., hostmon receives raw host performance data collected by the local hostmon-client (may be platform specific). Thus a per-service entity may be present in the system. The simplicity of NOCOL enables it to be integrated with other systems. The monitored data may prove the source of information for other analyzing tools that may be selectively triggered based on the severity.

2.10.2 AAFID (Autonomous Agents For Intrusion Detection)

AAFID [23] is a distribute intrusion detection system developed by COAST [21], Purdue University. The model is composed of three entities, viz., agents, transceiver, and monitors. The user Interface is separated from data collection and processing. The architecture consists of autonomous agents (software entities), which are independent and monitor a specific event. They report their findings to the per-host entities transceiver, through system IPCs like message queues, named pipes, shared memory or sockets. The transceiver has a data processing and control role. In its data processing role, it reduces the data reaching the monitor. In its control role, it starts, stops and sends message to the agents (for reconfiguring). For reliability, the agent may report its findings to multiple tranceivers. This helps in the event of a failure of a transceiver.

The monitor is the highest-level entity in the architecture. Its role is similar to transceiver, except that the transceiver is involved with per-host entities and the monitor has control across the host boundaries. The user interface interacts with the monitor. An interactive user interface that can control the operation of the monitor is available. The AAFID has been developed with PERL. The architecture of AAFID is illustrated in Figure. 2.3

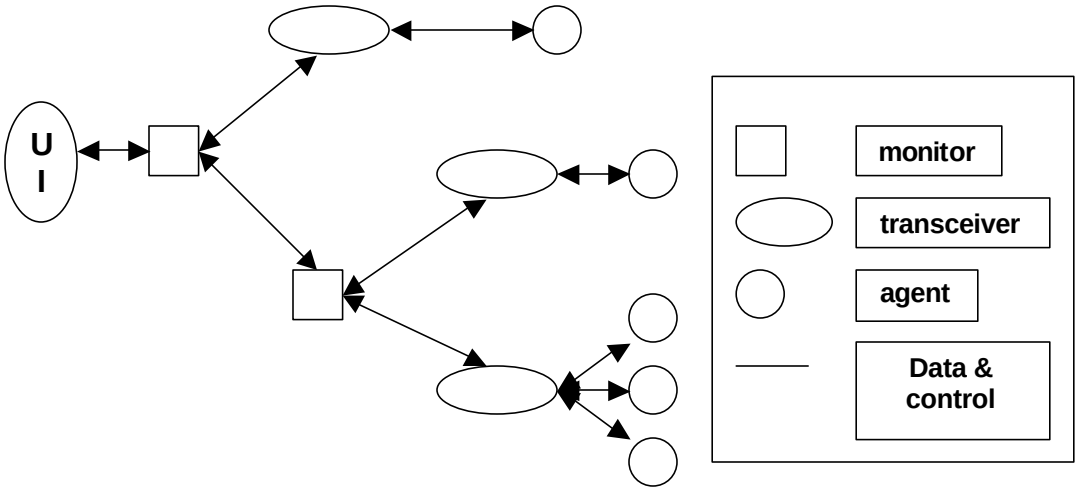

Figure 2.3 AAFID architecture [23]

The interesting feature in AAFID is its flexibility in introducing the analyzing engine in the architecture. Suggestions to include mobile agents in the architecture are also popular.

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

SRI International began research into an intrusion expert system and has come up with a commendable EMERALD [13]. The objective of EMERALD is to bring a collection of research and prototype development efforts into practical world.

EMERALD environment is a distributed scalable tool suite for tracking malicious activity through and across large networks. The distributed, independently tunable service monitors combine the signature analysis with statistical profiling and provide real-time protection of the widely used network services.

The generic EMERALD monitor architecture (Fig. 2.4) is designed to enable flexible introduction and deletion of analysis engines from monitor boundary as necessary. Separating the analyzing engines and coordinating their functions appropriately achieve a global detection and response capability. The EMERALD environment discards the previous centralized, host-based, user-oriented, intrusion detection efforts that suffer scalability and integration to large networks. It introduces the concept of resource object that embeds the event-collection and storage, analysis and response methods. EMERALD‘s resolver is the implementer of the “response policy”. It correlates the analysis from the analysis engines and implements sophisticated management and control policies. It can also inter-operate with third party analysis engines.

The EMERALD eXpert is a generic signature-analysis engine based on the expert system shell p-BEST [14]. **p-BEST** (Production-Based Expert System Toolset) was originally written by Alan Whitehurst and employed in MIDAS. It was later enhanced at SRI by Whitehurst and Fred Gilham and was employed in IDES [24] and NIDES [12]. p-BEST provides a production rule language that allows users to express the inference formula for reasoning and acting upon the facts. The facts may be derived from external sources (events) or from the other production rules (rule triggers). The p-BEST allows type declarations and rules. Event specific type declarations may be done using ptype declarations. This is used for fact representation. The fact may be added or removed from

expert system factbase. The fact is matched with a rule (antecedent, the if part of a rule) and a match is complemented with the actions associated with the rule (consequent, the then of a rule). Interfacing to standard and user-defined C variable and functions is done through the p-BEST external type declaration mechanism xtype. p-BEST is simple, easy and effective.

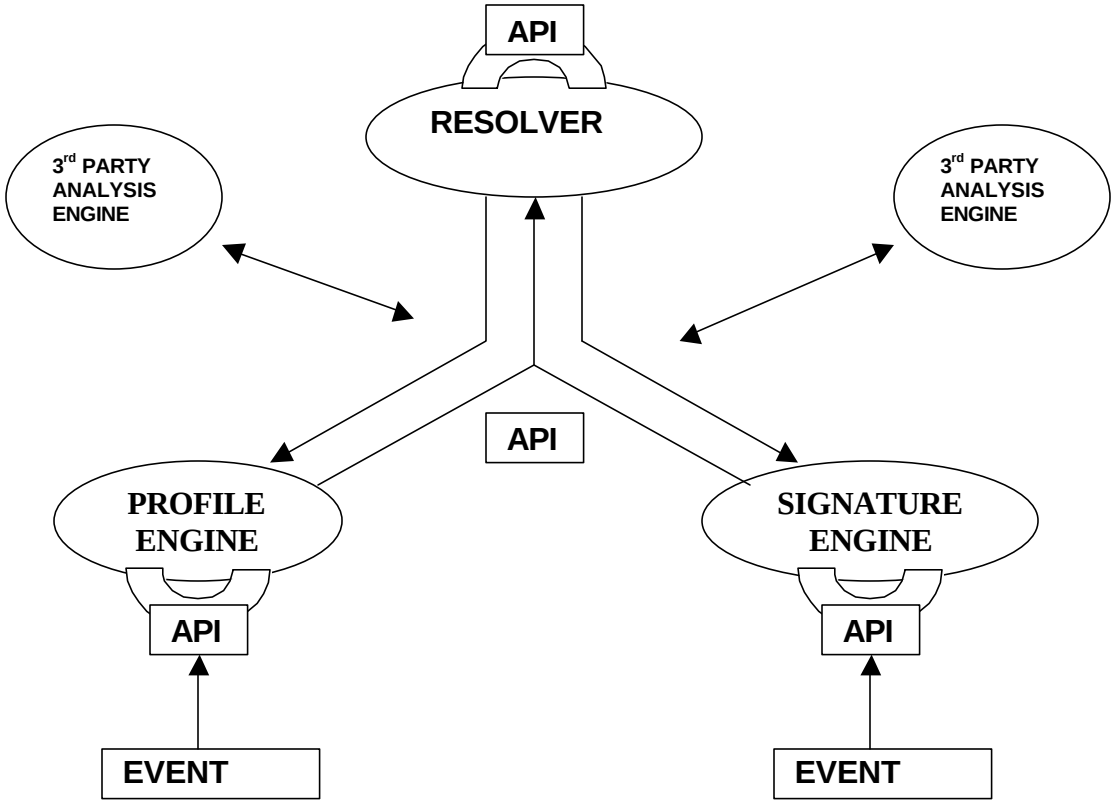

Figure 2.4 EMERALD architecture [13]

2.10.4 Agent Based Protection

Agent technology has introduced a new distributed computing paradigm of broad applicability. Mobile agents are software programs written in a neutral and portable object-oriented or script languages that may be dispatched from a client computer and

transported to a remote server computer for execution. Agents permit one dynamically to extend the computational capabilities to a remote computer. Agents can be particularly useful in providing scalable paradigm to protect distributed systems.

Tailored agents can be dispatched to remote resources to monitor and analyze their activities and detect attacks. The may also perform an equally interesting role in providing target-specific responses. The basic problem associated with mobile agents is to protect the agent from the host (hostile host) and the host from the agent (hostile agent). Since a mobile agent, with a malicious tint may perform like a worm/virus (for protection of which the architecture has been proposed), it must be employed with care. Although it is undoubtedly one of the emerging technologies used to dynamically deploy customized agents efficiently, security in mobile agents is a serious issue to be addressed.

2.10.5 Data Mining Approaches

Though the rules (conditions) may be expressed in a flexible manner, identification of the clauses composing the rules is still not automatic (currently framed by manual analysis). It is possible to employ data mining concepts to realize rule discovery and learning [25]. Basically, the event information is maintained in the in-memory database and it is possible to apply the knowledge discovery techniques that employ any of the following techniques to achieve its objective: machine learning, statistical, pattern recognition, etc. Basically, it is like identifying the new kid in the block, i.e., identifies a classification or a category from the large set. The rule framing for anomalous events may be achieved through data mining techniques. Though a clear misuse intrusion signature discovery might not be probable (not to be read as impossible), it would be flagged as an anomalous event and a fuzzy rule for the same, probably matching the actual may be derived.

Thus, some of the intrusion detection systems like the UNIX-based NOCOL and generic models like the AAFID and EMERALD were discussed. The main feature plausible in these models is their extensibility. Controlling (initiating, terminating and reconfiguring) independently tunable agents is the main objective. The growing agent technology that enables to deploy customized agents at remote location and collect the state of the distributed system, though remarkable must be carefully analyzed for hostile agents and hostile hosts. Though a rule-based system is favored, it might be necessary to employ data mining techniques to learn these rules.

2.11 Summary

Intrusion and denial of service attacks have become serious problems in the Internet community. In many cases, inside threats are severe than the outside threats in firms. Intrusion may be physical, system or remote. Hackers try to identify the vulnerabilities in the system and break into it. Software bugs, system configuration, weak passwords, etc are weak points that aid the intruders. Intrusions may be identified by the initial reconnaissance stage itself. Various anti-intrusion techniques are in practice among which intrusion detection is far more favored than the rest. Various intrusion detection models have been developed based on the source information, event generation, event processing, intrusion and approaches. IETF models an intrusion detection system composing a sensor, analyzer, and manager. NOCOL, AAFID and EMERALD are some of the expanding list of intrusion detection systems. After studying the various models, let us explore the architecture of our system in Chapter 3.

CHAPTER 3
ARCHITECTURE

The purpose of this thesis is to develop a real-time, generic intrusion detection model (GIDEM) that is able to deploy dynamically highly distributed, independently tunable heterogeneous agents for DoS attack detection and issue policy-based responses. It is necessary that the system take care of the heterogeneity in the data analyzed, the events reported and the policy- and time-based responses.

The main components in the system are coordinator, detectors, transceiver and response agents (as shown in Figure. 3.1.). The detectors sense the activity and report their findings to the transceiver and coordinator that analyze the events and trigger the response agents for appropriate actions.

3.1 Coordinator

The coordinators are the highest-level entities in GIDEM. The coordinator is an event-driven system. The coordinator has data and control processing roles. As a data processing agent, the coordinator parses the events from the detectors. As a control agent, it controls or tunes the activity of the detectors and response agents. The coordinator receives the findings (events) from the analyzers (transceiver/detector), checks the rule-base whether the event triggers an action, and issues the specified response to the appropriate response agent(s). Basically it is an **ECA** (Event-Condition-Action) model. A rule is a (condition, action) pair. The coordinator is driven with a configuration in which all the elements (viz., event, condition and action) are expressed.

This boils down to expressing the monitor profile (concerning the detector) and response profile with the same rule handled by the coordinator. The coordinator maintains the state information and updates its state on the arrival of an event E.

```
event E + old state S -> new state S'
```

The rules are applied for the new state S' and checked for the condition (in the rule). If the condition is matched, the corresponding action is issued. The action may control the activity of a detector (start/stop/reconfigure) or realize a response (run). In some cases, the response may be to notify the administrator. These actions issued by the coordinator are as follows.

Start/Stop Detector

This directly maps with the objective of running the detectors on the "as-needed" basis. If a suspicious activity is noted, then depending on the resources available and the severity of the situation, customized detectors may be triggered. At the same time, it is also necessary that a detector that is no longer necessary must be stopped - otherwise we end up with garden of weeds rather than flowers. For example, when an event (say, a packet in a network) indicates a new connection, a detector to monitor the new connection's activity may be started. The detector is stopped when the connection terminates (a FIN or RST packet is detected in that stream).

Many of the detectors spawned by the coordinator are related and duplicating a detector's function with another detector may be avoided. It is to be noted that such a misconfiguration might be detrimental because duplication of the detector means not only multiplicative wasted effort and resources (e.g., duplicate processing of messages by the coordinator), but also a potential bug open for an attack. After all, what we sow is what we reap.

Reconfigure Detector

This directly maps with the objective of running the detector at the "required" (analyzing) level. It is to be noted that the detectors are either event-based or poll-based based on the event generation/gathering. In the poll-based detectors, that sample the information at regular intervals, the frequency of the polling is a critical factor to be configured. In either case, parsing details may be reconfigured for the detector. Since the event stream might be a large set, the parsing details might help to narrow down the search set for analysis. This is important for the efficiency of the system - narrowing the search to an appropriate set, e.g., in a normal mode of operation, the network monitor may filter/ignore the packets pertaining to the set of trusted hosts; in panic mode, the detector may consider all the packets.

Run

This is usually opted as a “countermeasure” for an attack. Note that the target of the response may be terminal screen, file system, SMTP server, pager, firewall, process (VM related) or any response system. The coordinator triggers the response agents that handle the heterogeneity of the target. This requires the coordinator to have the knowledge of the appropriate parameters passed to the target (or these may be imparted to the response agent). It would be meaningful to base a response based on the target and the time when issued, e.g., if an alert information is to be communicated to the administrator, a mail at midnight might not be that helpful; paging the administrator would be a more favorable suggestion. In the usual working hours, displaying an alert on the console would be favored over sending e-mail.

Usually automatic responses (without human intervention) could be prioritized over calling for human interaction. A nagging system is usually despised, e.g., when a file system integrity check reports an error, a response to reload the original file may be first taken and then let the system admin know, rather than informing the system admin and expecting him/her to take the action. The key is the ability to express and send the appropriate parameters to the target via the response agent. The response agent may or may not be involved in the parsing of the parameters reaching the target.

It is quite noticeable that compared to IETF model, the coordinator plays a part of the analyzer and the manager. An event-driven, rule-based coordinator that issues policy-based response is the focus of this thesis.

3.2 Detector

The detectors are the eyes and the ears of the GIDEM. They are the low-level, independently running entities in GIDEM. The detector senses an activity in the system. The various classifications of the detection model may be considered as the dimensions of a detector and it is possible to derive a hybrid detector from various compositions. Their primarily roles are data collection and possibly data analysis. It must also have means of communicating with the higher-level entities, transceiver(s) and/or coordinator(s). This may be by shared data (if it runs as a thread), system-specific IPCs (pipes, shared memory, message queues), or sockets (TCP or reliable UDP). Thus in the IETF model, though the primary role of a detector is as a sensor, it may also include a data-specific analyzer. For example, though the NOCOL detectors are primarily performance monitoring tools (sensors), yet they also assign severity states of an activity (analyzer). Some detectors are responsible for maintaining the hysteresis of a condition

(persistence of the condition), some perform correlating activity depending on specific alerts. In such cases, secondary-level detectors that derive their input from a group of detectors may also be present. The fact that the detectors monitor different data sources is the reason for the diversity in the event reported. Unfortunately, it might not be possible to report the event with a single format. Though initial efforts to represent the events with a common NOCOL format, gave some success, it was soon realized that a flexible, extensible format was in need. A common format should allow components from different ID systems it be integrated more readily.

