DESIGN AND IMPLEMENTATION OF A HOST-BASED AND EVENT-BASED DETECTOR

By

JIN CHEN

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

2001

Dedicated to my Mom and Dad

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. Richard Newman for his constant encouragement and the opportunity to work on this research project. His innovative ideas and encouragement have made this work interesting and challenging.

I would also like to thank Dr. Randy Chow and Dr. Doug Dankel for agreeing to serve on my committee.

Many thanks go also to Amit Gandre and Matt Wilson for their support and great friendship. My very special thanks go to Yuan Li for being a constant source of encouragement and support throughout.

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

I sincerely thank all the people who have helped and supported me directly and indirectly in the course of this thesis work.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

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

DESIGN AND IMPLEMENTATION OF A HOST-BASED AND EVENT-BASED DETECTOR

By

Jin Chen

August 2001

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

In the recent decade, network system and e-commerce models can go to every "corner" of our world. While people enjoy such unprecedented convenience and happiness, hackers unceasingly "crack" the security of the network. In particular, Denial of Service (DoS) attacks have become a very concerning issue of network security. Research on network intrusion detection systems has been widely undertaken, including the development of several feasible systems such as AAFID, EMERALD and GrIDS.

This thesis seeks to design and implement a host-based and event-based detector, which is a part of the intrusion detection system in CONS lab. The designed detector can provide an upper level controller, such as a coordinator, with the capacity to start, stop or reconfigure the system. The detector conducts a fast detection and response based on its tree-like data buffer. An aging policy is applied to increase the durable detection capacity and usage time of the fast detection data buffer. Besides detecting the failure of login events by the host, the analysis and detection of successful login events draws the

designed detector's interest. A combination of two detection results can enhance the intrusion detection possibilities and accuracy. A compressed log file created and maintained by the detector makes it possible to recover a detection system and launch a post-event polling analysis, if needed. An object-oriented language makes the detector more generic and scalable for further development.

CHAPTER 1
INTRODUCTION

While the world stepped into the new millennium without Y2K complaints, people started enjoying a fast growth of network and E-business glory. However, in early February, 2000, a handful of internet sites operated by such respected Internet electronic commerce companies as Yahoo, CNN.com, eBay, E-Trade, Buy.com, Amazon.com and ZDNET.com were shut down one by one for several hours. The attacks, which are so-called denial-of-service (DoS) attacks, may have directly cost these organizations tens of millions of dollars in lost sales. The damage to reputations and credibility may be more difficult to gauge [1]. Consequentially, a big question mark always keeps returning to people: *"Is the web secure?"*

1.1 Denial of Service Attacks

The Internet is rapidly evolving as the "Information Super Highway" as academic, business and government organizations as well as individual users deploy a proliferation of local area networks (LANs) and wide area networks (WANs). However, this increasing popularity of the Internet comes with inherent security risks [2]. Ideally, every computer system should be completely secure. Unfortunately, it is not possible in today's environment. The 1998 Computer Crime and Security Survey conducted by the Computer Security Institute (CSI) and Federal Bureau of Investigation (FBI) reports a continuing growth in U.S. computer incidents [3]. Intrusion activities seek to compromise integrity, security, confidentiality and availability of a computer system,

including illegally accessing internal information, breaking into computers externally, inserting erroneous information into files and flooding the network thereby reducing its effective channel capacity.

A major well-known type of attack is the denial-of-service (DoS) attack, which is characterized by an explicit attempt to prevent legitimate users of a service from using that service. In a typical connection, the user sends a message asking the server to authenticate it. The server returns the authentication approval to the user. The user acknowledges this approval and then is allowed to use the services of the server. In a denial of service attack, the attacker sends several authentication requests to the server, filling it up. All requests have false return addresses, so the server cannot find the user when it tries to send the authentication approval. The server waits, sometimes more than one minute, before closing the connection. When it does close the connection, the attacker sends a new batch of forged requests, and the process begins again--tying up the service indefinitely [4]. Therefore, the server cannot provide the service to the one who truly needs it.

Denial of service attacks come in a variety of forms and aim at a variety of services. There are three basic types of attack: consumption of scarce, limited or non-renewable resources; destruction or alteration of configuration information; physical destruction or alteration of network components [5].

Distributed DoS (DDoS) attacks are a new variation on the theme of denial of service, and they pose a serious threat to any Internet-based enterprise, regardless of infrastructure redundancy or robustness. The first DDoS attack in the wild was reported in the middle of 1999. During the week of February 7th through 11th, 2000, it emerged

as a new major category of attack on the Internet. Their danger stems from their simplicity: the tools that are necessary to set up and initiate the attack are easy to obtain and implement, and the victim can experience hours or even days of service interruptions. Therefore, any organization that uses the Internet must take immediate actions. Unlike DoS removing a given resource from service, such as a server or network, without permission, DDoS attacks--in which multiple systems generate the attack on a single target--are the next logical step. Distributing the workload across hundreds or even thousands of systems is one of the best ways to accomplish an intensive DDoS attack. It not only increases the impact, but also makes stopping the attack--much less identifying the attacker's true source--much more difficult [6].

The study of security in computer networks is a rapidly growing area of interest since late 1980s. Controlling access to a computing system is the first choice to reduce the risk of penetration threats. Several intrusion prevention techniques can be applied to enforce network security. *Authentication* can be used to verify the identity of a user. Different *access control* policies limit different classes of users' rights to access to data and resources. *Security policies* are intended to preserve the privacy and integrity of every user. *Firewalls* and *cryptography* can be used to block and avoid illegitimate accesses. Good *software specification, coding and testing policies* can reduce introduction of intrusion holes during development [7]. With all the above techniques, it is still not possible to prevent completely security threats in open data networks. Monitoring and recording activities of users and networks will be necessary.

1.2 Intrusion Detection Systems

Intrusion detection is defined as “the problem of identifying individuals who are using a computer system without authorization and those who have legitimate access to the system but are abusing their privileges. (i.e. the ‘insider threat’)” [8]. An intrusion detection system is a computer system (possibly a combination of software and hardware) that performs intrusion detection. Most intrusion detection systems try to perform their task in real time [9].

Desirable characteristics of an intrusion detection system include the following [9]:

1. *Continuous running*: An Intrusion detection system should run under minimal human supervision.

2. *Fault tolerance*: When the system crashes, the intrusion detection system must have the capability to resume its operation and with no need to rebuild its knowledge base.

3. *Resist subversion*: An intrusion detection system must have the capability to monitor itself and prevent itself from being attacked by intruders.

4. *Minimal overhead*: A well-designed intrusion detection system should ensure a minimal system load.

5. *Configurable*: An intrusion detection system should be configurable. Therefore, it can be deployed on different hosts.

6. *Adaptable*: An intrusion detection system should be able to adapt to changes in both global and local activities.

As the number of systems to be monitored increases and the chances of attacks increase we also consider the following characteristics as desirable [9]:

1. *Scalable*: An intrusion detection system should have the ability to increase the number of hosts it monitors by adding new components without any other modification.

2. *Graceful degradation of service*: If some components of the intrusion detection system stop working for any reason, the rest of them should be

affected as little as possible.

3. *Dynamic reconfiguration*: Changes in intrusion detection configuration should be done independently and invisibly to other components. They should not require restarting the whole intrusion detection system.

Intrusion detection systems usually include three components: sensors, analyzers and a user interface. Sensors are responsible for collecting data. The input of a sensor may be network packets, log files and system call traces. Sensors collect and forward this information to an analyzer. The analyzer receives input from sensors or its peer analyzers. It determines if an intrusion has occurred and generates an alarm when an intrusion occurs. The analyzer may provide guidance about what actions to take as a result of the intrusion. The user interface allows a user to view the output from the system or to control the behavior of the system. A generic requirement is summarized in the IETF (as shown in Figure 1.1) [10].