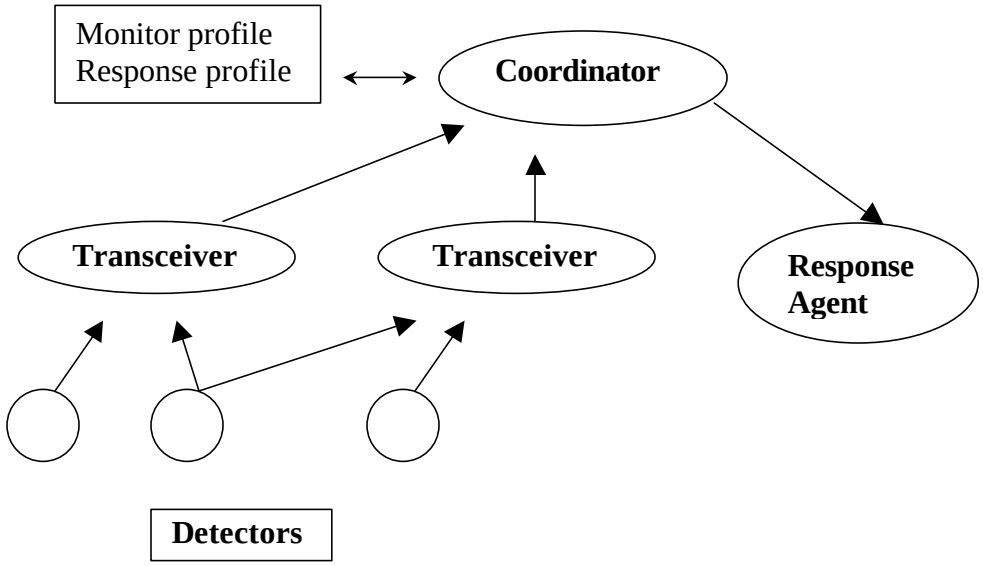

Figure 3.1 Architecture of the system

For example, a network-based system desires a

`(source IP, source port, destination IP, destination port, protocol)`

component apart from the system and attack specific data, whereas a host-based system lays emphasis on (source/attacking host, destination/target system) representation. In addition to collection and analysis, it may include a translator that converts the event to a native format comprehensive for the coordinator.

The detectors must be dynamically re-configured by the coordinator. The extent to which the reconfiguration is possible also depends on the source of the detector. Usually, the third-party vendor detector provides little control on the dynamic configuration at run-time (may be otherwise). Reconfiguration plays an important role in molding the system to perform effectively for a particular situation responding to the dynamic environment.

The primary detectors do not communicate with one another (to remove dependencies among the detectors). The communication among the detectors, if necessary is usually handled by the transceiver.

3.3 Transceiver

This component is an idea borrowed from the AAFID architecture. They are the external communications interface of each host. Thus, they are the per-host entities that have control and data processing roles. In its control role, it is able to start, stop or reconfigure detectors as directed by the coordinator. Hence it must also keep track of the detectors running under its control. In its data processing role, it parses, translates and filters/aggregates the event it reports to the coordinator(s).

Thus, it has an adaptor (which may be external) that is necessary for translating the raw input from the detector. It is notable that this component may be pushed into the detector too. However, third-party vendor-specific detectors may be opaque and the

parsing (translation and/or reduction) is achieved through the transceiver. It is to be noted that the transceiver may behave as per-host or per-service based entity. It is a wrapper for the various detectors and present the "need-to-know" information to the coordinator. It is responsible for the communication between the entities under its control.

3.4 Response Agents

The coordinator takes “countermeasures” on specific circumstances to the target. In the IETF model, it maps to the operator or the automatic response triggered by the coordinator. The target may be a user, node, process, service or the like. The response agents are the executioners of the command from the coordinator. The response agents are intermediates (man-in-middle) between the coordinator and the target. The response agent may be intelligent or dumb. A dumb response agent is one that blindly executes the command from the coordinator. In some cases, this might be sufficient if the coordinator has the suitable response profile. In others, the response profile from the coordinator is distributed to the response agents. In such cases, the command from the coordinator is first analyzed and then executed.

The information maintained by the response agent might be simple (e.g., the communication mechanism to reach an entity) or complex (e.g., hysteresis of the response or the choosing an appropriate time-based communication mechanism for the target). Thus time-specific, target specific responses may be handled by the response agent triggered by the coordinator. It might include a formatter (parser/translator) to deliver the proper information understandable by the target. For example, the IPv4 address format from the coordinator to a firewall might be translated to `source_ip/`

`source_port/ destination_ip/ destination_pt` (four integer format) understandable by that specific firewall.

Further, the user interface is modeled as a response agent. It is necessary to separate the data collection and analysis from the UI. They may usually interact with the coordinator to request for information (or receive information pushed by the coordinator) and format it for the suitable UI. Configuration of meaningful responses is the burden of the administrator. Otherwise the spirit of the model is lost.

3.5 A Generic Entity

In a way, it is possible to conceive each of the components as a generic entity. Each entity (Fig. 3.2) has a collection module, a parser module, a reconfiguration module, a response module, and a fault tolerant module. All these modules closely interact with each other. The complexity of the various modules, give them the status of the afore-mentioned components (detector, transceiver, coordinator or response agent).

3.5.1 Collection Module

This is the sensor part of the entity. The data may be raw, directly from the data source, or it may be a conditioned, derived or analyzed data processed by any external component. The complexity of the module is critical, if the data is collected passively (as in the promiscuous mode in the network traffic). The frequency of sampling (by polling) is important in active data collection (executing queries at regular poll intervals).

The frequency of the traffic, the event size and format, the processing of source-specific information, the number of sources may be the characteristics of the data

collected. These factors hold relevance while defining a collection module of an entity and are explained in the following.

Processing of source-specific information

In general, the primary detectors usually have a complex collector module. They need to analyze the raw traffic and process platform (OS) specific, source-specific data. Two examples follow.

1) telnet login failure detector might need to analyze "syslog" information (the syslog format)

```
Timestamp host service[port, if applicable]: information
```

2) SYN Flood detector needs to analyze the network data (packet information).

Table 3.1 Raw packet content

Ethernet (Data Link)	Source MAC address Destination MAC address		
IP (Network)	Protocol Packet size		
TCP / UDP/ ICMP	TCP : Src pt Destn pt Ctrl flags	UDP: Src pt Destn pt	ICMP: Type Code

The heterogeneity/source-specific information processing is reduced as the data get tendered to a more common format as it transcends transceiver and the coordinator. The string representation of the raw data (event) is usually taken care in the low level (detector). In some cases, such translation of event is not necessary (like the "syslog"). We can trace through the packet information collected by the detector, processed by the

transceiver, reaching the coordinator and that triggers the response agents. The raw packet received by the detector has the format as in table 3.1. (Ethernet packet)

The detector needs to resolve the underlying OS-dependent information (the number of bits allocated each information unit) and the link-specific information (e.g., MAC address, Network layer address). Though ethernet is quite popular, other links have different requirements and representation. Further, the device-specific differences might need to be considered which might require special add-on hardware or software for data collection. Hence a customized detector for specific data collection is necessary.

Whereas a coordinator gets its hands dirty working on more generic information format of the form of (source, destination and event), the transceiver processes information that is partly generic and partly specific. Again the response agent might have target specific information. The burden on the response agent may be increased when the time-based response is handled by it.

Frequency of traffic

The transceiver, coordinator and response agents are usually event-based and receive asynchronous events. It is to be noted that the clock timer is also considered an event. The low level detector is challenged with a stream of rapidly occurring events and it would be unwise to allow the coordinator to deal with each of them. The whole architecture loses ground, if each event from the source is just notified to the higher entity without performing any analysis. In this case, each entity would be just acting as a buffer and it contributes to communication delay.

For the detectors, the frequency may also be governed by a polling interval. In some cases, the events arrive at the rate of microseconds (we are in the age of Gigabit

routers). The traffic of the incoming data directly maps to the processing power requirement of the collecting module.

Event format and size

The size of event representation to be processed holds significance in memory requirements. In detectors, a fixed structure might be used to store the event, whereas in the higher entities, the "need-to-know" information is stored linking to the source event. The average event size and total event stream size dictate the amount of I/O overhead required.

Number of sources

It is possible to extract the same information from multiple sources. While the detector usually has a single source, the high level entities have multiple sources. The transceivers receives its input from a group of detectors and coordinators; the coordinator from various transceivers and the response agents from multiple coordinators. Even a detector might receive its input from different sources, e.g., a syslog detector might receive input from the various syslog dumps.

Thus, a customized sensor that is able to collect the events from the different sources, handle source-specific discrepancies, represent the events and has the processing capacity to handle the traffic defines the collector module.

3.5.2 Parser Module

This is far more one of the important modules in any entity. Any audit trail is usually enormous and a possible attack is hidden in it like a needle in a haystack. Hence it is essential to filter the unnecessary information and analyze the necessary information. This is tricky because there is a danger to filter the critical information that contains the attack evidence. Hence an appropriate filter is necessary. The next problem

is to express this filter to be applied, unambiguously. Some of the means to express the parser constraints are regexp, optional arguments and conditional parameters.

Regular expression

Regular expression is a common method of representing a filter. The data stream format is usually known and the regular expression filters the necessary data to be processed. It has the inherent advantage of clarity and fairly easy implementation (with a finite state machine). Further, it is possible to express many constraints through regular expressions. Some examples are given below.

1) NOCOL uses regular expression to determine the data reaching a 'syslog' detector. Consider a 'service failure' detector, that monitors the number of connection failure within n seconds, and say it is concerned with telnet, sshd, login and ftp services, then a PERL regular expression,

```
\s*\S+\s+\S+\s+[telnet|sshd|login|ftp]\(.*)
```

will be able to filter the appropriate messages (of interest) for the detector.

2) for a Network detector, if the reduced packet information is represented as a string of the form

`(source MAC address, destination MAC address, protocol, source IP, source port, destination IP, destination port, packet size, time, misc)`, a SMURF detector might have a filter

```
(*,*,ICMP, source IP, *, destination IP, *, *, *)
```

since it might be more interested to analyze the traffic reaching a broadcast server (destination) and the ICMP traffic is not associated with source and destination ports and hence they are masked.

In general, the following streams are analyzed,

- connection-based (source IP, source port, destination IP, destination port)
- service-based (destination IP, destination port)
- destination machine based
- destination port based

A common format for expressing the same can be framed with

`<protocol, source IP, source port, destination IP, destination port>`

The protocol is usually an element from the set (tcp, udp, icmp) though in future means to support other protocols may be added.

In the five-tuple format, each entity can be

```
* -> any value
x -> don t care
specific value
```

For example, to express the telnet stream destined to 128.227.170.6 (a service-based detector), `<tcp, *, *, 128.227.170.6, 23>` could be used.

Thus the aggregate ICMP traffic stream would be expressed as

`<icmp, *, x, *, x>` since ICMP does not involve ports.

Hence it is possible to construct a demultiplexer with a series of filters that can determine the data streams reaching the specific detectors. By this approach, the information may be replicated in each detector, which could be avoided by storing it once, as a circular buffer and allowing multiple links from the concerned detectors.

It is to be noted that while regular expressions have been used to filter the raw information (as in the syslog) or the processed information (the raw packet put into an

structure), but it might not be possible to express easily many constraints through the regular expression alone, e.g., it is hard to express the optional/additional arguments in random order or conditional parameters using REs.

This is more useful for coordinator and transceivers, which have a predominant data processing role.

Optional parameters

In some cases it is necessary to express the variables of the format `(<name, value>)*` where the `<name, value>` pair may occur in any order.

Many UNIX commands have the argument list (parameters) expressed in the hyphenated option in any order, e.g., `gcc -o object-file -l library file-name`

where the -o and -l are prefixes to identify the argument name and the next entity is assumed to be the value. In some cases, the argument might have multiple values, e.g., multiple libraries linked. To express this, duplication of the argument option is allowed, e.g., `gcc -o example -lpcap -lpthread example.c` (-l duplicated)

`gcc -lpcap -o example -lpthread example.c` and

`gcc example.c -o example -lpcap –lpthread` would mean the same. This provides the flexibility in expressing the arguments in random order with proper prefixes.

The GET format used in HTTP expresses the URL sends parameters to the CGI in this form `protocol://CGI?(name=value&)*,`

where

`protocol` -> http | ftp | file

`CGI` -> path of CGI executable

`name, value` -> strings

`"?"` is used as a separator for the CGI executable and the arguments and "&" is used as a separator between the various parameters (<name, value> pairs).

For example, `http://www.bmg.com/search.pl?search-type=album&record-type=CD&album=savage+garden.`

Similar to the previous method of expression, the order of arguments usually does not matter. Hence the previous example may also mean the same as

`http://www.bmg.com/search.pl?search-type=album&album=savage+garden&record-type=CD.`

Such flexibility in expressing the arguments is desired in the parser.

The coordinator might receive an event's information (parameters) in any order, depending on the processing of the transceiver/detector.

Conditional parameters

Flexibility in parameter order might not always be sufficient. It might be necessary to specify the condition associated with that parameter as a filter. In most cases, while expressing the optional parameters, the data type information may not be necessary. Once a condition is tagged with the value of the parameter, it must be necessary to know the data type of the parameter, e.g., integer for port-related information; string for IP address/machine, etc. Most of the parameters in the system and network-related information can be covered by integer, float and string data type. It is a desirable feature to add the IP address as a data type `a.b.c.d,`

where `a,b,c,d -> Integers (< 255).`

While analyzing how this conditional parameter can be useful, as an extension of specifying the demux stream, to express the stream of telnet, ftp and smtp stream destined to 128.227.170.6, using the previous approach, an aggregate of the following expression must be realized

```
(tcp, *, *, 128.227.170.6, 21)
(tcp, *, *, 128.227.170.6, 23)
(tcp, *, *, 128.227.170.6, 25)
```

A single expression to express the aggregate could be

```
(tcp, *, *, 128.227.170.6, 21 | 23| 25)
```

which is better. The parser, of course must be sophisticated to handle such complex expression.

Both these features could be combined with operators (&& - and , || - or , ! - not) to express the parser constraint with flexibility and freedom, e.g., to express the udp and tcp streams of telnet, ftp destined to 128.227.170.6 using the combined approach would be

```
<protocol ==  tcp|udp && destination_port ==21|23|25 &&
destination_ip == 128.227.170.6>
```

Other conditional operators like <,>,<=,>= also are required in expressing parser constraints.

New data sets

Some processing entities are concerned only with specific parameters in an event stream. Some entities provide the flexibility to define new data sets/structs, e.g., p-Best allows to specify an event struct **ptype**.

```
ptype[conn_event e_type:integer, sec: integer, seq_id:
integer, client_ID:string]
```

By this, we are trying to narrow the data of interest from the whole event stream.

Thus, the parser module uses regular expressions and conditional arguments to filter the aggregate traffic and might need to handle optional arguments. A fixed, well-defined event format usually involves less processing power; handling optional arguments (variable event format) is computationally expensive.

3.5.3 Response Module

As the name implies, this module handles the countermeasure for any event. Any entity is active; they are reactive to some specific events. It must be ensured that these responses are triggered only when needed, since the response issued by an entity would normally affect the other entity (to respond). For example, a primary detector noting that the disk space is getting filled up might report to the transceiver; the transceiver on analyzing the critical nature of the event has to report to the coordinator which responds by directing the response agent to delete unnecessary files. Thus each entity responds to an event. The response module governs the data flow (of event). It ranges from simply reporting to a high level entity to taking decisions to start/stop/reconfigure other entities.

The low level entities have a "reporting module" as the response. In this case, the target(s) to which the event is reported is of concern. Even in the reporting module, the response might be to dump the information to a target - which need not be a process. Both the logical and physical means to deliver the information to the target is to be maintained. For instance, user is a logical entity to be reached through any device - terminal, PDA, pager, and mail. Suitable formatting of the information is necessary. The information may be filtered/aggregated depending on the target.