Intrusion detection systems can be classified by a variety of characteristics. Based on the way their components distributed, the intrusion detection can be distinguished as *centralized* or *distributed* intrusion detection systems. When the data analysis is performed in a fixed number of independent locations and the location of the collection components are not considered, the system is recognized as centralized intrusion detection systems, e.g. IDES, IDIOT, NADIR and NSM. When the data analysis is performed in a number of locations proportional to the number of hosts that are being monitored and only the locations and number of the data analysis components need to be considered, the system is recognized as distributed intrusion detection systems, e.g. DIDS, GrIDS, EMERALD and AAFID [9].

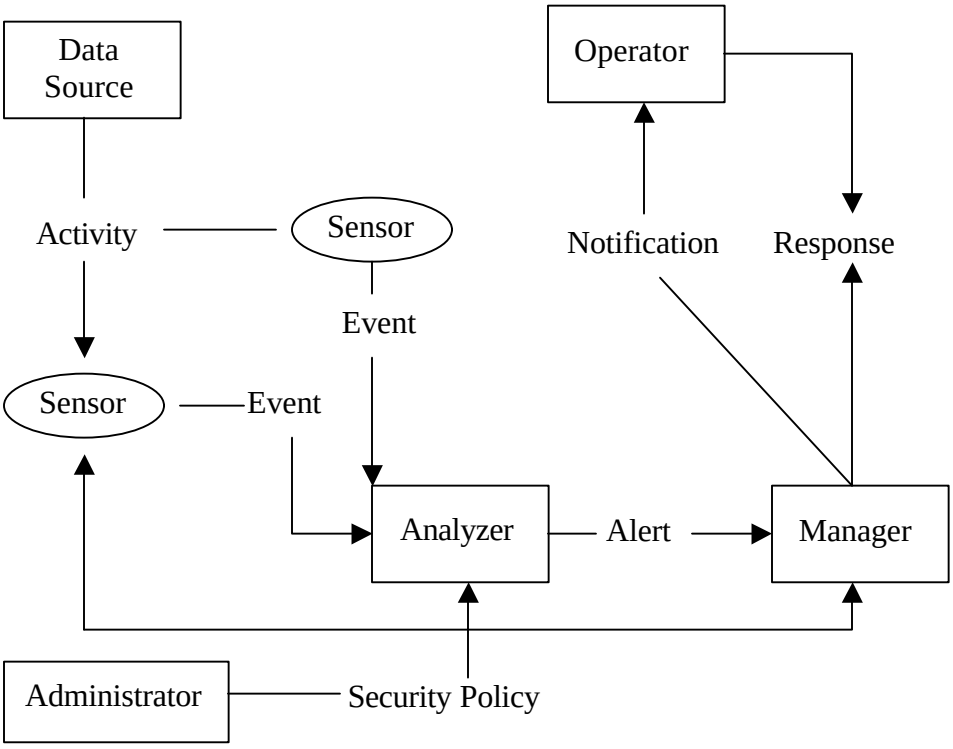

Figure 1.1 The IETF model [10]

The different characteristics of analyzers divide intrusion detection systems into two categories: *knowledge*- and *behavior*-based intrusion detection systems. These are discussed further below. The different response models of the intrusion detection to attacks categorize intrusion detection systems into two models: *passive* and *active* models. The passive system merely generates alarms while the active one takes more energetic actions in response to the attacks. Depending on the audit source location, intrusion detection systems can be classified as *host*- and *network*-based intrusion detection systems. Usage frequency distinguishes among intrusion detection systems based on whether they have *real-time* continuous monitoring capabilities or they *run periodically* [11]. The next two sections introduces knowledge-based versus behavior-based and host-based versus network-based intrusion detection systems in detail.

1.3 Knowledge-based versus Behavior-based Intrusion Detection

Intrusion detection systems have different strategies for the interaction with the input information. There are two trends in intrusion detection: knowledge-based and behavior-based. Knowledge-based intrusion detection is based on accumulation of signatures from specific attacks and system vulnerabilities and is referred to as *misuse detection* or *detection by appearance*. Once it detects any attempt to exploit a known vulnerability, an alarm is triggered. Activities not explicitly recognized as attacks are considered acceptable. Knowledge-based techniques have good detection accuracy because of their low false alarm rates. However, their detection completeness is affected by the difficulty of collecting the necessary information on the known attacks and updating them with new vulnerabilities and environments. Expert systems, signature analysis, *Petri-nets* and state-transition analysis can be used to implement knowledge-based intrusion detection [11, 12].

Behavior-based intrusion detection is implemented by detecting the behavior that deviates from normal or expected behavior of the system or the user. It is also known as *anomaly detection* or *detection by behavior* [11]. The current activities are compared with a reference mode. If any significant deviation is observed, an alarm is generated. Therefore, its good detection completeness can contribute to automatic discovery of new attacks. However, it may cause a relatively high false alarm rate, which reduces its detection accuracy. Implementation of behavior-based intrusion detection includes expert systems, neural networks, user intention identification and computer immunology [11, 13].

1.4 Host-based versus network-based intrusion detection

There are two kinds of intrusion detection models, network-based and host-based, which recognize attacks. In either case, these systems check for attack signatures and specific patterns that usually indicate malicious or suspicious intention.

The primary difference between host-based and network-based intrusion detection is the data source from which the system derives the information used to conclude the occurrence of an attack. Host-based intrusion detection is the first area that was explored in intrusion detection, and it involves loading a piece or pieces of software on each server that needs to be monitored. The loaded software uses log files, process accounting information, user behaviors or output data from application-based intrusion detection systems operating on the host. Network-based intrusion detection monitors the raw network packets on its network segment. It does not require software to be loaded and managed on a variety of hosts [14].

Since host-based intrusion detection involves looking at the logs containing events that have actually occurred, it can measure whether an attack is successful or not with more detail and accuracy than a network-based intrusion detection system. It can monitor all user log activities as well as what each user does during the time that he connects to the network. Host-based intrusion detection may also check the integrity of the system files and watch for suspicious processes, such as file accesses and attempts to install executables or access-privilege service. Host-based intrusion detection systems reside on each host to monitor the activities that occur at the user level, much of which can not be detected by a network-based system. For example, attacks from the keyboard or rlogin within a LAN do not cross the network and, therefore, cannot be detected by network-based intrusion detection. In switched environments, it is difficult to achieve

sufficient network coverage by network-based intrusion detection systems. Host-based intrusion detection provides better performance due to residing on as many critical hosts as need. Furthermore, encrypted network traffic can be hardly handled by a network-based system. When the incoming traffic comes to an operating system, it has been de-encrypted. A host-based system is able to monitor such events easily. Finally, a host-based intrusion detection system does not need any additional hardware and is cost-effective. These are the reasons why host-based intrusion detection system has been deployed first [14, 15].

Network-based intrusion detection systems operate at the network level without regard to the type of operating system. They monitor all packet headers and can investigate payload contents that may slip by a host-based system. In addition, a network-based system examines live packets in real-time so that any attempt to remove or hide evidence of an intrusion is impossible. By monitoring network traffic, network-based intrusion detection systems can detect malicious attacks. Real-time detection and response is one advantage of network-based detection. In host-based intrusion detection, events have already happened before a host-based intrusion detection system can observe them in log files. In addition, network-based detection can also detect unsuccessful attacks. If placed outside a firewall, a network-based system can identify an activity targeted at resources behind the firewall, even if the firewall rejects the attempts. This unsuccessful attack information can be used to evaluate and refine the security policy. Accordingly, it re-enforces the system to ensure that newer attacks will not pass through [14, 16, 17].