Data reduction module and formatting module is therefore present in the response, e.g., dumping information in a pager and mail may be different. A detailed summary is affordable in mail with a suitable header; whereas, a pager requires concise (header) or 911-like information.

If the target is a process, depending on whether it is local or remote, optimal communication interfaces are used. The target may be even the file system in a node, i.e., to dump the information in a file, which may be local or remote. An additional parameter to choose whether the file is stored encrypted and/or compressed by suitable algorithms may be applicable depending on the target. Thus the target may be any of the resources - node, process, file, I/O, memory. Of course, the user is a logical entity.

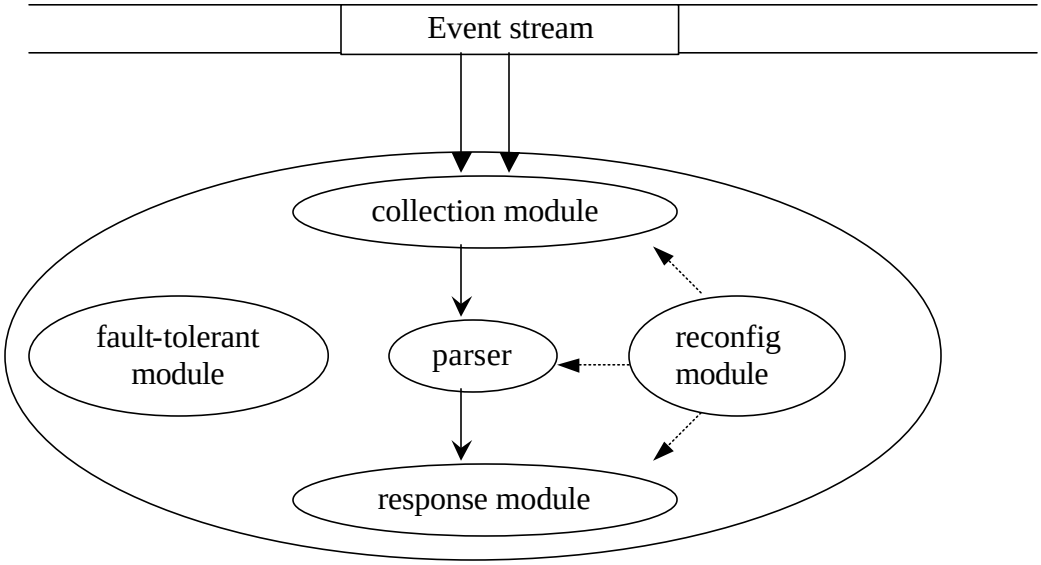

Figure 3.2 Generic entity

The response may also be start/stop/reconfigure other entities or launch programs in a node. Formatting plays a critical role in target OS-specific information. For example, to clean the unnecessary files, the DOS command 'del' may be used for

Windows and 'rm' used for UNIX. Thus target specific commands and interfaces are to be maintained. The decision to launch a new detector as a process or thread depending on the support facility of OS and the programming language used may be taken and started by the response module, though the coordinator decides to start a detector.

Usually the response module is complemented with a memory of the past responses, so that the duplication of wasted efforts could be avoided. This helps to identify the target's reaction to the issued responses. It might be unnecessary to repeat the same response and consume resource when the target is passive to issued response(s). Again a time-based response is a desired add-on feature in the response module. Depending on the target and the time at which the target is actuated, a suitable response might be chosen. The communication interfaces play an important role in the response module.

3.5.4 Fault Tolerant module

Failure detectors are one of the simplest mechanisms for dealing with failures in processes. The concept of failure detector has its roots from developing fault tolerant distributed systems. The various requirements of a failure detector are scalability, flexibility, accuracy, timeliness and low overhead. The failure detector usually recognizes each other by sending heartbeats and checking mutual-reachability. The failure detector has a heartbeat monitor to check mutual reachability and a recovery mechanism.

The failure detector must be able to handle the following issues.

- Monitor a set of processes.
- Inform other failure detectors (in other processes) of the detection of a failed process, so that all the correct processes eventually learn of that failure.

- Inform any local interested module about the processes that fail, and to accept failure-related information from any module interested in reporting a process failure.

There are specific failures that make a process unreachable. Some of these failures are listed below.

- The process is dead (process failure).
- There is a network partition and it is unable to communicate with a set of processes.
- Messages may be lost or suffer delays (link failure).

A failure detector may suspect a (process or link) failure since the detectors themselves run a protocol according to which at least one message should be received from each monitorized process during some specified period. This period can be negotiated between the entities before starting the exchange of heartbeats. This is **direct detection**. A failure detector may also suspect failure because another detector reports the failure of a process. This is **indirect detection**.

The failure detector will suspect processors of failing. The failure detector is permitted to make an arbitrary number of false suspicions, but eventually the following must hold:

- All faulty processors must eventually be permanently suspected,
- At least one correct processor must eventually not be suspected by any processor.

While studying the quality of service (QoS) of failure detectors, the need to develop a specification that quantifies how fast the failure detector detects actual

failures, and how well it avoids false detections. Once a failure detector has suspected a failure, there are various remedial measures that could be taken.

A state transition table expresses the operation of the failure detector. The modes of operation of the failure detector, the states in the FSM, the messages involved are discussed below.

The various **modes of operation of the failure detector** are

- Accept (Seek connection): try to initiate a connection even if the other end is not interested in initiating the connection.
- Accept if asked: does not try to initiate the connection and responds positively to an initiation request.
- Accept if resources are available: accepts or responds to an initiation request for a connection if the resources are available.
- Reject: does not initiate connection and ignores initiation request from the other end.

States

The various states of the proposed finite state machine are

1) Listen
2) Wait_confirm
3) Connect
4) CC_wait
5) Reject

Messages

The various messages involved in the FSM necessary for the state transitions are

1) INIT_REQ destn, max, min, Astart

2) CONFIRM destn, period, Bstart

3) REJECT destn

4) HEARTBEAT destn, n, period

5) CEASE_CONFIRM destn

6) POLL REQ destn, n, <type, target>

7) POLL RESPONSE destn, n, <type, target>

Each connection has two timeout parameters:

- send timeout (t), last sent heartbeat count
- recv timeout (t+2*sigma), last recvd heartbeat count

The coordinator or the interface might decide on whether to maintain a connection with a process or not and send

8) ACCEPT or

9) PROHIBIT/REJECT

message to the failure module respectively.

Each failure detector has a set of parameters that denote its resource level. The number of processes it maintains connection, the number of heartbeats issued per second being the primary resource parameters. The finite state machine is represented as the set of state transition tables in tables 3.2, 3.3, 3.4.

While negotiating for a heartbeat interval between the failure detectors, it might be necessary to assign a leader who has the last say. The function IamLeader() is used for the same. It may use the numeric order of the IP addresses of the entities to identify the leader.

Table 3.2 State transition table for connection initiation and termination

State\ Msg	INITREQ	CONFIRM	REJECT	CC
LISTEN	if(Accept_State()) { k=Resource_avail() if(k withinlimits){ /*accept*/ send CONFIRM k; state = CONNECT; } else { /*re-negotiate*/ send INITREQ; state = WAIT_CONF; } } else send REJECT;	if(Accept_State()) { k=Resource_avail() send INITREQ; state-WAIT_CONF; }	-	-
WAIT_CONF	k=Resource_avail() if(k withinlimits){ /*accept*/ send CONFIRM k; state = CONNECT; } else { /*re-negotiate*/ send INITREQ; }	state=CONNECT	state=LISTEN	-
CONNECT	k=Resource_avail() if(k withinlimits){ /*accept*/ send CONFIRM k; state = CONNECT; } else { /*re-negotiate*/ send INITREQ; state = WAIT_CONF; }	If(IamLeader()){ Send CONFIRM; } else reset frequency;	send CEASE_CONF; state=LISTEN;	-
CC_WAIT	send REJECT;	send REJECT;	send CEASE_CONF; state=REJECT;	state= REJECT
REJECT	send REJECT	send REJECT	send CEASE_CONF;	-

Table 3.3 State transition table for heartbeats and timeouts

State\Msg	HB	Timeout

LISTEN	If(Accept_state()){
	Send INITREQ;
	State=WAIT_CONF;
	}
		-

WAIT_CONF	Send INITREQ;	Decr nRetries Incr TO; If(nRetries < 0) State=LISTEN;
CONNECT	If (expected HB) Reset rcv timer;	Decr nMisses; Incr TO; If(nMisses < 0) State=LISTEN;
CC_WAIT	Send REJECT or NOP	Send REJECT; State=REJECT;
REJECT	Send REJECT or NOP	-

Table 3.4 State transition table for interface messages

State\Message	IF.ACCEPT	IF.REJECT
LISTEN	Send INITREQ; State=WAIT_CONF;	State=REJECT;
WAIT_CONF	-	Send REJECT; State=REJECT’
CONNECT	-	Send REJECT; State=CC_WAIT;
CC_WAIT	Send INITREQ; State=WAIT_CONF;	-
REJECT	Send INITREQ; State=WAIT_CONF;	-

3.5.5 Reconfiguration Module

The objective of the thesis is to deploy agents dynamically at the required computational level. This calls for adaptive entities - which can change behavior when needed. The behavior is determined by the configuration and the re-configuration module must handle any alteration to the initial configuration. The parsing constraints,

frequency of polling, etc are some of the parameters that may be changed according to the situation. An entity might be created or destroyed in the system due to the reconfiguration. In a resource-starving mode, the coordinator might decide to stop non-critical detectors to save resources. Yet, some critical detectors must be created or maintained at required level, e.g., in a resource starving mode, it might not be necessary to track when a new process is created in user mode, but when a process is created in the su mode (root access), it is necessary to create a detector to track that.

An appropriate communication mechanism must be used to tune the detectors instantly. Sometimes, a third-part vendor detector might not be flexible and appropriate adaptors to implement the same must be created. Reconfiguration may also concern the response issued by the entity. Each factor mentioned in the response module might be reconfigured (target, time, hysterisis, response nature, etc). Usually a changing a single factor might also affect a set of parameters, e.g., inclusion of a new target in response module is accompanied with the new parameters specific for the target.

The inclusion of failure detector parameters (number of simultaneous connections, the heartbeat rate, etc) to be changed with the reconfiguration is a design issue. It can well be handled a separate protocol of renegotiation by the failure module itself. Thus, the monitoring profile and the response profile depending on the target behavior may be altered (adapted) dynamically.

The extent to which the various parameters in the collection, parsing, response modules may be altered signifies the control of the coordinator on that entity. Usually, even if a direct control on an entity may not be possible, a suitable interface may be added to achieve the same, e.g., to change a configuration file, kill the process and

restart with the changed configuration. Note that the previous state may or may not be reflected in the restart. The means to express the various parameters are discussed in 3.8.

Secondary Level Detectors

Secondary level hierarchical detectors are initiated by the coordinator on a demand basis, and obtain their data from existing primary/secondary detectors. These detectors form the fuzzy level between the transceivers and coordinator.

Examples of these detectors include the following.

- The profile detector (sampler) that tries to collect the data from the primary detectors and develop short and long term profiles of the target (user/node). A learning detector could also use the data accumulated.
- The differential detector that maintains a sliding window of two events to note the incremental event information.

Thus these tailored detectors may be initiated by the coordinator on demand and fed the necessary information for analysis.

3.6 Communication Interfaces

The coordinator usually employs network interfaces (TCP/UDP) to communicate with the other entities. The transceivers, being the per-host entity are encouraged to interact with the local detectors using the system IPCs (signals, pipes, message queues, shared memory). Some detectors are threads that are executed within the transceiver process, wherein the virtual memory becomes the shared memory.

Thus the various entities of the architecture were reviewed. For implementation, it would be clear if the architecture of the system were supported with a data model specification.

3.7 Data Model Requirements

The events exchanged between the entities are the messages involved in the system. The heterogeneity of the information, the format, the content and the communication interfaces define the data model [20]. Some of these issues are discussed below.

3.7.1 Heterogeneity

Alert information is inherently heterogeneous. The data representation must be flexible to accommodate different needs, e.g, some detectors report limited information like `(source, destination, event-name, timestamp)`, while some may give more detailed context like ports/services, users, etc.

Tool environments and capabilities are different. Different detectors may report different events for the same attack, since their data sources and analyzing strategy are different. Depending on the computational complexity and capability of the analyzer, detailed information of the attack may be imparted with the coordinator.

3.7.2 Message Format

The message sent from the detector/transceiver is termed an event whereas the message sent from the coordinator to response agent is termed a “response”. Due to the heterogeneity of the detectors and the targets, the message content must support the following features.

1. Full internationalization and localization: Usually the ID system may cross geographical and cultural boundaries. The messages might need to be formatted for the local system/human interaction, e.g., the units specified.

2. Support for filtering and/or aggregation of data: the transceiver performs data reduction by filtering or aggregation and the message format must support the same. A rigid, fixed message format would become unpopular and would not last long.

3.7.3 The Communications Mechanism Requirement

The communication mechanism is an important requirement in data model for it forms the backbone of the information flow in the system. The communication mechanism must meet the following requirements.

1. It must be **reliable** -- either an inherently reliable TCP or a custom-built reliable UDP may be used.

2. The **confidentiality** of the message content must be preserved. Suitable encryption techniques are desired depending on the sensitive information.

3. **Integrity** of the message content must be preserved. A suitable message digest mechanism that verifies the proper transfer of the information at the receiver end is a desired feature. An efficient authentication scheme to verify the other end is also necessary.

4. It must **resist** Denial of Service attacks. It is quite possible for any malicious program to attack the collection module with flood of messages. It is necessary for the communication module to resist such attacks.

5. It must resist malicious **duplication** of messages. This is directly related to the previous requirement. Each message must be uniquely defined and duplication of the same must be identified and discarded.

The communication mechanism transfers the information to the entities. It is also necessary to analyze what information is transferred, viz., the content.

3.7.4 Message Content Requirements

The information content that is exchanged between the entities is of concern. It might be meaningless if the same message is passed form the sensor to the coordinator via the transceiver. The information is parsed, filter/aggregated and passed to the higher level. The message content must satisfy the following requirements.

1. It must encompass all types of intrusion detection. The message may originate from a network-based ID system or a host-based ID system or an application. It must be possible to represent the information from these systems.

2. It must support additional detailed data related to event. Depending on the origination of an event, additional data may be added to the event, e.g., aggregation of events representing the profile of performance data, etc.

3. The message must be able to identify source of the event. This helps in verifying the authenticity and confidentiality (degree of confidence of the analyzer) of the message.

4. The representation of different types of targets must be possible with the content.

5. It must indicate possible impact of the event on target. The severity level of the attack helps to decide the response for the same.

6. The recommendation for response may be specified in the message.

7. It must be able to identity of the detector/transceiver that is reporting the event to the coordinator.

8. The degree of the confidence of the report may accompany the message. This may be represented as a numerical value or severity state.

9. It must be possible to uniquely identify the messages. This helps to avoid duplicate message processing and possible attacks.

10. The timestamp may accompany the message. This might be particularly useful when time is part of the feature extraction in events.

3.8 Rule Format

The rule in the GIDEM is able to express the policy with monitoring and response profile based on the target behavior. The general format of a rule (ECA model) is

```
rule -> ON EVENT Condition Action
EVENT -> where detector event event-subtype params
detector, event, event-subtype -> String
where -> String | a.b.c.d
a,b,c,d -> Integer
Condition -> CONDITION if(<exp>) | Null
Action -> action-type a-parameters
```

Here, the action parameters are name=value pairs. This is the key to mould the monitoring and response profile. The extensibility of the modeling the profiles is strengthened by this format.

Further, the expression of the various parameters associated with the various modules of the generic entity must be possible with this. To express a rule common to all sources for an event, it is possible to give a wildcard (a formal parameter that allows the use of the specific source later in the rule) to the source. The wildcard is represented with a “*”, whereas a formal parameter is usually preceded with a ‘?’ to distinguish it

from the other variables. Even though, a ‘*’ can be substituted with any actual value, it cannot be referenced in the condition.

For example,

```
ON ?site hostmon CPUstat null CPUidle=?c
if(<?c,>70>) run program=CPU_overload_response host=?site.
```

By this rule, we are able to issue response (launch the CPU_overload_response) in any site with a 70% CPU usage.