Both network- and host-based intrusion detection system solutions have unique strengths and benefits that complement each other. Integrating network- and host-based approaches is the choice of most advanced intrusion detection systems. Network-based detectors monitor network activity, while host-based detectors sit on various hosts, and both report back to a single management center/administrator. A center may have a view of the network at all levels so that the enforcement of security policy is ensured and a great flexibility in deployment options is provided.

1.5 Organization of the Thesis

This thesis begins with the introduction of problems in network security and intrusion detection systems. In Chapter 2, four distributed intrusion detection system architectures, AAFID, EMERALD, GrIDS and GIDEM, are introduced. Chapter 3 presents a host-based and event-based detector architecture and the detector's implementation is described in Chapter 4. Chapter 5 will state the detector's performance results and Chapter 6 gives a conclusion and possible future directions.

CHAPTER 2
BACKGROUND AND PREVIOUS WORK

There are hundreds of distributed systems available that perform intrusion detection, intrusion prevention and system security checking. Among those, AAFID, EMERALD and GrIDS are representative. At the end of this Chapter, the intrusion detection model (GIEDM) in the CONS lab is introduced.

2.1 AAFID (Autonomous Agent for Intrusion Detection)

AAFID is a distributed monitoring system with an orientation towards intrusion detection developed at the CERIAS, Purdue University. It is an agent-based intrusion detection system that uses many autonomous agents [18] working independently and collectively to perform distributed intrusion detection.

AAFID architecture has four components which are referred to as entities: agents, filters, transceivers and monitors. The AAFID system is designed as a hierarchy of components with agents at the lowest level performing the most basic functions, and monitors and transceivers at the top-level overseeing the operation of agents and filters on a per-host and per-host set basis. Figure 2.1 [9] shows the physical layout of the components in an AAFID system. Figure 2.2 [9] shows the corresponding logical organization of the same system [9].

An AAFID system can be distributed over arbitrary number of hosts, each of which may contain any number of independent agents working simultaneously to monitor certain aspects of a system and reporting anomalous behaviors or occurrences of specific events [19]. Some agents may perform network monitoring functions, while others may

perform host monitoring functions. The results produced by the agents are collected on a per-machine level. In this way, the system may find correlated events that are reported by different agents, such as ones caused by the same attack. Furthermore, detection of attacks involving several different hosts is feasible since the reports produced by each

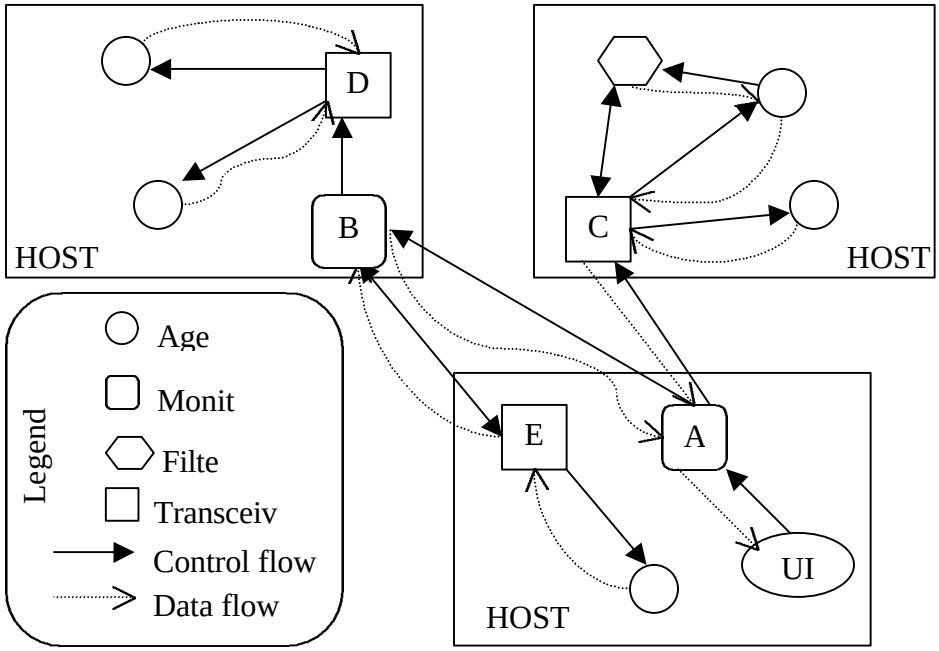

Figure 2.1 Physical layout of an AAFID system [9]

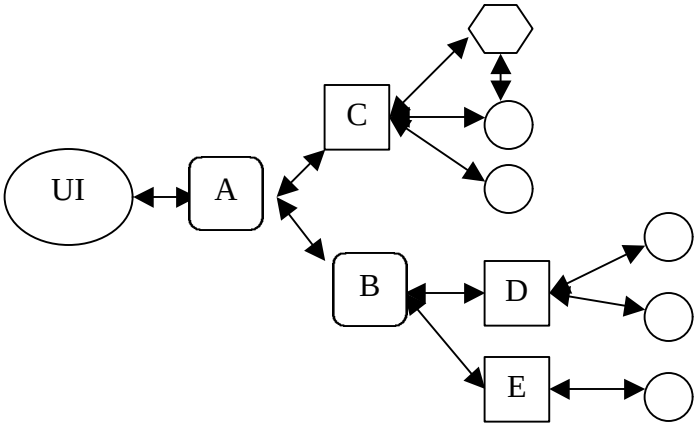

Figure 2.2 Logical representation of the same AAFID system [9]

machine are aggregated at a per-network level. Agents in an AAFID system can be added, started, stopped or removed without altering other components. Additionally, agents may employ mechanisms of reconfiguration and update at run-time without the need to restart the whole system [20].

Agents may use the data collected by filters that are started by transceivers. There is one filter for each data source that provides a subscription-based service to multiple agents. When an agent subscribes to a filter, it specifies the data it needs, and the filter only sends the agent the data that it wants. This is the data selection function that the filter performs. The other function that performed by filters is data abstraction. Filters implement all the architecture- and system-dependent mechanisms for obtaining the data that agents request. Therefore, the same agent can run under different architectures simply by connecting to the appropriate filter [9].

AAFID agents report their findings to one or more transceivers instead of communicating directly with each other. Transceivers are per-host entities exerting control, such as start and stop, over the locally running agents, and they perform analysis and reduction processing on the data received from the agents. Transceivers on different hosts may transmit data to each other. By combining the reports from different agents, transceivers oversee the status of their host. If any suspicious activity takes place, it is reported up the hierarchy to one or more monitors [9].

Monitors are the highest-level entities in the AAFID architecture. They analyze the data received from transceivers to detect intrusions in the network and can do higher-level correlation to detect an event that involves multiple machines. Furthermore, monitors have the ability to detect intrusions that may escape the transceivers. A monitor may report information to a higher-level monitor. They can control transceivers and other

monitors. Monitors can access network-wide data, therefore they have a global picture of the status of the network they are monitoring. In the AAFID architecture, the user interface is separated from the data collection and processing entities. Monitors provide the access point for the user interface to obtain information and send commands [9].

The desirable feature in AAFID is flexibility and scalability. However, it has several limitations. For example, it may be useful to allow more communication between entities in the AAFID architecture, that is, to allow one agent to communicate directly with another agent.

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

EMERALD is a distributed scalable intrusion detection project undertaken in SRI International's System Design Laboratory. It addresses detection of malicious activity through and across large and complex networks.