3.8.1 Event Format

The event must have the unique identity of the source (host + detector). The event type follows this and subtype and the parameters associated with the event. Every event is usually associated with a unique event-id.

Examples follow.

1) For a 'new connection' event (triggered by a packet of the connection)

```
2133 128.227.170.6 Network-monitor new-connection null source_ip=128.227.170.6 source_port=23 destination_ip=128.227.170.10 destination_port=25
```

2) For a new process event

```
2322 128.227.170.10 Process-monitor new-process null user=bsraman pid=576
```

3) For a CPU cycle related information,

```
2555 128.227.170.6 hostmon CPU-stat absolute-values CPUidle=70 CPUnice=5 CPUus=10 CPUsy=15
```

It is to be noted that startup of the system/process is considered as an event and it is possible to express a rule of the form

```
ON startup start detector=transceiver node=128.227.170.6 report=UDP port=5500
```

may be used to start a new detector at a specific host and send its contact information. Even a clock event must be considered an event. This is used in the failure detector to send heartbeats at specific clock events and report error for the failure of receiving an event within a timeframe.

The timestamp always accompanies the event. This may be different from the detection time (from the sensor) and the analyzed time (may be from the transceiver/detector to coordinator or from coordinator to response agent). To maintain the entities that may be spread across time zones, it might be optional to add the offset as an element in the time (use Greenwich Mean Time as reference).

In addition, the degree of confidence on the analysis and the possible impact is desirable, e.g., for a port scan attack on a host 128.227.205.228, the event may be represented as

```
1232 128.227.205.228 port-scanner Port-attack null confidence=50 impact=successful-recon time="May 23 2000 12:30" portlist=123-514 action="reconfigure firewall"
```

The analyzing process is the port-scanner. It is also possible to include the intrusion (anomaly/misuse) or analyzing technique (statistic/formal language based/rule-based) in the event. A suggestive action may also be included in the event reaching the coordinator. We need to draw thin line between an event and alert. The rules in the coordinator differentiate them. Thus, only when an event is processed, does it achieve the status of an alert.

When a new detector has started, it registers with the coordinator with its contact information, the variables it intends to report, and detector-specific parameters,

e.g., `1233 128.227.170.10 hostmon MonitorActive null contact=UDP:7500 type=process variables=Dfspace,CPUus,CPUidle poll=5`

In a rule, it is possible to specify an element of condition in the event itself. For example, when specifying the rule, `ON voyager hostmon CPUstat …,` we are inherently specifying that the site must be voyager; the detector must be hostmon and the event must be CPUstat. Usually the patterns associated with the other event related parameters may be specified in the condition.

3.8.2 Condition Format

The condition is modeled after the function semantics wherein the formal parameters are specified with the function name and the processing occurs in the body. Similarly to formal definition the event part in the rule specifies the possible references that can be made in the condition. The actual expression is specified in the condition part. Just like the body of a function can be empty, the condition may be a null expression. In some cases, the mere arrival of an event may be sufficient to trigger an action. Thus the condition expression is not mandatory by this format. Nevertheless, the event information is stored.

The condition is expressed as a combination of &&, || and ! operators. It must be possible to express any condition with the combination with these operators. The traditional operators like <, <=, >, >=, eq, ne are necessary to form a term that may be combined with the relational operators to form an expression representing the condition. It is a regular if expression with each term as `<name, op operand>` format. For

example, `(<?dest_port,==20>||<?dest_IP,eq 128.227.170.6>)` is equivalent to `((?dest_port==20)||(?dest_IP==128.227.170.6))`.

Special functions like

`inlist(element, list) to check whether the element is in the list`

`range(element, low, high) to check whether the element lies within the range`

come handy to represent the conditions. Currently, the data types supported and recognized by the coordinator are integer, float, string and IP address.

3.8.3 Notification/Response Format

This is the most important part of the ECA model. The action part is used to specify the monitoring and the response profile and plays the major role of policy expression. While the monitoring profile is oriented with the activation, termination and reconfiguration of a detector; a response profile is a countermeasure (e.g., launching program) for a target. In both cases, the target behavior is to be taken into account, since the action is realized there.

Thus the rule is defined by the event, condition and response formats. The response format also takes into account the target behavior.

Monitoring profile

To control the monitoring profile, the action types: start, stop, reconfigure may be used. In the case of stopping and reconfiguring the detector, the information of the detector must be available, which must have been stored on starting the detector. Hence, the coordinator maintains a registry of the monitors, which essentially is composed of the name, the host where activated, the type (process/thread), the id, the function

name/executable file, contact information, the variables reported, the failure detector information, the state (active/suspend/dead).

The coordinator initiates a detector by triggering the proper node and entity. If it is executed as a process, the node in which it is activated and the executable/function are the primary parameters in addition to the parameters passed to the executable/function. The contact information of the coordinator is usually provided so that the detector, on startup registers its presence with the coordinator.

For example, to start hostmon,

```
start detector=hostmon host=128.227.170.6 type=process expected-variables=CPUidle,CPUus,Dfspace contact=UDP:5000
```

On startup, the detector responds to UDP:5000 confirming its status and information.

In reconfiguration, apart from identifying the monitor which is the target (with the node and detector), specific parameters which need to be altered are specified in the name=value format.

Response profile

This is usually associated with the 'run' or 'msg' action types. The run action type is usually augmented with the program that needs to launched and the related parameters are specified with the name=value pair(s), For example, an action may be specified as

```
run program=Free_Disk dir=/var/mail/ user=bsraman reduction=10
```

where Free_Disk is a program which deletes or compresses the files in a specified directory and executes this with the specified user privileges. An additional expected level of reduction may be a parameter it takes.

The msg option is usually associated with the logical entity of user. The user profile, namely the communication interface to be used at the time of issuing may be used, e.g.,

```
msg user=admin host=128.227.170.6 event=Disk_space_filled_up impact=70
```

the information may be properly formatted to the target device, the time issued and filtered accordingly.

Thus the monitoring and response profile are defined in the notification format. The event-driven system is configured by the rules, which define the event, condition and action formats.

3.9 Summary

The architecture of the generic intrusion detection model (GIDEM) consists of coordinator, transceiver and response agent. The detectors are primarily involved with sensing the activity. The transceivers filter or aggregate the events from the detectors and alert the coordinator. The rule-based coordinator has the monitor profile (governing the detectors) and response profile (governing the response agents). Based on the events/alerts, the coordinator issues policy-based responses. The response agents issue target-specific, time-based responses triggered by the coordinator. A generic entity composing collection, parser, reconfiguration, response and fault-tolerant modules exhibit the nature of any component in the architecture. The architecture resembles the AAFID architecture while the generic entity resembles the resource object of EMERALD. The implementation of the architecture and integration of other detectors and response agents with the same are discussed in Chapters 4 and 5.

CHAPTER 4
IMPLEMENTATION

This chapter gives the implementation details of the coordinator and integration of the detectors, transceivers and the response agents. The implementation of the coordinator was done in PERL 5.005 as a user-level application program on SunSparc 5.6. The detectors and transceivers were distributed in SunSparc 5.6 and FreeBSD 2.2.2 and were developed in C and Perl.

PERL (Practical Extraction and Report Language)

PERL [26] is a rich language endowed with regular expression and eval features. Both of these features play an important role in the parser module. In the parser module, the event arguments are expressed in random order and conditions are evaluated. The regular expression feature is used for the former and the eval in the latter. PERL 5.005 supports multi-threading that is used for optimization. Further, the coordinator maintains the monitor registry and the state information. PERL supports complex data structures like hashes of hashes and array of hashes that helps a great deal to store, delete and retrieve data efficiently.

The various steps of the implementation are

1 Event generation and gathering

2 Configuration and startup

3 Event processing

4 Rule matching and response

5 Integration

The event generation and gathering are associated with the detectors; the rule matching and response are associated with the coordinator; processing of event is handled by all the entities; integration of detectors, transceivers and response agents are determined by the configuration. The various steps of the implementation are discussed below.

4.1 Event Generation and Gathering

This module involves the implementation of the sensor module of IETF model. The following event streams are considered:

1) Performance data (host-based)

2) 'syslog' audit (host-based)

3) network stream

The NOCOL that was installed and formed the core of Paramtap Desai's [27] thesis is used for the Performance monitoring detectors, CONDEM and the agent- based network monitors developed by Tolga Keski [28] and Sushmala Yarramreddy [29] is used for collecting network data. The event-based detectors developed for syslog by Wenqiu Zhang [30] are used for collecting syslog related data.

Performance Monitoring Detectors

NOCOL and related monitors are primarily created using PERL [26]. Many of the monitors are platform-independent and use (issue) the shell command to identify performance variables and to extract the necessary using PERL. Some of them have been created with C. They have been tested in FreeBSD 2.2.2 and RedHat Linux, though NOCOL supports a wide range of UNIX platforms. Critical resources like CPU, Disk

space, swap space, processes, ports, DNS, BGP routers, etc are monitored by NOCOL and play a vital role in identifying Denial-of-Service attacks and tuning performance in the detectors. NOCOL does not include any specific misuse intrusion detector. The rules are used to identify the same. NOCOL has a particular structure to represent its event

`(sender, site, variable, timestamp, severity, threshold)`

where, the site is usually identified with `(device name + device address)` and the variable is composed of `(variable name + value + units)` as the primary components.

The tuple `(device name + address + variable name)` is used as the primary key in identifying the event. Some of the detectors are poll-based while some dump information at regular intervals.

Network Detectors

The network detector is an embodiment of misuse and anomaly intrusion detectors written as an application program in C. It uses the packet capture library (pcap and libpcap) in UNIX and sniffs the packets from the network tap and customized for the popular ethernet link. The detector demultiplexes the incoming traffic into four streams (connection-based, service-based, destination-based, source-based) apart from the aggregate traffic and starts a detector when a new event stream is detected (e.g., a packet belonging to the stream, a SYN packet). Port hammering and SYN flood detectors have been added to the same monolith to demonstrate the hybrid intrusion detection model.

The raw packet information collected is composed of `(ethernet, IP, TCP|UDP|ICMP)` related information. The demultiplexer usually contains the

`(protocol, source IP, source port, destination IP, destination port)` parameters.

Specific detectors have additional parameters, e.g., SYN flood keeps a count of excess SYN for a machine; port hammering keeps the count of packets for a port over a period of time, etc.

System Audit Detectors

The output of the 'syslogd' is analyzed for specific stream and thresholds are used identify anomalous events, e.g., 50 telnet connections within 5 seconds may not be a good sign. It has been written in C and tested in UNIX (FreeBSD 2.2.2 and SunSparc 5.6). The thresholds play a critical role in identifying an attack, since if misconfigured, undesired results may crop up (which is the curse of the anomaly detector) in general. The syslog dumped file format is the base input for the detectors.

Each of the detectors may be independently functioning entities but the control of these detectors is local at present. The idea of coordinator is to have a centralized control and dynamically control these entities. Basically the reconfiguration and the fault tolerant modules are the missing elements in many detectors. To aid in reconfiguration, a communication interface is to be opened in the entity so that it receives the command from the higher entities. In many NOCOL detectors, the frequency of polling is an important parameter modified. In the earlier network detectors, the decision to start a new detector was done locally (within process); now it is done at the coordinator that triggers the network detector through a communication channel.

4.2 Configuration and Startup

The rules in the configuration file of the coordinator are static, only the parameters to the same differ. Further there are startup events, which are usually the initiation of the detectors under its control. It must be noted that some of the detectors might have been already running and the coordinator needs to update its monitor registry. Hence the monitor registry must have the necessary fields that must be able to maintain the profile of the monitor.

Monitor Registry

The following elements form the monitor registry.

Table 4.1 Monitor registry

Element	**Function**
Name	name of the monitor
where	site of activity
type	Process/thread
Id	process id/thread id
exec/function	executable file in process/ function name in thread
firstrep	time when the monitor first reported
Lastrep	time when the monitor was last heard from
Variables	A hash table containing the variables reported
contact	site and port used to contact the specific monitor
state	active/suspend

Only after hearing the first **MonitorInfo** event from the detector, will the coordinator be ready to receive the events from that detector. Until then, the events from that detector are disregarded.

While parsing the rules, the coordinator realizes the event patterns relevant to a rule. If for processing every rule, the coordinator has to search the entire in-memory database (state), the performance of the coordinator would be drastically affected. However, even in the parsing stage we are able to ascertain the future events relevant to the rule, so an in-memory index for each rule that is able to map the suggested patterns of events of a rule with the maintained database is done. For each rule, the index is also updated.

For example, a rule of the form

```
ON ?site hostmon CPUstat null name=CPUidle,value=?c
name=CPUus,value=?u name=?site,*,Dfspace,severity,value=?s
```

suggests that the rule is interested in patterns

```
<*,hostmon,CPUstat,CPUidle>
<*,hostmon,CPUstat,CPUus>
<*,*,Dfspace,severity>.
```

It creates indices of these patterns so that any event related to any of these arrives, apart from updating its database; it also updates the corresponding index, e.g., For the event

```
23112 voyager hostmon CPUstat null CPUidle=50
CPUus=29,
```

the `<*, hostmon, CPUstat, CPUidle>` and `<*, hostmon, CPUstat, CPUus>` are mapped to the corresponding information. To achieve this the

source and event information are separated. So, hashes of hashes `<site, detector> <event,property>` is used as the key to map the actual value.

Using the reverse mapping feature in hashes, it is possible to use the `<event,property>` to find the set of `<site,detector>` mapped to it.

4.3 Event Processing

After decrypting the event, checking for the integrity and duplication the event is processed. If the event is not in the expected-variables of the detector, the event is disregarded. These are used to eliminate the unnecessary cost involved in rule matching if the event is invalid. The rule in the coordinator is used to segregate the event and the alert. This is realized after processing an event. Though the event may have many parameters, not all the parameters may be used in a rule. A rule may be interested only in a set of the event parameters. Only the necessary event parameters are stored in the state of the coordinator. Since, an event is unique for (site, detector), it is stored in the hash state `("site+detector+variable+property") -> value`.

Further in the rule, we had specified the patterns we are interested while checking a rule. For example, `(?site, hostmon, CPUstate, CPUidle)` denotes that in the future we would be exploring patterns of the form `(*, hostmon, CPUstate, CPUidle).` Hence it would be better to map the specific patterns to the meta-patterns. The search cases are usually involved with identifying the host that matches a criterion, and the response is issued in that host.

The mapping of `<site,detector><event,property>` mapping is also helpful to erase the information when a detector has been terminated. Instead of searching the whole of `<site,detector,event,property>` and deleting the affected

ones, it would be enough to find all the links to the particular `<site,detector>` and remove all the `<event,property>` from the global `<site, detector, event, property>`. The removal can be done by undefining (‘undef’ in PERL) the value associated with hash. There is a difference between assigning 'undef' and null value. Assigning 'null' value recognizes the presence of an expected variable whose value has not yet been received; assigning 'undef' neglects the existence of the variable.

4.4 Rule Matching and Response

Each rule has an event tag that is used to trigger the same. When an event bearing the tag arrives, the appropriate parameters (formal) are substituted with the actual values and the condition is evaluated. Thus, parameter substitution and condition evaluation are the key issues in this section. If the condition evaluates to be true, the corresponding actions associated with the rule are fired. Substitution of the parameter for the arrived event is not an issue, but in some (many) cases, a search for a host that matches a criterion (expressed in the rule) might be required. In that case, every pattern matching the suggested pattern, e.g., `(?site, *, Dfspace, severity)` is matched to find a possible `?site` that satisfies the condition. Thus it may not be a single substitution and condition evaluation; rather, a set of values are substituted and evaluated. Even the response might have a variable tag (e.g., `?site`) which needs to be substituted before issuing the response.

The input variables are received as parameters from the event, so once the formal parameter is bound, it stays bound for the rest of the rule. In the condition part, a parameter is bound on a match, for which there may ne many bindings. Each of these hold for the rest of the rule.