The EMERALD architecture is composed of a collection of independently tunable service monitors deployed in hosts and performing monitoring functions. EMERALD monitors can be used to prevent DOS (denials of service) and loss of availability. They work independently with application logs and network services to detect and respond to malicious activity at the operating system and network layers, and can interoperate to form an analysis hierarchy. Hierarchical data processing helps scalability to large networking applications.

The EMERALD monitor architecture is illustrated in Figure 2.3 [21]. The EMERALD architecture employs three analysis units: a signature engine, a statistical profile engine and a resolver within the monitor. These three types of units surround the target-specific resource objects.

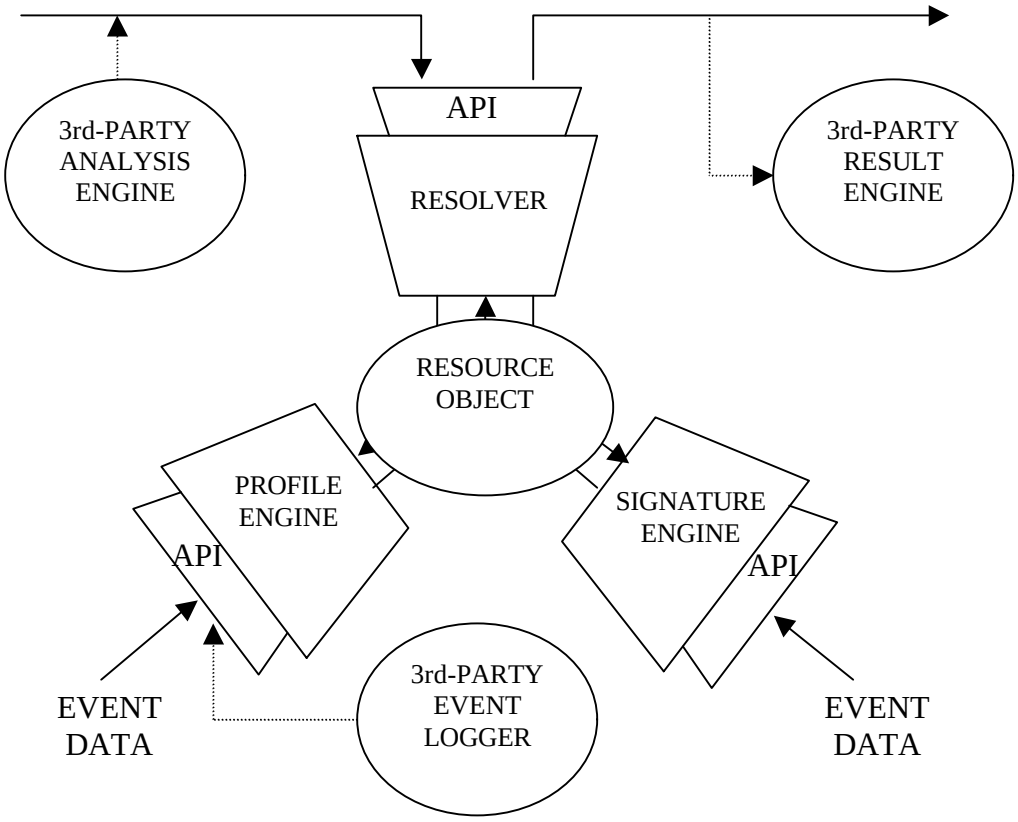

Figure 2.3 EMERALD monitor architecture [22]

The signature-based and profile-based anomaly detection subsystems will analyze data at various logical layers of abstraction, on appropriate platforms to minimize the amount of data that must be analyzed at each logical layer. Signature-based analysis engine uses historical records to build normal detection models to verify a new instance as a known intrusion activity. The profile-based component is used to monitor network traffic and detect unanticipated anomaly modes [21].

A resolver is a countermeasure decision engine interpreting analysis results generated from signature and statistical engines. It correlates alarms from anomaly detectors throughout an enterprise network. The EMERALD resolver provides further analysis of security violations and intrusion events and implements responses to intrusions, which involve reconfiguration of the intrusion detection system. Signature

analysis and the integration of an expert system also improve the correlation capabilities [21].

EMERALD monitors demonstrate a streamlined, distributed intrusion detection system providing localized real-time analysis of multiple network services (e.g., FTP, SMTP, HTTP) and infrastucture (e.g., routers or gateways). An EMERALD monitor may interact with its environment either passively or actively. Monitors provide a well-defined application programming interface (API) to interoperate with other third-party intrusion-detection tool suites, such as deriving and receiving event data and analytical results among other third-party security services [21].

Each monitor also contains a resource object that provides a reconfigurable library of information to customize the monitor to the target analysis. The monitor code base is analysis target independent. Therefore, the EMERALD monitor is scalable with no modification to anything except the content of the resource object [22].

EMERALD presently includes several eXpert components that use P-BEST rule based inference. The EMERALD eXpert is a signature-analysis engine based on the expert system shell P-BEST. P-BEST is an expert system shell with a set of rules that enable it to identify or suspect an anomalous activity in the system. P-BEST-based eXperts may be independently distributed to analyze the activity of multiple network services or network elements [22,23].

EMERALD builds multiple local monitoring capabilities coordinating the distribution of analyses to detect both local attacks and coordinated attacks such as distributed DoS. It can interoperate with other analysis platforms within its own scope. Providing global rather than local analysis is an important issue that further research should address.

2.3 GrIDS (Graph-Based Intrusion Detection System)

GrIDS is intended to detect large-scale automated attacks on networked systems and was designed at UC Davis. It is a hierarchical system, which puts together results of localized host-based and network-based intrusion detection systems into graphs and is able to aggregate those graphs into simpler forms at higher levels of the hierarchy. The whole GrIDS hierarchy requires an Organizational Hierarchy Server (OHS) to oversee the operations and to provide global view of the hierarchy [24].

GrIDS employs data source modules running in each host to report incidents and network traffic to graph engines that build a causal structure of activity in the network called an activity graph. The activity graph allows automated attacks to be detected in near real-time. For scalability to large networks, these graphs are constructed locally and passed upward in the hierarchy where a parent engine maintains a graph of coarser resolution. Tracking of a worm is an example of building such an activity graph which is shown in Figure 2.4 [24]. The nodes of an activity graph correspond to hosts in a system, while the edges in the graph correspond to network activity between those hosts. In addition to forming graphs from network communications, GrIDS allows the nodes and edges to be annotated with attributes that supply additional information about the events.

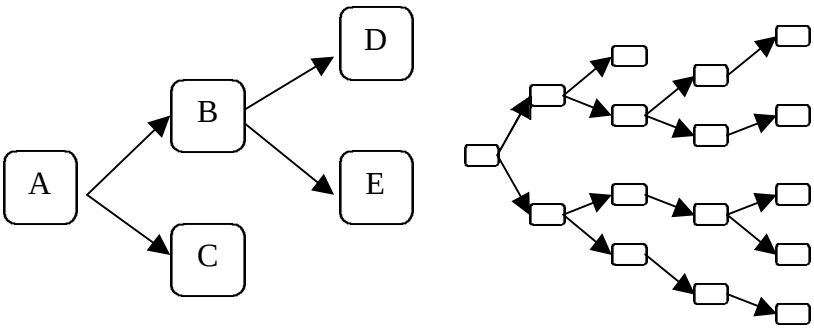

Figure 2.4 The beginning of a worm graph, and a more extensive view of the same worm [25]

User-customizable rule sets describe to GrIDS how to build these graphs to find certain activity patterns and when to send alerts. These graphs are then compared against those patterns of intrusive or malicious activities, and if they look similar an alarm and corresponding reactions are generated. The rule set has rules to build the graph and also assessment rules to evaluate the graphs and to take appropriate actions. Not all types of network behavior are related to each other. GrIDS implements several independent graphs representing different types of activities, as specified by the rules. It may be much easier to form and to analyze the structures independently than a large single graph containing unrelated information [25].