4.5 Integration

The integration of new detectors, analyzers and response agents are the key issues. Any entity that is to be integrated with the coordinator must reveal at least the bare elements of identity and contact information. Components of the collection, parsing, response, reconfigure and failure elements of the entity must be ensured to be present. This is the key of integration. The extent to which detector is configured by the coordinator depends on the parameters that control the various modules of the detector. For example, the parameters associated with the source-type (network/host), source-name (ethernet ei0 or '/var/log/sshd.log' or '/dev/klog' or UNIX command 'df', etc), the parser (filter expressed through regexp), and its reporting target (response config), etc might be used to start the detector from the coordinator or it may deal with application level using the detector-name, exec-name and node. It is necessary to create an adaptor to interpret the information from the detector in a common format or possibly add a module in the detector (if modifiable) to take care of that issue. A reconfiguration input in the form of signal/IPCs/socket communication would be necessary. A failure detector that monitors the health of the detector is also necessary. This may be functioning as a thread in the detector or as a separately running module.

When a new analyzer (may be even a detector) is integrated with the coordinator, it must be noted whether it is possible to reduce the data collection activity. In many cases, the data collection activity would be replicated and this accounts for the wastage of resource. Ideally, if there is a single sensor of data and different analyzer process this single source - it might be efficient. In some cases, the data collection module is combined with a filter so that the input data itself is a demultiplexed stream.

The new responses may be mapped in the action part of the rule. Usually at the application layers, there might be minimal problems associated with configuring response agents. While integrating a new user interface, formatting becomes important. For example, a Web interface, the data must be represented in HTML. A meta language for the same would be of immense use here. A suggestion to use XML for the same is made for its inter-operability and universal acceptance. For integration, the daemon nature and the source of the information of the detectors/analyzers were taken into consideration for demonstration. **fcheck** is a standalone program and is a host based detector, whereas a Weird-connection detector is a daemon program that is a network-based detector.

Integration of Fcheck- - A System Integrity Check Tool

Fcheck [31] is a file system integrity check tool (poor man's Tripwire [7]). The system is provided with the list of files that are considered to be read-only. This can be given with the configuration file by specifying the Directory and Exclusion rules. The Directory rules are used to specify the files and all the files in a specific directory, while the Exclusion rules are used to specify the files not be considered in the specified directories.

A file /etc/passwd can be specified to be considered read-only as

```
Directory  /etc/passwd.
```

If we need to consider the /etc directory (including all the files in /etc), it can be specified as

```
Directory /etc/.
```

If we need not consider /etc/motd in the /etc directory, it can be specified in the exclusion rule:

```
Exclusion /etc/motd.
```

With the configuration file, the fcheck system first creates a digest of the information that can be used to label it as "unmodified". To create this list, the system is initialized with the command `fcheck -ac`. To monitor whether the files have been changed (altered/added/deleted) with the built database, `fcheck -a` is used.

Daemon

In order to integrate this system with the ID system, the `fcheck` was first altered to report its findings in the generic format to the coordinator. Since it was a standalone program, in order to function as a daemon that can continuously monitor the integrity of the system, a daemon `fcheck_daemon` was created. This daemon on initiation first initializes the fcheck database with `fcheck -ac` and subsequently issues the `fcheck -a` command every x seconds (pollinterval). In order to facilitate reconfiguration, the fcheck waits on a port to receive the reconfiguration parameters from the coordinator or response agents

Responses

Suitable responses for modification, addition and deletion of files in the "read-only" area were decided and implemented. When a file is modified, the administrator is issued a warning through a mail. The mail details include the file that was modified and the way to approve of the modification (by initializing the fcheck database). If the fcheck complains of the modification persistently, then after n seconds, the coordinator triggers the response agent to issue a initialization of the fcheck database.

If a file has been removed from the "read-only" area, it is always written back. This happens without the intervention of the system administrator or any notification for the same. Considering removal as a first-priority alert, this action is taken.

If a new file has been added to the "read-only" area, a similar response for modification may be undertaken. An extreme measure could be to delete the file that has been introduced or move it to another location (say, /tmp) for later examination.

Pitfalls

It is desired that the configuration file for the fcheck not be altered too often (i.e., the system is not initialized too often as a reconfiguration). Otherwise the system starts learning the modification. The frequency at which the daemon issues fcheck must be controlled so that the interval is neither too large nor too small.

Integration of Weird-Connection Detector- - A Port Scanner Detection Tool

The Weird-connection detector is modeled after an AAFID detector. This is a honeypot that opens a specific random port and waits for connections. Since it is a port on which connections are not normally expected, when a connection is encountered, it is usually from an intruder. It is possible for the system to also open two or three random number ports in different ranges. It is to be noted that this is not a foolproof port scan attack tool, but only a possibility of a port scan attack. This can be confirmed later with the analysis from the network-monitor to check for the number of attempted connections.

Responses

When a weird connection is noted, the coordinator is notified of the same. No withholding of the information is permitted, since the weird-connection detector is itself based on only a few port connections. When a first connection is noted, it might be sufficient to notify the administrator through e-mail. On detecting subsequent attempted (weird) connections, the firewall may be reconfigured to filter the packets from the source machine's address of the weird connections. Suitable changes to the source ip

address may need to be done. For example, `from 128.227.170.6 to 128/227/170/6.`

Suitable formats may be adapted for firewall reconfiguration. FWTK (FireWall tool kit) from TIS (Trusted Information systems) specify allow/deny hosts in the config file for specific services [32].

Pitfalls

This is a detector prone to Denial of Service attack through the ID system. Since it is possible to spoof the source IP address, the configuration of strict responses would come handy to victimize an innocent source or user. At the same time, when an attack is in progress, it is also necessary first to protect the system and a filter of the supposed source is necessary. The responses may be configured for the system, depending on the policy of the organization.

Integration of the Network Monitor- - A Generic Network Anomaly Detection Tool

Sushmala Yerramareddy did the design and implementation of the anomaly detector. The integration of the same with the coordinator is the central issue. The network anomaly detector is basically a sniffer that monitors the packets in the network adaptor (ethernet) and analyzes the packets. The effectiveness of the detection is explained in Sushamala s work. The network detector consists of the following streams.

1) Connection-based

2) Service-based

3) Destination-based

4) Port-based

A packet may effectively create four detectors belonging to the afore-mentioned streams. This was considered unnecessary and a rule-based creation and termination of

the same was desired. This rule is stored in the configuration of the coordinator. When a packet belonging to a particular stream is first noticed, it is passed onto the coordinator that decides whether to initiate the stream. Hence, subsequent packet information to the coordinator (until it conveys its decision) must be avoided.

Further, the monitor must alert the event to the coordinator and must receive the response (from the coordinator or the response agent). This created an unexpected problem. The network monitor was executed with superuser privileges and it was continuously monitoring the network event stream. It was not able to respond to any other input port (command from coordinator), which is necessary for dynamic reconfiguration.

The first approach adopted to achieve the objective was to use an UDP port and make a thread wait on the same. Unfortunately this approach failed and the waiting thread never got a turn to read the UDP datagram since the master thread reading from the network adaptor never opted to preempt.

The subsequent approach was to create another process that sucks the formatted packet information from the network sensor and analyzes the same. The new process behaves like the transceiver whereas the first process that feeds the information through the pipe behaves as a sensor. The transceiver must be able to alert the coordinator of the new streams and must receive the command form the same.

Another pipe that inputs the commands (control inputs) was created. A select system call to alternate between the data and control inputs was used. The transceiver formats the alerts reaching the coordinator in a generic format. The coordinator, depending on the rule-base decides whether to initiate a detector or not. If it is interested

to create a new detector, it issues a command to start the same with the necessary information. The action parameters provide a means to express the same.

As soon as a detector is created, it reports to the coordinator with a MonitorInfo packet. Since they execute as threads in the coordinator, they merely report the contact information of the network-monitor (transceiver).

It is possible to create specific attack detectors like the SYN-Flood detector (which monitors the aggregate traffic and keeps the count of effective SYN and ACK and detects the SYN flood) or port scan attack (which monitors the number of port connections attempted per machine - the number of service based detectors for a machine), tear-drop attack (which monitors the fragmented packets), smurf attack (which monitors the aggregate and checks whether the number of ICMP echo replies exceeds a threshold), etc. The sensor module was written in C. The transceiver, the coordinator and the response agent were written in PERL.

Pitfalls

The packets are analyzed in real-time and the timing is the issue - the communication and analysis delay. Further, UDP, an unreliable transport mechanism, is used and the first packet information sent to the coordinator may be lost and with it the information of a stream is itself lost. Since no negative acknowledgement for preventing the creation of a stream is used, reliance on the one-way communication from the transceiver to coordinator is made. This was not a problem in the previous work wherein the sensor, coordinator and the response agents were bundled as threads in the same process. Thus the inter-process communication (IPC) delay is a challenge to be faced with the real-time, network monitor.

Integration of the Secondary Level Detectors

Profile-builder detector

In order to define the profile of a system, it is necessary to define the normal value ranges of specific parameters that govern the system. Some of these parameters are the CPU usage, disk space utilization, I/O usage, etc. We propose to use statistical measures to define the profile of the system. Some of the statistical measures like mean, trimmed mean, median, mode, minimum, maximum, predicted value, etc of the parameters are used to define the profile of the system.

The profile may be defined with the following parameters, say,

`CPUusage (mean=45 standard deviation=5.00 mode=60), disk space(mean=70 standard deviation=4 mode=80), MemFree(mean=45, standard deviation=10 mode=55)`.

It must be noted that the parameters are collected over a period of time (window) and the statistical measures are determined. It is also possible that each parameter's window might be different.

Thus for a parameter, the profile building detector inputs the window size as an argument. Further, it might be necessary to use a sliding window or a fixed window and report the current range of values. Thus a generic profile builder, which inputs the variable whose statistical measure, is to be determined is needed.

A window to collect the values of parameters is created and this is used to compute the statistical measures. A profile-builder detector for a parameter is defined as `Profile (site, variable, window).` Here, the `(site, variable)` pair defines the unique identifier for the parameter. If window is not initialized, a default value is chosen. In a generic profile detector, it must be possible to define unique

identifier at will. For the performance-monitoring detector, since we are aware that the `(site, variable)` pair compose the unique key, we hard-code it in order to save the parsing effort. Apart from defining the profile of the system, the detector is helpful in setting thresholds of the system variables.

Setting thresholds

The profile detector plays a major role in setting thresholds of the system parameters. Though it is possible for the system administrator, with his/her experience, to configure the coordinator by setting thresholds for certain system variables, it might be unfair to expect the administrator to have the knowledge of all the machines in the domain. With the rule that an incorrect configuration may result in unexpected or unwanted responses, it is necessary to be careful while setting thresholds. Further, taking into account of the number of system variables (over 50 for hostmon detector alone), it would be better for the system to automatically learn the normal value ranges and suggest to the administrator or set thresholds automatically. The profile-builder helps to achieve the same. The thresholds may also vary depending on the time (e.g., thresholds for working hours may differ from night hours), day (e.g., between weekdays and weekends), etc. The profile of the system parameter must maintain these differences that can be applied at the suitable circumstances.

Differential detector

The differential detector, illustrated in Fig. 4.1 is used to identify the slope of the value of the parameter. The slope is calculated with the following formula.

`slope = (current value – previous value) / poll interval`, where poll interval signifies the time interval between the recording of the current and

previous value. It may also be considered any unit time (e.g., every 5 sec, poll interval=1). Hence it is necessary to retain the previous value of a parameter. A thread waits on the current value of the parameter; computes the difference with the previous stored value continuously. Thus it maintains a sliding window of the differences in the values of the parameter recorded at specific intervals.

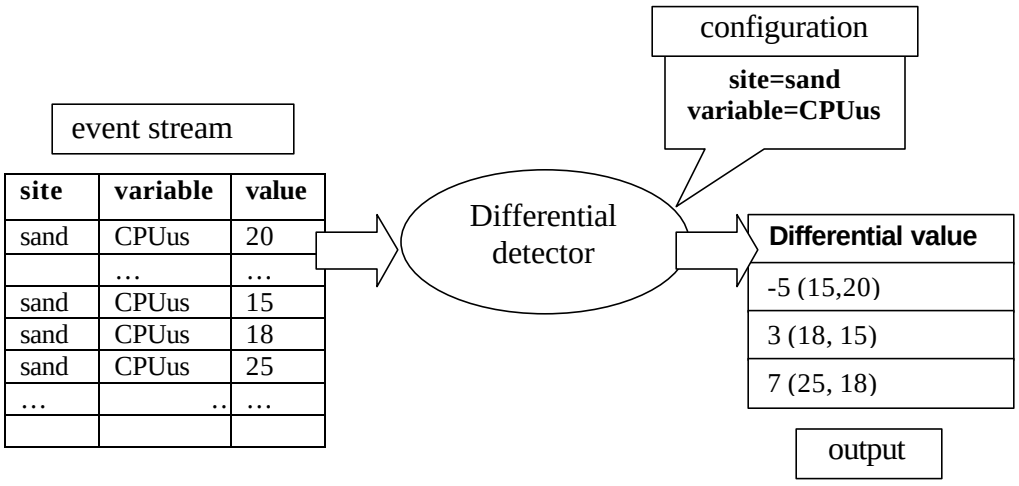

Figure 4.1 Differential detector

This could also be considered as a special profile-building detector with a minimal window size (1), since only the previous value is used. It might be necessary to compute the difference between the past value (nth previous value) and the current value in which case the window size is n. A differential detector is defined as `Differential (site, variable)`

Threshold detector

The threshold detector (illustrated in Fig. 4.2) is configured with x thresholds (in our case, we have considered three thresholds). Thus the detector inputs three thresholds

and a value. Depending on where the value falls in the threshold range, a severity state is assigned. Four severity states, viz., normal, warning, error and critical are used, but the important parameter that is going to determine the effectiveness of the detectors is the thresholds. Using statistical measures derived from the profile-builder may set the thresholds.

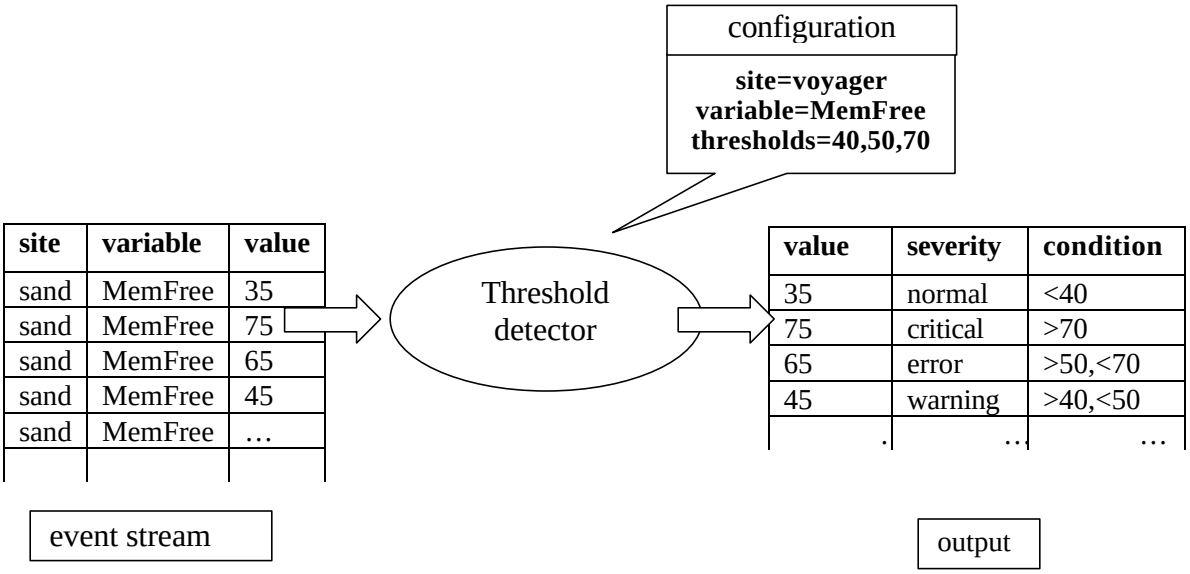

Figure 4.2 Threshold detector

Demultiplexer

A demultiplexer may be used to push the information to the necessary spigot. The demultiplexer must recognize a pattern associated with a specific spigot and a communication port for passing the information. The pattern associated with a spigot may be a function of the host, variable name, an argument associated with the variable. In some cases, the value and the time may be a part of this pattern. In the case of the profile-

building detector, the differential detector or the threshold detector, the spigot pattern may be associated with the host and variable name.