GrIDS detects large-scale correlated or automated attacks by checking to see if the constructed activity graphs at any level have violated some policy as captured in a rule set. Policies are compiled into rule sets [26].

GrIDS provides a scalable design for intrusion detection in large networks. It allows specifying finer grain access control policies than firewalls. It also has some limitations. It may not be possible to capture all intrusions as violations of an explicit policy. Further, since the system depends on a centralized OHS for hierarchy management, it may not be useful in a large dynamic network.

2.4 GIDEM (Generic Intrusion Detection Model)

A real-time generic intrusion detection model (GIDEM) is presented in Seetharaman Balasubramanian's thesis [27]. The system consists of coordinators, detectors, tranceivers and response agents. The structure of the design is shown as Figure 2.5.

The coordinator is the highest level of the system. The design of the coordinator applies an IETF model [10], which is a part of the analyzer and the manager. It provides

the event analysis and detection system control. When detectors report events to the coordinator, the coordinator parses them. Using a rule-base, the coordinator decides whether to trigger an action and issue a specified response to the corresponding response agent. The coordinator also can start, stop and reconfigure the detectors as needed.

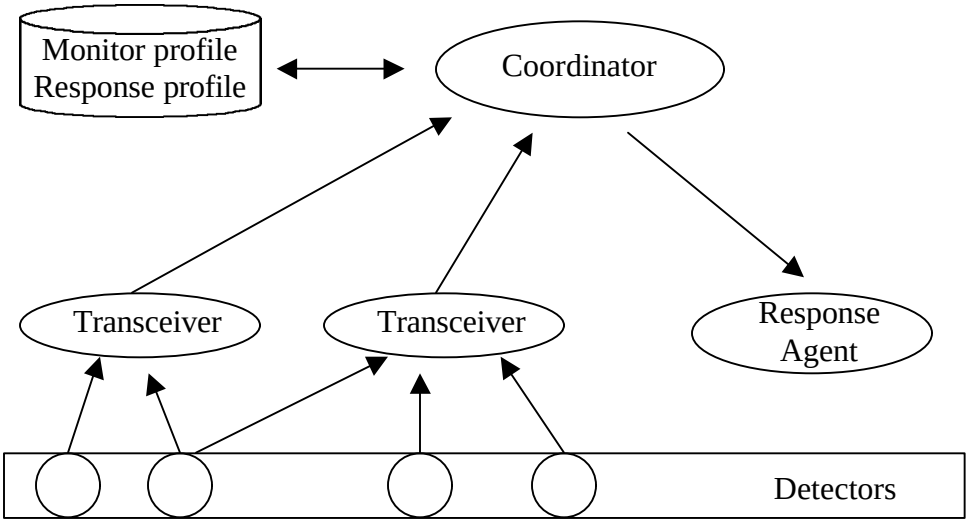

Figure 2.5 Architecture of GIDEM [27]

The detectors distribute in the entities of the GIDEM. Their primary duties are to collect data and conduct possible data analysis. They have the capability to communicate with the transceiver and the coordinator. The idea of the transceiver is similar to that of the AAFID architecture. The transceivers provide the external communication interface for each host. Basically, the transceivers have the same functions as the coordinator does. The only difference between the coordinator and the transceiver is that transceiver only controls one or limited number of detectors while the coordinator globally controls all the detectors in GIDEM. The responder is responsible to execute commands from the coordinator. In addition, the user interface is modeled as a response agent.

This thesis presents a host-based and event-based detector for GIDEM. The goal of this thesis is to design a near real-time host-based detector, which can analyze raw data quickly and efficiently, eliminate the redundancy in the log data and be capable of reconfiguration and recovery. More details are explained in Chapter 3.

CHAPTER 3
DESIGN

The purpose of this thesis is to design a host-based detector that can efficiently monitor and analyze routine log activities and report an anomalous intrusion to the coordinator. Its duties are to bring and transfer the log data from the log file in near real-time, to parse the data and rebuild it to the standard format. Then it analyzes the routine login data and reports any anomalies. Finally, it compresses the raw data to save space in the log file. In this chapter, the detector design is described in two parts: detector components and detector strategy.

Detector components include four functional units: data collector, data buffer, data operator and data container. After the detector is initialized by loading its configuration file and user profile, all of detector components are started based on the configuration information. The detailed design for each component is presented in the following sections.

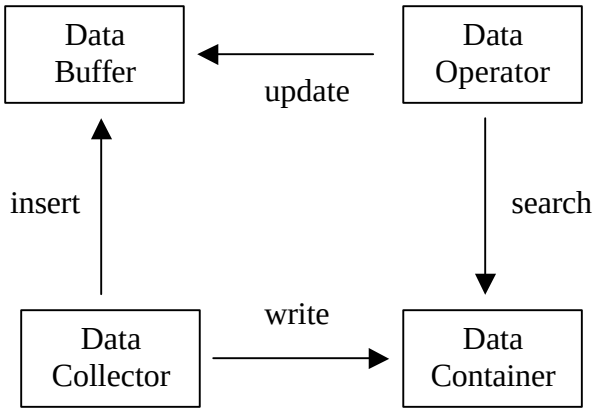

Figure 3.1 The detector components

3.1 Detector Components: Data Collector

Raw messages are generated by *Syslogd*, which is a collection mechanism for logging various messages generated by the kernel and applications running on the UNIX operating system. Every message collected by *Syslogd* is associated with a level of importance, which consist of the facility – the system component from which the message originated. Combinations of priority and facility specify actions related to the message. The values of both parameters (priority, facility) are described in detail on the syslog(3) man page as shown in Table 3.1 [28] (priorities are listed from lowest to highest below). The priority levels between debug and warning is the major concern in this design.

Table 3.1 Syslogd facility and priority [28]

Facility	Description	Priority	Description
Auth	Used by authorization systems (login)	Debug (lowest)	Normally used for debugging
Cron	Used for the cron and at system	Info	Informational messages
Daemon	System/network daemons	Notice	Conditions that may require attention
Kern	Produced by kernel messages	Warning	Any warnings
Lpr	Printing system	Err	Any errors
Mail	Mail system	Crit	Critical conditions like hardware problems
Mark	Internally used for time stamps	Alert	Any condition that demand immediate attention
News	Reserved for the news system	Emerg (highest)	Any emergency condition
User	Default facility, used for any program		
Uucp	Reserved for the uucp system		
Local0…7	Reserved for local use		

The function of the data collector is to collect data from the *Syslogd* daemon, multiplex the raw log to the corresponding data buffer and finally pass the formatted log messages to data container. It has three parts: data reader, data parser and data compressor. The responsibility of data reader is to obtain the raw data in real time or near real time and pass it to data parser. The data parser is to parse the raw data string that is read from the *Syslogd* and to multiplex it to the corresponding type of data buffer. Another responsibility of the data parser is to assemble the data into the compressed file format, which is delivered to the data compressor. When the data compressor receives the log data (which is in standard format), it converts it into a binary format and passes it to the data container.

3.2 Detector Components: Data Buffer

When data is converted to the standard format by the data collector, the data buffer holds the data in memory for fast analysis. However, holding all this information wastes memory and is unnecessary. The data buffer consists of two basic units, the data indexing tree and temporally ordered data indexing list, which has a sliding window. The sliding window plays a major role in dynamically maintaining all log data read from the log file.

Generally, the IP address is composed of four numbers between 0 and 255. A tree structure is selected by the data indexer to maintain an index of all log data. The depth of a detector tree is four. The main advantage of the tree structure is to provide for faster searching performance than a circular list buffer does. If the frequency of log data redundancy is high, saving more space will be another benefit as well.

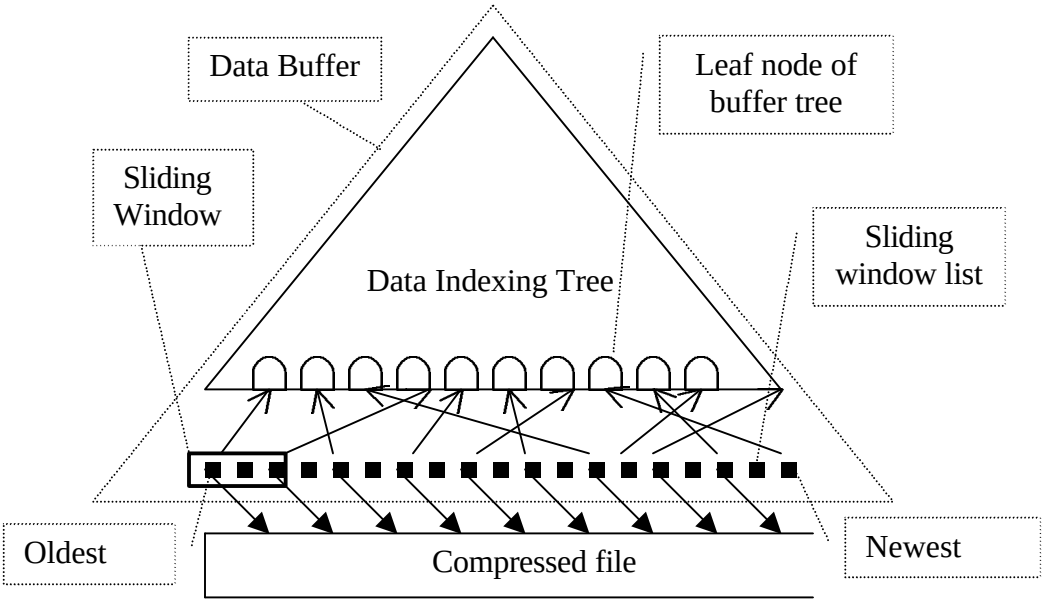

Figure 3.2 Architecture of the data buffer

However, if a detector tree keeps growing, it will finally reach its threshold and use up all the pre-allocated system resources. To monitor the capacity of the detector tree, a data list is created and is extended as the detector tree is modified. Every leaf node of the buffer tree (representing an IP address) has a link with one node in the sliding window list (representing timestamped audit event). When time passes a certain interval, an aging policy is applied to the data buffer. That is, the sliding window will move backward on the list with a certain size, even if an anomalous intrusion is detected. All out-of-date log data to the left of the window will be removed from the buffer tree and the sliding window list. As stale sliding window list nodes are deleted, the information on the corresponding data indexing tree nodes (both leaf and non-leaf) are necessarily updated. Therefore, the capacity of the buffer tree is changing. Selecting a suitable sliding window moving rate will result in increasing the efficiency of the buffer tree.

3.3 Detector Components: Data Operator

Monitoring and updating the data in the data buffer is the major function of the data operator. It contains a series of engines that perform searching, inserting and deleting data on the buffer tree that function upon request. To keep every engine running without blocking any other engine, a multi-threaded detection system is selected. Each thread does not have to wait for any other thread to finish. All the threads proceed simultaneously with fair concurrency control. Properly formed threads can often improve the performance of an application and they do not incur significant implementation overhead. They effectively give good performance for the effort expended.

The searching engine seeks to search the information of a specific or suspected IP address. Usually, the searching starts from the root of the buffer tree. The insertion engine is requested when needed. When standard log data comes from the data collector, it will traverse the buffer tree and update the corresponding node's information. If it is a non-existing node, the insertion engine creates a new node. The deletion engine starts to work upon being informed that one specific IP needs to be deleted from the buffer tree. The deletion request will be needed in either of the following cases: 1) all the sliding window list nodes related to that IP address appear to the left of the sliding window; 2) an anomalous intrusion is detected and confirmed.

Besides its updating function, the data operator is responsible to communicate with the upper level coordinator and to report an anomalous intrusion at different warning levels. For example, when an IP reaches one warning level's threshold, the detector needs to report to the coordinator that a warning state is occurring and to consider that IP as suspect. When a suspected IP later reaches the threshold of higher level warning state, the detector needs to report that higher state to the coordinator again.

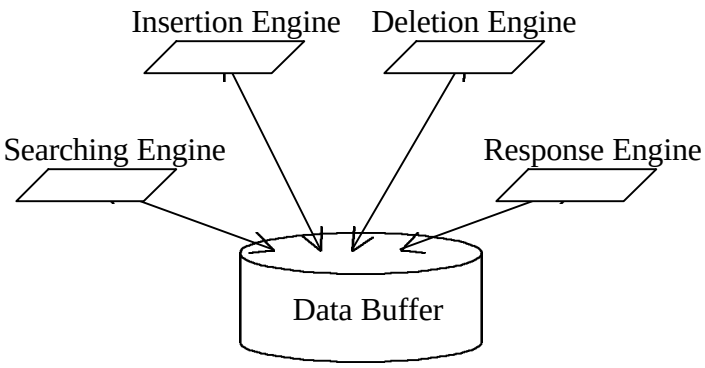

Figure 3.3 Architecture of the data operator

A client and server model is applied to implement the relationship between the detector and the coordinator. When the detector finishes the data analysis of the raw log message, the data response engine of the data operator cooperates with the coordinator. Its responsibilities include notifying the coordinator about its initial launching, sending any detection result to the coordinator and then waiting for the coordinator's response. When the coordinator makes a decision on the reported event, it will send the response back to the detector. Once the response engine obtains the response from the coordinator, it will update the configuration of the detector or user profile, if necessary. Then the detector will start to execute detection based on the new configuration. Therefore, it provides the coordinator a facility to control the detectors operation. In a word, the data operator functions by keeping the buffer tree running more flexibly and efficiently.

3.4 Detector Components: Data Container

When the data collector delivers the standard format data to the data container, the data container then writes the data into the compressed log file. Three issues force the designed detection system to compress the log file. First is the growth of the log file,

second is Unix or Linux log file rotation policy and third is the detection system recovery. Left unaddressed, the first issue will result in a huge log file occupying too much storage space. The second and third issues will cause losing some old data which might be useful for slow attack detection. In the designed detection system, all login data needed by the detector is necessarily stored in a compressed file so that two advantages, keeping records and saving space, are realized. Keeping records helps necessary polling analysis. When a polling analysis request is invoked, all of the records can be grouped according to the requirement. Saving space is implemented by converting each log data format to a binary format. The compressed log file's creation depends on help from the data collector. The compressed data interpreter helps to read data from a compressed file. More detail on compressing files is given in Chapter 4.

3.5 Detection Strategy

Data analysis is based on two different situations, namely failed login and successful login. Regarding login failures, keeping a count of how many attacks from a certain IP address is the most important key for intrusion detection. A login failure may occur from accidental failure or intentional attack. Occasional failure login occurs when one user forgets his password or mistypes it. Although login failures from same remote node may appear for different usernames or passwords, their source IP should be identical. Therefore, only one leaf node in the data buffer can represent all such login attempts. That is, space within the data buffer can be saved by eliminating redundant records with the same IP address. In the process of detection, the occasional login failure can easily be discovered because it is not repeated over time. Once the user recalls the correct password or obtains it from the administrator, this type of login failure will not

appear again for an indefinite period. Therefore, its representation in the data buffer will not be updated. With the motion of the sliding window, such login failures will be removed from the data buffer when they become out-of-date and appear to the left of the window. As a result, the capacity of the data buffer is made available to hold other different IP logins.