Depending on the computational complexity of the detector, the workload is pushed to a detector or is handled by the demultiplexer itself. In the special case of a threshold detector, wherein the value is checked against a specific threshold, the action may be handled by the demultiplexer itself.

In generic sense, any detector is associated with a pattern. This pattern concerns the variables associated with the function of the detector. The pattern checking is usually handled either fully with the demultiplexer or its checking is distributed with the detector. In some cases, the spigot pattern may be simple but the action associated might be complex necessitating creation of a detector. In other cases, the action may also be simple and may be handled by the demultiplexer itself.

The simplicity of a pattern may depend on the number of variables involved and the complexity of each operation in the pattern. In one extreme case, all the pattern matching may be done in the detector and the demultiplexer may allow the aggregate traffic to flow through all the spigots. In the other extreme case, all the pattern matching is done in the demultiplexer, the detector is associated with only the action. Neither of these cases should be employed for all the spigots. An aggregate spigot filter is merely a multiplier of the information that is processed at the detector end. A complete spigot filter that handles the entire pattern matching for the detectors, involves processing delay since the demultiplexer goes through a list of filters for each detector. Hence the processing delay is proportional to the number of detectors. If the frequency of the traffic is high and the number of detectors is more, then the efficiency is reduced. The pattern matching is to

be distributed efficiently between the demultiplexer and the detectors such that the efficiency of the system is increased.

In a generic demultiplexer, the parsing constraint expressed in the event section of the rule (such as specific value) may be expressed as a precondition which is prefixed to the actual condition that is to be evaluated.

Table 4.2 Examples of demultiplexer pattern check and action methods

#	**Detector type**	**Pattern check by demux**	**Action by demux**	**contact**
1	Profile_builder Thread (sand,CPUus)	$host eq sand && $var eq CPUus	$snd=”$val,$time”; $queue{1} ->enqueue{$snd};	$queue{1}
2	Condition thread (sand System detector)	$host eq sand && ($var eq CPUus || $var eq MemFree)	$snd=”$var,$val”; $queue{2}- >enqueue{$snd};	$queue{2}
3	Threshold (sand, CPUidle,50) no detector thread	$host eq sand && $var eq CPUidle && $val > 50	send coordinator alert info	…

As defined in the table 4.2, each event is checked against the pattern-matching list (conditions) and if a match is found, the corresponding action is taken. These are customized action depending on the type of detector (e.g., profile-based or condition detector) and the specific detector instance (e.g., thread queue $queue{1} or $queue{2}). In some cases, a thread is not necessary for the detector, since the action associated may be simple, e.g., send coordinator an alert notice (as in detector 3 in table 4.2).

One way to improve the efficiency is to hard-code the spigot patterns to some level, e.g., in a network-based detector, there are four types of detectors, connection-based, service-based, destination-based and port-based detectors. It is possible to add a spigot pattern for each detector created as in Fig. 4.3.

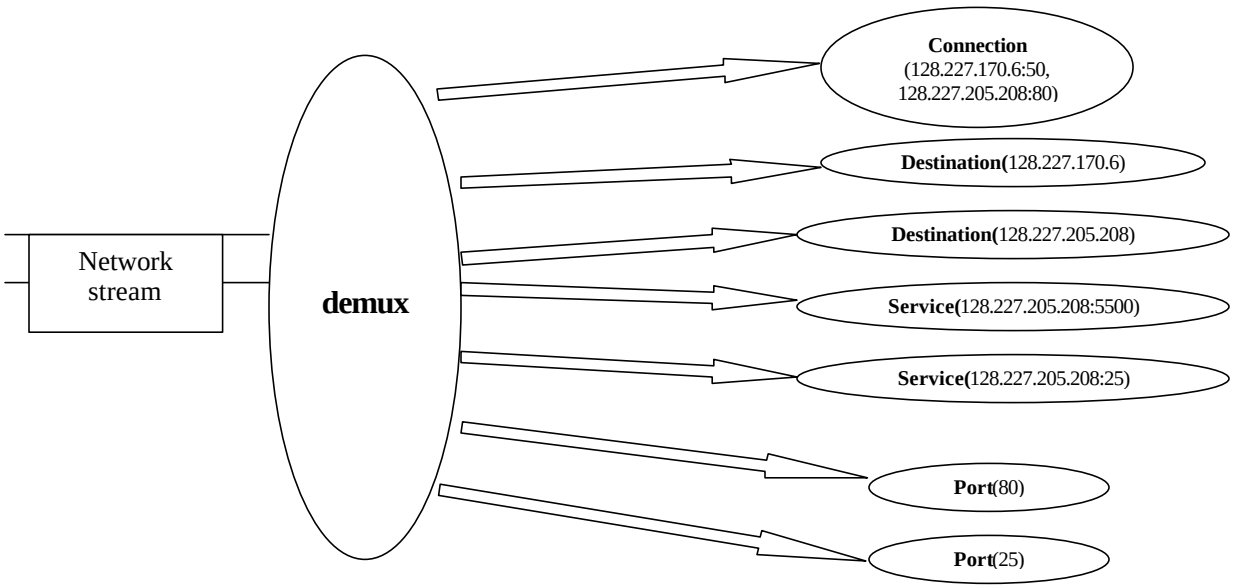

Figure 4.3 Generic demultiplexer

This would mean that the list of patterns to be analyzed for each event is increased. This may be reduced considering the fact that each event may potentially trigger four spigots. Hence it would be sufficient for the demultiplexer to analyze four patterns, associated with each type of the detector that can be only hard-coded. This is important since the traffic of the network stream is usually high.

Table 4.3 Pattern match table for a generic, network-based demultiplexer

#	Detector	Pattern (condition)
1	**Connection** (128.227.170.6:50, 128.227.205.208:80)	source_ip == 128.227.170.6 && source_port == 50 && destination_ip == 128.227.205.228 && destination_port == 80
2	**Destination** **(**128.227.170.6)	destination_ip == 128.227.170.6
3	**Destination** **(**128.227.205.228)	destination_ip==128.227.205.228
4	**Service** **(**128.227.205.208:5500)	destination_ip==128.227.205.228 && destination_port==5500
5	**Service** **(**128.227.205.208:25)	destination_ip==128.227.205.228 && destination_port==25
6	**Port**(80)	destination_port==80
7	**Port**(25)	destination_port==25

Considering a generic demultiplexer and the currently executing detector as in Fig. 4.3, the generic demultiplexer introduces patterns to be checked in its list for each detector as in table 4.3. Each packet is checked through the list of patterns and if the pattern matches, it is fed that information. For example, the packet (tcp, 128.227.170.6, 8900. 127.227.205.228, 25) is matched against the seven patterns and is fed to detectors 3, 4 and 7.

Once the functionality of the detector is satisfied it must cease to exist. The detector may kill itself and report to coordinator or may report its completion that may trigger the coordinator to kill that specific detector.

Thus in a hard-coded system as in Fig 4.3, minimal pattern processing is done at the expense of customizing to the network-based system.

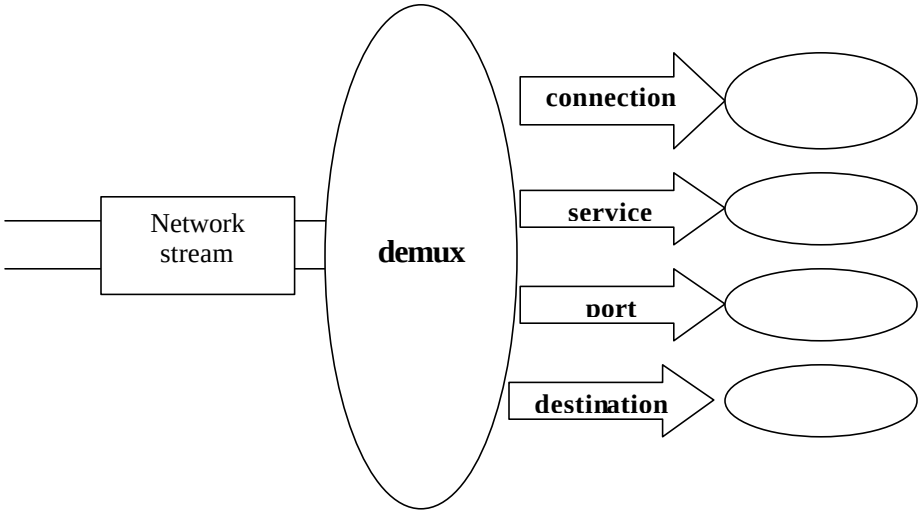

Figure. 4.4 Customized network demultiplexer

Considering the detectors in Fig 4.3, for the customized demultiplexer in Fig 4.4, the pattern checking is limited to four, one for each type of the detector. A mask for each type of detector is applied and checked for the existence of that detector for each packet event. If such a detector exists, the packet information is passed to the detector.

Table 4.4 Customized network demultiplexer pattern matching

Entities\types	**Connection -based**	**Service -based**	**Port-based**	**Destination-based**
Source IP	X			
Source port	X			
Destination IP	X	X		X
Destination port	X	X	X	

For example, the packet (tcp, 128.227.170.6, 8900. 128.227.205.228, 25) is passed to Connection (128.227.170.6,8900,128.227.205.228, 25), destination (128.227.205.228), service (128.227.205.228,25) and port (25) if they exist. Here only four patterns are matched (or selective masking), as shown in table 4.4 (X represents the required field). Except for the connection detector (which does not exist in Fig 4.3), the other detectors exist; hence the packet information is passed to the three detectors. Thus the efficiency of the system is increased.

4.6 Limitations

Some of the limitations of the ID systems are discussed below.

4.6.1 Points of Vulnerability in ID Systems

Every component in the GIDEM may be attacked for different reasons.

The sensors (detectors) act as the ears and eyes of the system as they sense the activity in the system. An attack against this hampers the event generation capability and

blind the system, e.g., an attack against the sniffer prevents the analysis of the raw network traffic.

The analysis engines (detector/transceiver/coordinator) are heavily relied upon to provide security information. The reliability of this is of utmost concern because the whole system is built on its foundation. A complicated analytical technique may provide many avenues for an attack, but an overly simplistic system may fail to detect attackers.

A reliable, stable storage is necessary for recording details of attack. If the attacker compromises this, attackers may alter information after an attack has been detected, eliminating its forensic value. In batch mode, the attack may not be even detected.

The response agents are usually the favorite component to be attacked. Instead of leaving traces of compromising the detectors or analyzing components, it is easier to attack the response agents and masquerading as though response has been taken. Thus the ID systems that rely on the countermeasures are affected.

The ID system continues to take desperate measures to rectify the situation and in some cases, the ID system may be turned against the system, e.g., a response profile might indicate to reboot the system if the system has not responded to an attack after n seconds.

4.6.2 Denial of Service Attacks on and Through the IDS

A basic goal for an attacker is to cause the IDS to fail without losing access to the machines being attacked. The IDS may become a zombie that has an existence but is under the control of an attacker. Denial-of-Service through IDS is severe because it creates the vicious circle of having a need to protect the IDS against Denial of Service. Further, you must now combat your intrusion detection system (now, a possible fifth

columnist) apart from the attacker. The following factors cause denial of service in a system.

1. Resource exhaustion

The attacker identifies some point of processing (say, network processing) that requires allocation of some resource, and causes a condition to consume all of that resource. Resources obviously have limits (e.g., CPU cycles, memory, disk space, network bandwidth, etc) and the effective processing of information is hampered. Note that the ID system may not be killed but will not produce the desired result.

To attack the CPU processing capability, the attack determines the computationally expensive processing operations and forces the IDS to spend all its time doing useless work. For example, forcing the data/event producer of application to fill up the buffer faster with a slower consumer (data analyzer). Critical data might need to be dropped to continue processing.

Any ID system might require memory for its operation, to store the state information or processing events. An attacker can identify which processing operations require ID system to allocate memory and force the IDS to allocate all its memory to store meaningless information. The ID system usually stores the logs of activity on disk. Events/Alerts usually consume space and the system has a finite disk space. An attacker can create a stream of events of less importance that are continually stored and exhaust the disk space and eventually the IDS will not be able to store real events.

Most network-based monitors operate in promiscuous mode, reading all the traffic off the network tap. Resources are consumed for this apart from the network bandwidth. An attacker can overload the network with meaningless information. All these are

effective decoys for an attacker to continue with the real attack by distracting the ID system with such problems.

2. Abusing reactive ID systems

Most ID systems have a response agent to provide a countermeasure for an attack. The IDS may be tricked to launch an unwanted activity, e.g., ID system may have a response capability to reconfigure the firewall to set packet filters in reaction to an attack. Such a system may be tricked with a false positive (through IP spoofing) to react to attacks that have not occurred, thus denying service to a trusted host (completely blocks access for legitimate traffic).

This is like using your fingers to poke your eyes.

3. False positives and false negatives

By tuning the sensitivity of the ID system, it is possible for the system to become a cry-wolf, reporting attacks when nothing happened, or an insensitive system that does not report attacks. Since many analysis engines rely on proper thresholds, tuning to incorrect values may cause adverse affects. Usually false positives may be tolerated to an extent if the false negatives are reduced since we may want to identify all attacks. The effective tuning of the thresholds determines the rate of false positives and negatives in a system.

4. A determined hacker always finds a way to break in or attack the system

This is very much like the problems faced in the medical community. There are some diseases for which vaccines have not been found, and when they are found, a new disease poses a fresh challenge, just like a new virus creating havoc in the Internet community. And sometimes, the vaccine that is to combat the disease may produce side

effects that might be detrimental than the disease and is not be recommended. The hacker may create a virus for which an easily configurable response may be attractive but creating a new problem.

5. It is necessary for the ID system to protect itself from the denial of service attack, the very purpose for which it was created to prevent. It may look like a vicious circle when a process created to prevent denial of service attack must be protected from the same, but is the fact since IDS is also an entity using system resources.

6. Denial of Service inherently is easy to launch or detect but **hard to prevent.**

7. Configuration issues

A highly responsive system that issues drastic responses for attacks targets innocent victims in the event of a false positive. On the other hand, mild responses do little to mitigate the effect of an attack. Hence it is necessary to issue meaningful responses of the required severity level.

4.7 Summary

The rule-based language that can express the policy, namely the monitor and response profiles of the detectors and response agents respectively, was formulated. The coordinator that enforces these policies and triggers an intelligent response agent was also implemented. The response agents can issue time-based and target-specific responses. The detectors and response agents are target-specific and this must also be handled. Integration of third-party detectors with slight modifications was also done with ease. Though ID Systems are helpful in detecting and responding to Denial of Service attacks, they have their limitations, the primary one being that Denial of Services can occur on and through ID systems. The related issues must be carefully analyzed. The effectiveness

of the architecture (coordinator) to deploy agents and issue policy-based responses is discussed in the next chapter.

CHAPTER 5
TESTING

The objectives of testing are to test

1) the validity of the architecture,

2) the policy expression and enforcement through the language,

3) the ease of integration of detectors and response agents,

4) the dynamic creation and termination of detectors.

By testing the validity of the architecture, we mean to test the communication channels, the information flow (event, alert, notification). For policy expression, we can analyze well-known attacks and analyze how to express them in the configuration file for the coordinator.

In order to test, it is necessary to prepare a test bed and identify the ideal machines suitable for the various components. It is also necessary to observe the resource consumption of the components. The resource consumption of the real-time (network) detectors and the polling detectors are of interest. The timing issues, related to IPCs and response time relate to the real-time deployment of the system.

It is necessary to develop some Denial-of-Service attacks. This must include host and network based attacks. Since the host based attacks are more concerned with resource exhaustion, the CPU and memory (physical and virtual) are potential targets. When it comes to CPU, the zombie attack, the fork attack, excessive system hogging is targeted for process table exhaustion. With physical memory (in UNIX), inode exhaustion or disk

space attacks are common. With virtual memory, large and unnecessary 'malloc()' allocation to steal system resources may cause denial of service.