Prevention of intended attack is primary function of the detector. Such attacks have two kinds of format: burst attack and slow attack. Burst attack implies many login failures happen within a short time. This kind of attack can be from a single source or multiple sources. As for those from a single source, they can be detected by finding the counter of the leaf node increasing dramatically. When the login counter reaches detection threshold, it can be detected immediately. Multi-source attacks have two formats: non-targetted attack and targetted attack. The solution for non-targetted attack is the same as that for a single source attack. Targetted attack indicates that multiple sources are attacking the same user account simultaneously. A polling analysis on that account will be necessary. Checking each user account's login data within a certain time can detect targetted attacks. Such a polling analysis can be invoked by the data collector when it discovers login time has surpassed its reasonable threshold.

A hacker may attack the system more cautiously and patiently. He may not try each attack continually but keep trying periodically; this is a "slow attack." To detect this kind of attack, a polling analysis needs to be based on the weekly or monthly log data. With a more extensive search, the possibility of detecting such slow attacks is increased. For example, if there may be an IP address of a hacker from which many failure login actions originate. However, since the interval time between two adjacent actions is

longer than that of the sliding window moving interval time, the hacker's data is always removed from the data buffer and he escapes detection. Fortunately, this can be discovered by weekly or monthly statistical analysis.

A lucky hacker may get a chance to log onto the system successfully due to various reasons, e.g., obtaining a normal user's information in an inappropriate way. Depending on knowledge-based successful login analysis, the chance to catch an unauthorized login is enhanced. If the same "user" logs in from two separate sites at the same time or in a short time, that account can be considered to be "infected." If a user logs in at unusual times, that account can be suspected of being infected. In order to find if there is an intrusion, it is necessary first to confirm the authenticity of the user by coordinator. If coordinator or detector gets negative response from the user, a polling searching on previous failure logins from that suspected IP is necessary.

3.6 Summary

A host-based detector is designed to deal with data extraction, data analysis, data compression and data storage with the Linux Syslogd facility. The architecture of the detector is shown in Figure 3.4. Real-time data acquisition makes it possible that an intrusion detection system can be running in real time or near real time. Meanwhile, such acquisition also reduces the chance that a hacker may remove a record from the log file. A tree-like data buffer enhances data analysis performance, which is most important for abrupt attack detection. Furthermore, with the help of sliding window aging, the data buffer can dynamically change its size thus enabling it to continuously monitor login actions efficiently. To keep a data buffer of a relatively small size running efficiently without losing information, a compressed file acts as a complementary part of the data

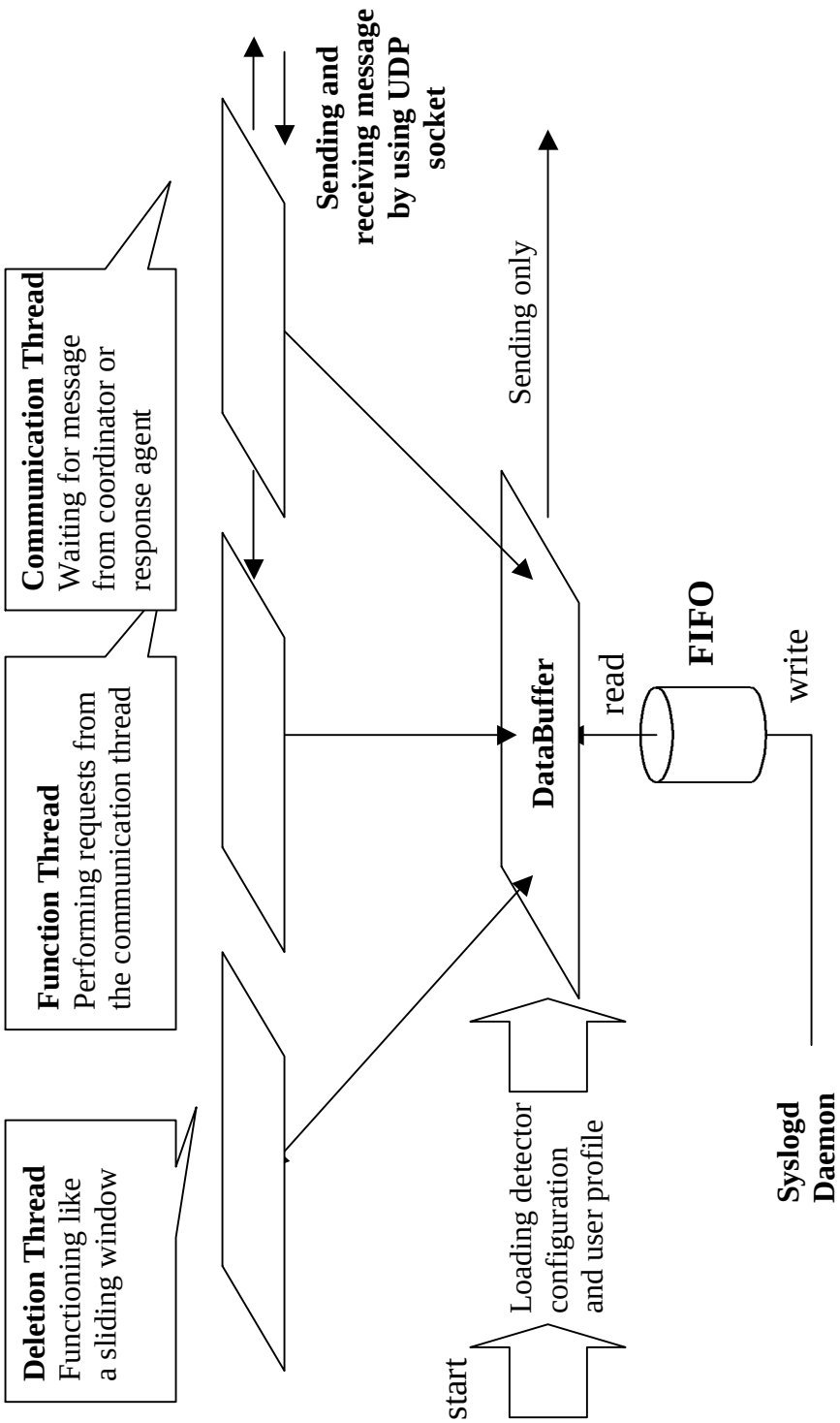

Figure 3.4 Structure of Host-based Detector

buffer. Since all out-of-date as well as current data are recorded in the file, a polling analysis then becomes feasible, which is beneficial for slow attack detection. As a supplement, knowledge-based detection is added to detect those anomalous intrusions with successful unauthorized login. As a result, the host-based detector is designed to be capable of re-configuration, to run concurrently as a multithreaded system, dynamically to detect both login failures and successful unauthorized logins and to be able to do post-event polling analysis.

CHAPTER 4
IMPLEMENTATION

The implementation of the detector is described in five parts: initialization of the detector, data collection, data buffer creation, successful login detection and system controlling. All parts of the detector are written in C/C++ programming language.

4.1 Initialization of the Detector

To be an easy-controlled detector, the designed detection system has implemented a configuration and re-configuration policy. A previously prepared configuration file specifies major characters of the detector. The configuration file must be read when a detector starts. During the detection process, other agents such as coordinators or response agents can modify the configuration file. Each time this file is changed, the detector is requested to re-read it and operate on a new configuration.