For network-based attack, the IP spoofing is the trickiest, but trivial attack. This is helpful for the attacker to remove the traces (fingerprints) of the source. This is hard to detect and we may end up punishing an innocent victim.

A simple intrusion, its detection and response is also to be tested. For example, a security breach such as deletion of the password file. The ideal response would be to restore the password file. The configuration, detection and response must be tested for the same.

The existing network in the computer security lab in University of Florida is utilized as a test bed for the developed system. The setup involves running the detectors, transceiver, coordinator and response agents. The network-monitor was tested in an x386 with FreeBSD 2.2.2 (voyager.cise.ufl.edu - 128.227.170.6) since it was the gateway for the public CISE domain (as cowboy0.cise.ufl.edu). The other detectors were tested in FreeBSD 2.2.2 and SunSparc 5.6 (in the public CISE domain) and RedHat Linux 6.2 (ripley.dcs.cise.ufl.edu and bates.cise.ufl.edu). The coordinator was tested in both sand.cise.ufl.edu and voyager.cise.ufl.edu. The coordinator in voyager.cise.ufl.edu was exclusively configured and tested for the network monitor running in the same machine. The voyager is a less powerful machine compared to ripley and bates. Apart from testing the correctness and validity of the architecture, the effective policy-enforcement should also be checked.

The CPU usage and disk usage while executing a real-time, network-monitor, transceiver, coordinator and response agent are illustrated by the graph (Fig 5.1, 5.2, 5.3).

Sample attack scripts were used to test the effectiveness the architecture. The integration of some third party detectors with the existing system was also implemented and tested.

5.1 Performance Measurement of Network Detector

The performance of the network detector (which includes the collection, demultiplexer, coordinator and response agent) is expressed through the CPU user mode (Fig 5.1), CPU system mode (Fig 5.2), and CPU interrupt mode (Fig 5.3) utilization averages (%).

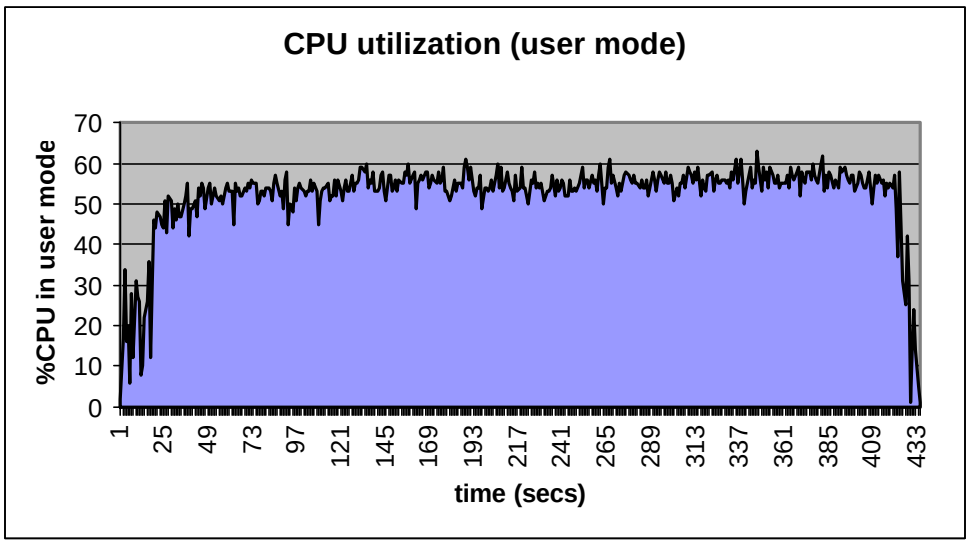

Figure 5.1 CPU utilization (user mode) for executing network monitor and coordinator

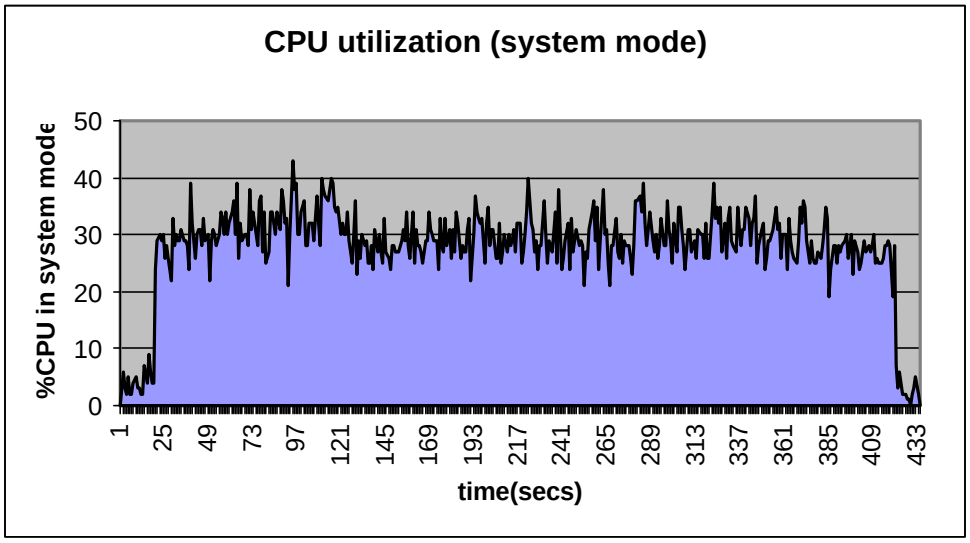

Figure 5.2 CPU utilization (system mode) for executing network monitor and coordinator

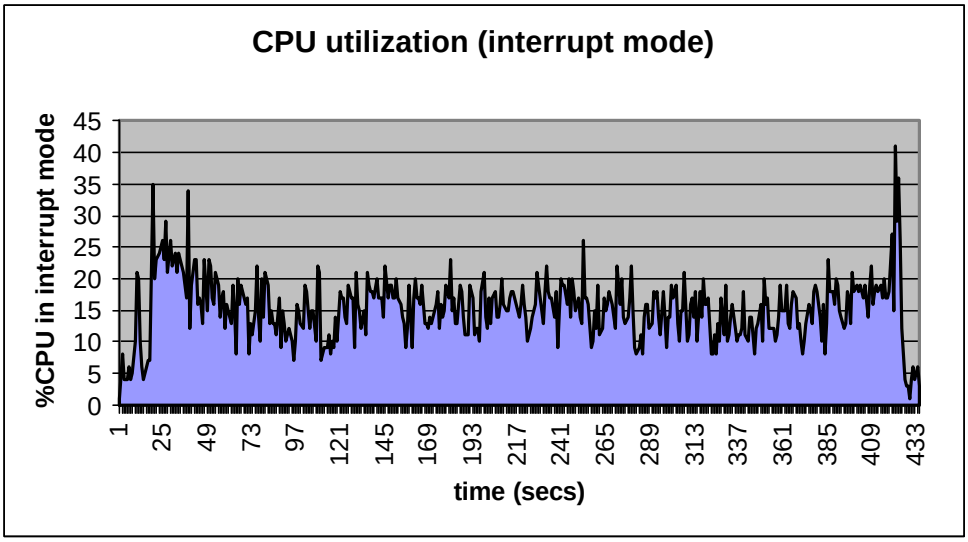

Figure 5.3 CPU utilization (interrupt mode) for executing network monitor and coordinator

The network detector was executed in voyager. It is evidenced that the network detector is a resource-hungry detector (about 60% CPU usage) and must be configured and executed with care. In this case, the network detector, coordinator and response agent

are separate processes. Further, the collection module was coded in C and the rest was coded in PERL.

Since the network detector has high resource consumption, it is advisable to deploy them in dedicated servers. In order to construct the global network traffic, a single network detector or a set of network detectors distributed over the right locations must be deployed. The system that executes the network detectors must have enough resources to match the network traffic.

5.2 Performance Measurement of Polling Detectors

Most of the performance-measuring detectors are polling detectors. At regular intervals, they execute queries and record the values. The polling interval is a potential factor affecting the performance and efficiency. Hostmon is a performance-measuring detector that records the system variable values at regular intervals. It is possible to configure the polling interval of hostmon. The hostmon was executed with different polling intervals (one sec, ten secs, one minute, two minutes, five minutes) and its CPU usage (user mode, system mode and idle) was measured. The tests were performed in voyager and ripley and their measurements are shown in the graphs (Figures 5.4, 5.5, 5.6)

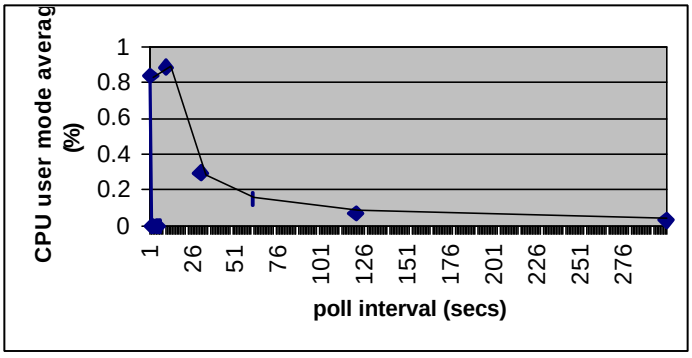

Figure 5.4 CPU user mode usage average for hostmon in ripley

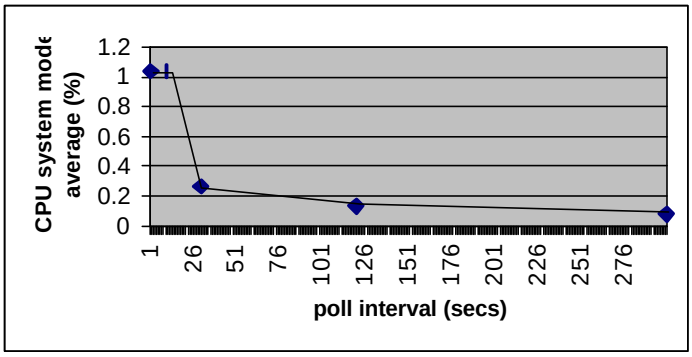

Figure 5.5 CPU system usage for hostmon in ripley

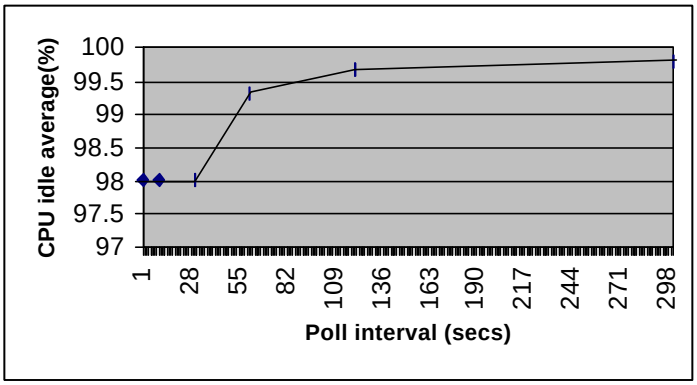

Figure 5.6 CPU idle average for hostmon in ripley

It is encouraging to note the CPU usage is sparse in ripley for running hostmon. The detector requires less than average of 2% CPU usage. Hostmon is coded in PERL and it merely executes a series of UNIX commands and extracts the system variable values. The CPU usage for hostmon in voyager (results in Figures 5.7, 5.8, and 5.9), which is comparatively a resource deficient machine than ripley is also less. It takes less than an average of 2% for running hostmon even in a panic mode. The hostmon can be executed in a distributed manner (in every host) since it does not consume much resource.

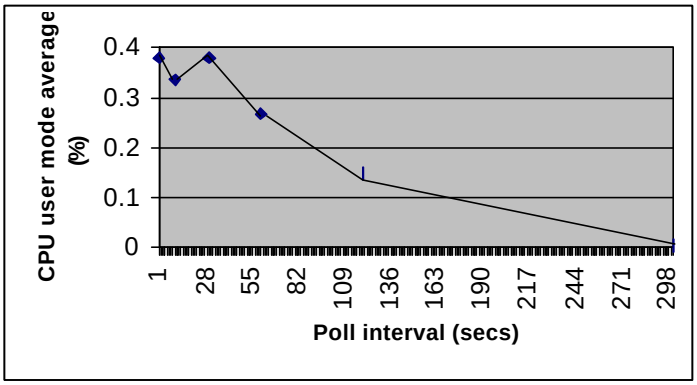

Figure 5.7 CPU user mode average for hostmon in voyager

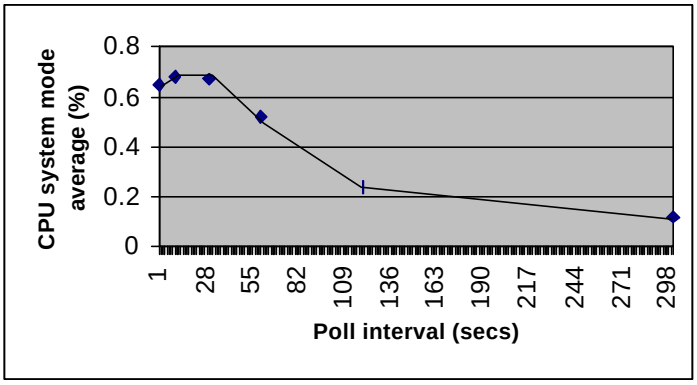

Figure 5.8 CPU system mode average for hostmon in voyager

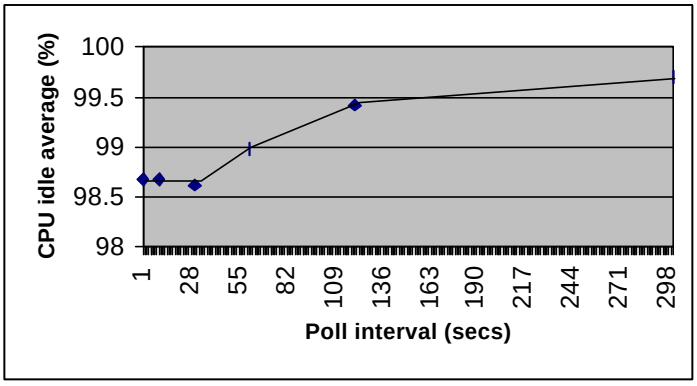

Figure 5.9 CPU idle average for hostmon in voyager

Thus reconfiguration of hostmon to switch between the normal and panic mode might not cause a radical difference in the usage of system resources. Though hostmon is just a representative of detectors that consume minimum resources, it is possible to identify other detectors by measuring their resource (CPU) requirements. These detectors may be classified as “safe”. When needed, the polling interval may be tuned to run at the required level. The unsafe detectors must be configured carefully. Otherwise, they may become potential attacks on the system itself (suicide).

Portmon is a polling detector that detects the working of the various ports in a system. It checks a set of ports (services) at regular intervals. Further it is also a centralized detector that is capable of checking the services in multiple machines.

The performance measurements for executing portmon in ripley and voyager are illustrated in tables 5.1 and 5.2 respectively. These measurements reflect a panic mode portmon execution. Portmon may be considered a safe detector.

Table 5.1 CPU usage for portmon detector in ripley

CPU	**Average(%)**	**Min(%)**	**Max(%)**	**Variance(%)**
User mode	0.28	0	10	1.29
System mode	0.12	0	2	0.162
Idle	98.67	82	100	2.4933

Table 5.2 CPU usage for portmon detector in voyager

CPU	Average (%)	Min (%)	Max(%)	Variance (%)
User mode	0.19	0	11	1.59
System mode	0.19	0	6	0.72
Idle	99.5	83	100	4.40

The file integrity checker detector, fcheck is a polling detector. The daemon initializes a database of the file information and then polls every x seconds and checks whether the file has changed. The CPU usage for executing fcheck in bates for different poll intervals (one second, two seconds, ten seconds, one minute, two minutes and five minutes) were measured and illustrated in Figures 5.10, 5.11, and 5.12.

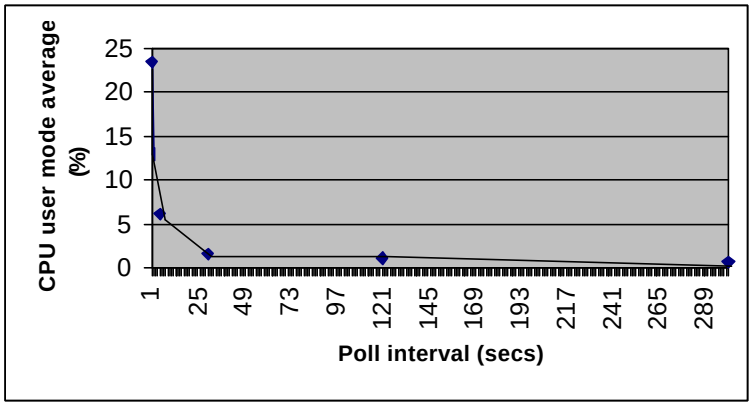

Figure 5.10 CPU user mode average for fcheck in bates

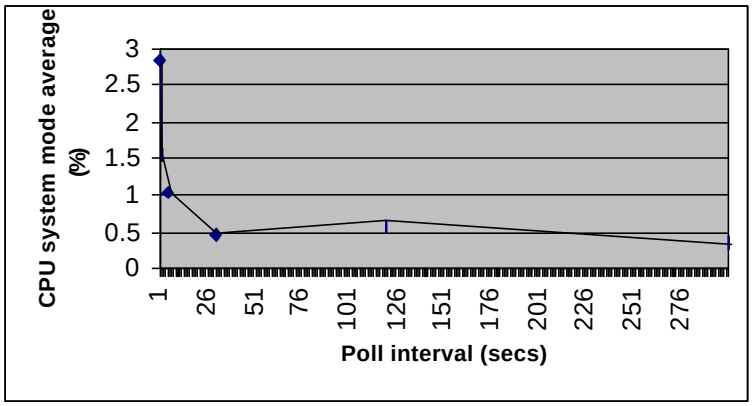

Figure 5.11 CPU system mode average for fcheck in bates

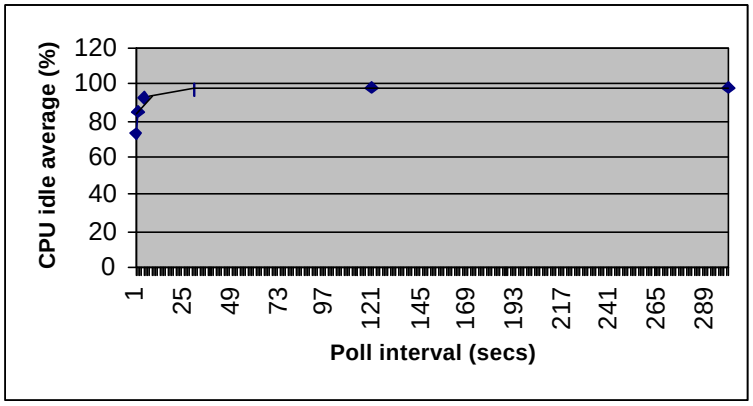

Figure 5.12 CPU idle average for fcheck in bates

Fcheck when executed at less than a minute poll interval might be detrimental to the system. When executed at one-second poll interval, it consumes about 25% of the CPU. It is to be noted that if complex message digests were used (like the md5), the CPU usage would have been more. Hence, the fcheck must be used cautiously.

5.3 Performance Measurement of Transceiver

The transceiver is responsible for executing the secondary level detectors on the demand basis. For the host-based IDS, a transceiver with a set of secondary level

detectors was tested in sand. The secondary level detectors included the profile-building detectors, differential detectors and threshold detectors types. Each detector is executed as a thread (spigot) in the transceiver. The demultiplexer evaluates the constraint associated with the spigot and pushes the data to the spigot if the constraint is satisfied. A generic demultiplexer that has no prior knowledge of the event parameters was constructed and tested. The parsing times associated with processing 52 events are tabled in 5.3. Each detector is associated with a condition and as each detector is introduced in the transceiver, an additional condition is added.

Table 5.3 Generic demultiplexer processing time

# of conditions	**Parsing time (average) (seconds)**	**Parsing time (max) (seconds)**
1	0.0007	0.002
2	0.001245	0.0024
4	0.002335	0.00353
8	0.004791	0.007474
16	0.008684	0.014
32	0.03	0.118

For customized demultiplexer in the transceiver, which is aware of the unique patterns, it classifies the detectors depending on the patterns. The timings of these detectors are shown in table 5.4.

Table 5.4 Customized demultiplexer processing time

Number of detectors	**Average parsing time (seconds)**	**Max. parsing time (seconds)**
1	0.000143	0.001985
2	0.000172	0.001945
4	0.000231	0.001922
8	0.000341	0.004533
16	0.000609	0.003404
32	0.000838	0.005424

The customized demultiplexer is roughly five times faster than a generic demultiplexer. Hence it is advisable to generate customized transceievers and demultiplexers that can be deployed dynamically by the coordinator.

5. 4 Timing Issues

For the dynamic creation and termination of detectors, especially secondary level detectors in transceivers and network-based detectors, timing is an important issue. The communication delays are critical depending on the detector. A real-time, network-based detector expects faster response than a polling detector. For the transceiver, communication delays using Perl Thread queues among demultiplexer and the detectors was found to be about 0.00252 seconds. A communication delay of this order is usually unacceptable for real-time network monitors. The delays may also be related to the thread context switching. For polling host-based detectors, this approach may be taken, but it is not advisable for network monitors.

A typical detection and response scenario was tested and timed. The file integrity checking detector fcheck was used to detect any deletion of files. If deleted, they were automatically restored. The attack is the deletion of the password file. The attack is first detected by the fcheck, which is a polling detector executing every minute. It reports the file deletion event to the coordinator. The coordinator processes the event and issues the necessary action to be handled to the response agent. The response agent on the recipient of the command takes the necessary action to restore the file. The timing for the sequence of events is illustrated below.

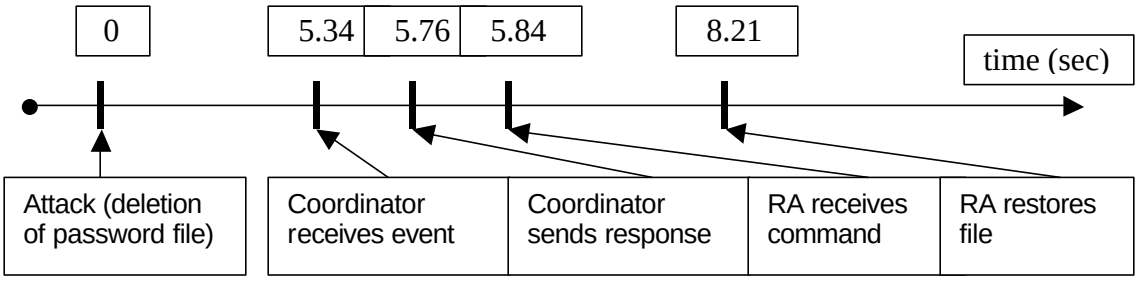

Figure 5.13 A typical attack and response timing

It is to be noted that the response agent may spawn a new process to realize the action. This is observed in Fig 5.13 between the time when the response agent (RA) receives the command and the time when the file is restored. The processing of command for the target and execution of the same approximately takes 2.5 seconds. Such delays might not be expected if the response was spawned as a thread. Further the response agent creates the grandchild that executes the command in order to avoid zombies (suggested by Desai [27]). This might increase the delay too.

Thus the validation of the architecture, its effectiveness, the flexibility of configuring the detectors and response agents through the language for policy enforcement were tested.

5.5 Summary

The system components (detectors, coordinator, transceiver and response agents) were tested in FreeBSD, RedHat Linux and SunSparc systems. Even though PERL has a rich regular expression feature (used for parsing) and eval (used for condition evaluation), it is an interpreted language and may strike hard in the performance issues. In real-time network detectors where response time is a critical factor, the thread context switching and communication delays (IPCs) are unacceptable, even though it proves to be efficient for polling detectors. A customized demultiplexer capable of classifying conditions based on specific patterns executes at least five times faster than a pure generic demultiplexer. Hence, a customized transceiver must be used when possible. The dynamic creation, termination and reconfiguration of secondary level detectors are an efficient means to detect and respond to attacks. Further, the language is also able to express succinctly the monitoring and response profile.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

A conclusion is simply the place where you got tired of thinking. Probably true. To escape from such an accusation, suggestions for future work has been made to make this thought a continued process. The architecture that has been evolved with an objective to deploy agents on the "as needed" basis at the required computational level has proven to be effective. The strands of this model are also observable in generic environments like EMERALD, AAFID, etc. The architecture has been implemented considering the IETF model requirements (though some features need to be implemented). The language to represent the policy is able to express the monitoring and response profiles, keeping target behavior as a consideration. It is extensible to express events, alerts, notification and responses. Formal language based techniques have been used to develop the rule-based coordinator. The coordinator controls the creation, termination, and reconfiguration of the secondary level detectors that may communicate through the transceiver. The secondary level detectors may opt to distribute the parsing effort with the demultiplexer depending on the resource level of the system. The integration of the detectors, analyzers and response agents can also be done smoothly through the language with the necessary adaptors. Thus a generic intrusion detection model that is able to integrate software

entities (analyzers, detectors and response agents) realizes the isolated as a whole and is effective.

6.2 Future Work

It is essential to map meaningful responses to various well-defined attacks. The strength of the architecture also lies in a proper and effective configuration. Though the language is flexible, it must not be degraded to represent simple and ineffective mapping. The parameter passing mechanism must be effectively used to bring out the glory of the model. A generic Native format could be introduced to represent the information sensed by the primary detectors. Though the event format currently used, serves the purpose, an effective format complying with the IETF format could be devised. Some of the IETF message and communication mechanism could be introduced or enhanced in the current architecture. Reliable transmission and mutual authentication are some key areas of interest to enhance the security features in the system.

For efficiency, the rule-based coordinator may be multi-threaded, with each specified rule spawned as a thread and is able to set the pattern for the demultiplexer, which filters the events pushed to them. This feature is present in the transceiver now.

REFERENCES

[1] Deborah Russell and G. T. Gangemi, Sr., *Computer Security Basics*, O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

[2] Frederick B. Cohen, *Protection and Security on the Information Superhighway*, John Wiley & Sons, New York, 1995.

[3] J. Howard, "An Analysis of Security Incidents on the Internet," CERT Help page *http://www.cert.org:80/research/JHThesis/Chapter5.html*, 2000.

[4] Whatis.com, *http://www.whatis.com*, whatis.com help page, 2000.

[5] G. Halbig, "An Active Network Approach to Defending and Tracking Denial of Service Attacks," *Master of Science Thesis*, Computer and Information Sciences and Engineering, University of Florida, Gainesville, 1998.

[6] Lawrence R. Halme and R. Kenneth Bauer, "AINT Misbehaving: A Taxonomy of Anti-Intrusion Techniques," SANS Institute resources, *http://www.sans.org/newlook/resources/IDFAQ/aint.htm*, June 2000.

[7] G.H. Kim and E.H. Spafford, "Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection," *Technical Report CSD-TR-94-012*, Purdue University, West Lafayette, IN, Feb 21, 1994.

[8] R. A. Kemmerer, "NSTAT: A Model-based Real-time Network Intrusion Detection System," *Technical Report TRCS97-18*, Computer Science Dept., University of California Santa Barbara, November 1997.

[9] S Forrest, S Hofemyr, A Somayaji, T.A. Longstaff, "A Sense of Self for Unix Processes," *Proceedings of 1996 IEEE Symposium on Computer Security and Privacy*, Oakland, CA, May 6-8, 1996.

[10] D. E. Denning, "An Intrusion Detection Model," *IEEE Transactions in Software Engineering*, vol SE-13, pp. 222-232, Feb 1997.

[11] NOCOL- -Design and Internals, "NOCOL SNIPS Network Monitoring Management," Netplex Technologies Inc., *http://www.netplex-tech.com/software/nocol/*, June 2000.

[12] D. Anderson, T. F. Lunt, H. Javitz, A. Tamaru, and A. Valdes, "Detecting Unusual Program Behavior Using the Stastistical Component of the Next-generation Intrusion Detection Expert System (NIDES)," *SRI-CSL-95-06*, SRI International, Menlo Park, CA, May 1995.

[13] A. Porras, and P. G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," *Proceedings of the Nineteenth National Computer Systems Security Conference*, pages 353-365, Baltimore, Maryland, 22-25 October 1997.NIST/NCSC.

[14] Ulf Lindqvist and Phillip A. Porras, "Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)," *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, Oakland, CA, May 9-12, 1999.

[15] L. Me, "Genetic Algorithms, a Biologically Inspired Approach for Security Audit Trails Analysis," *short paper presented at the 1996 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1996.

[16] G. Vigna, S. T. Eckmann, and R. A. Kemmerer, "The STAT Tool Suite," *Proceedings of DISCEX 2000*, Hilton Head Island, January 2000, IEEE Press, New York 2000.

[17] I. Koral, "USTAT: A Real-time Intrusion Detection System for UNIX," *Technical Report TRCS93-26*, Computer Science Department, University of California, Santa Barbara, CA, 1993.

[18] G. Vigna and R. Kemmerer, "NetSTAT: A Network-Based Intrusion Detection Approach," *Proceedings of the 14th Annual Computer Security Applications Conference*, Scottsdale, Arizona, December 1998.

[19] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle, "The Design of GrIDS: A Graph-Based Intrusion Detection System," *Technical Report*, Computer Science Department, University of California, Davis, CA, January 1999.

[20] IETF (Internet Engineering Task Force) model, Internet Security Systems, Intrusion Detection Message Exchange Requirements, *http://www.ietf.org/html.charters/idwg-charter.html*, 2000.

[21] COAST, Computer Operations Audit and Security Technology laboratory, Computer Science Department, Purdue University, West Lafayette, IN, *http://www.cs.purdue.edu/coast/coast.html*, 2000.

[22] WHITEHATS Max vision network security penetration testing, *http://www.whitehats.com*,2000.

[23] Mark Crosbie and Eugene Spafford, “Active Defense of a Computer System using Autonomous Agents,” *CSD-TR-95-008,* Department of Computer Sciences, Purdue University, West Lafayette, IN, 1995.

[24] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, P. G. Neumann, H. S. Javitz, A. Valdes, and T. D. Garvey, “A Real Time Intrusion Detection Expert System (IDES)” *Final Report*, SRI International, Menlo Park, CA, Feb. 1992.

[25] Wenkee Lee and Salvatore J. Stolfo, “Data Mining Approaches for Intrusion Detection”, Computer Science Department, Columbia University, New York, NY, *http://www.cs.columbia.edu/~sal/hpapers/USENIX/usenix.html*, 2000.

[26] PERL home page, *http://www.perl.com*, 2000.

[27] P. Desai, "A Performance Monitoring System for Network Hosts to Monitor Denial of Service Attacks," *Master of Science Thesis*, Computer and Information Sciences and Engineering, University of Florida, Gainesville, 1998.

[28] T. Keski, "Network Signature Based Detection of Denial-of-Service Attacks," *Master of Science Thesis*, Computer and Information Sciences and Engineering, University of Florida, Gainesville, 1997.

[29] S. Yerramreddy, "Design and Implementation of an Agent-Based Anomaly Intrusion Detection System," *Master of Science Thesis*, Computer and Information Sciences and Engineering, University of Florida, Gainesville, 1998.

[30] W. Zhang, “Event-Based and Agent-Based Intrusion Detection System,” *Master of Science Thesis*, Computer and Information Sciences and Engineering, University of Florida, Gainesville, 1999.

[31] Intrusion Detection and Security Software, Fcheck, Intrusion Detection and Policy Enforcement or Auditing Software for Unix & Windows NT/9x/3.x Platforms, *http://sites.netscape.net/fcheck/*, 2000.

[32] Firewall Tool kit FWTK 2.1, Trusted Information Systems (TIS), *http://www.fwtk.org/fwtk/*, 2000.

BIOGRAPHICAL SKETCH

Seetharaman Balasubramanian was born in Chennai, T.N., India. He received the Bachelor of Engineering degree in computer and information sciences and engineering from PSG College of Technology, Coimbatore, India, in June 1998. He will receive his Master of Science degree in computer engineering from the University of Florida, Gainesville, in August 2000. His research interests include computer network and security.