As a part of configuration and knowledge-based detection, a user profile is loaded at the beginning of detector initialization. The profile helps the detection system in making a judgment whether a successful login is suspicious. During the process of detection, the user profile can be updated to make the detection knowledge system more accurate.

After finishing the configuration procedure, the detector needs to register with a remote coordinator, at a location that is present in the detector's configuration file. The registry process is completed by using UDP datagram socket communication.

4.2 Data Collecting in Near Real Time

Once the detector is initialized, the next step is to obtain the raw log data in real time or near real time. The original message data are created by the syslogd daemon. Actions are associated with the syslogd's combinations of facilities and priorities. Those combinations specify what can be done with the actual log messages that match a given combination as shown in Table 4.1 [28].

Table 4.1 Sample actions of syslog.conf [28]

Action (example)	Description
/dev/console	Send messages to devices
/var/adm/messages	Write messages to files
@ loghost	Forward messages to a loghost
fred, user1	Send messages to users
*	Send messages to all logged-in users

Under Linux, syslogd can write a log message into a named pipe (FIFO). This makes it possible for the detector to obtain log messages from the named pipe in real time. In /etc/syslog.conf, a new line is added like:

. !/var/log/syslog.pipe

This means that all log messages are required to write into a named pipe, "syslog.pipe." When a detector is activated, the named pipe is opened by the detector with a "READ_ONLY" flag. Then one thread is kept blocked and awaits the passage of log messages from the opposite side. Once a new log message is written into the pipe

from the other side, the receiving thread can obtain it without any delay. From time to time, obtaining real-time data is implemented by that thread.

4.3 Implementation of Data Assembly and Compression

The data in the log file are saved in ASCII text format. Although ASCII format is more user-friendly, it is wasteful of disk storage space. If the data is saved in a binary encoding, it will dramatically reduce the storage size of the log file. Therefore, it is necessary to convert raw data from its text format, such as ASCII, into a binary format. For each data entry, there are five significant attributes: timestamp, user id, source IP address, login type and flag. One data entry can be formed as shown in Table 4.2.

Table 4.2 The format of data entry

User id 16-bit	Type 6-bit	Flag 2-bit
Timestamp 32-bit		
IP address 32-bit		

Each entry contains two fields: header and content. The header includes flag bit, type bits and user id bits. The flag bits have four different combinations which are listed in Table 4.3. Those four different combinations result from the comparison between the current and the previous data entry. If they have the same IP, the content part of the entry only has a timestamp field so that it reserves storage room. Also, because the first bit of the flag indicates whether current entry is from another IP, it functions as a delimiter within the whole compressed log file. If its first bit is equal to 1, the content field will only have a timestamp field.

Table 4.3 The description for flag bit

Flag bit	Description
00	Same user from different IP address
01	Different user from different IP address
10	Different user from same IP address
11	Same user from the same IP address

The IP field contains 32 bits which correspond to four numbers within the 0-255 range. Within the timestamp field, the detection system chooses data type **time_t** to represent calendar time. When interpreted as an absolute time value, time_t represents the number of seconds elapsed since 00:00:00 on January 1, 1970, Coordinated Universal Time (UTC). The login types can be classified in varying formats, namely ssh, ftp, rlogin, etc. Type bits can represent a maximum of 64 different login types. Table 4.3 lists several kinds of types.

Table 4.4 Type bits for different login

Type	Binary code	Type	Binary code
Failure ssh	000001	Successful ssh	100001
Failure ftp	000010	Successful ftp	100010
Failure rlogin	000011	Successful rlogin	100011
…	…	…	…
Unknown failure	000000	Unknown success	111111

Once the raw data is translated into standard format, the data compressor will write standard data word by word. In case that the system crashed, the detection system can be recovered with the help of the compressed file.

4.4 Implementation of Creating the Data Buffer

After the raw data is tokenized and converted into the standard format, it is ready to be added into the data buffer for analysis. In previous work, data redundancy has been a severe problem in the circular list buffer. Such a buffer is not flexible and efficient for modifying, updating or deleting data records. In this detection system, a tree-structure is being applied to solve these conflicts. Such a tree-structure is based on the idea of hierarchy of IP addresses. It consists of four levels that are related to IP address format. The root is the highest level. The address mapping between levels is aided by a hashing function. A hashing chain is used to save memory space and achieve fast searching performance. The relationship between levels is “pseudo double” linked. The upper level node (parent node) has a pointer which points to its hashing chain. Every node (child node) in the hashing chain has a pointer which points to its parent node.

In every node, the information is kept in the following format.

```
Class Node{

        int             num_child;
        Node            *ptr_parent;
        time_t          least_ts;
        time_t          latest_ts;
        HashingChain    *ptr_child;

};.
```

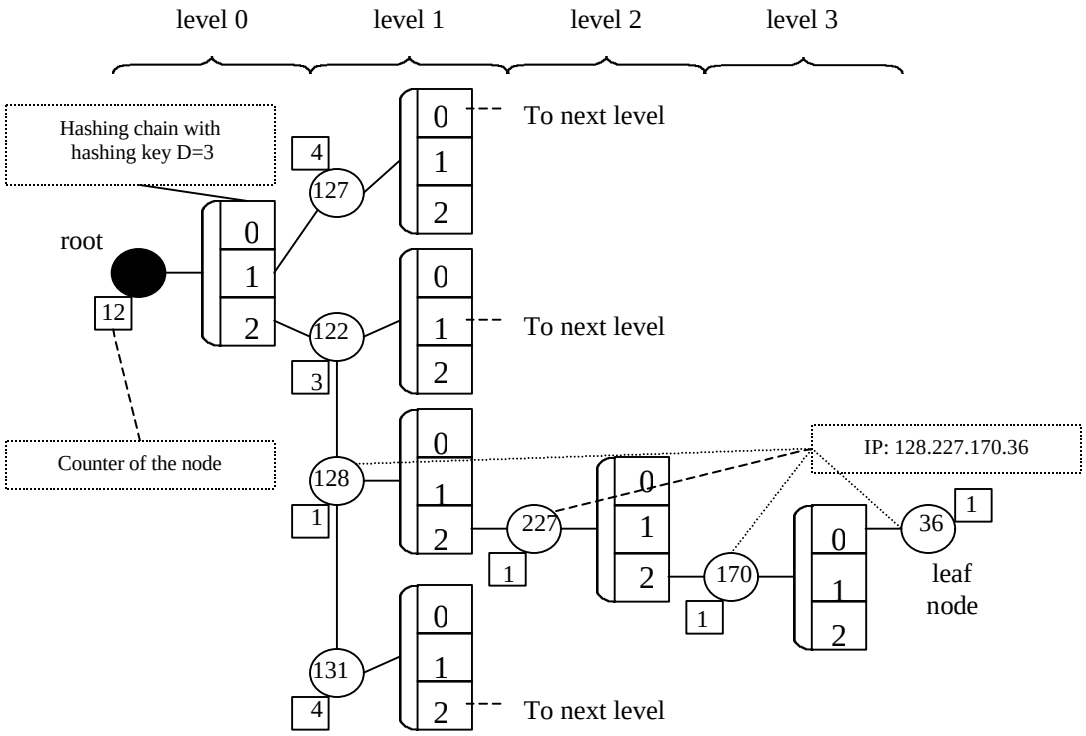

Figure 4.1 Hashing tree of data buffer

The **num_child** signifies the total number of its children; **ptr_parent** is a pointer to its parent; **ptr_child** is a pointer to its children level; **least_ts** is its oldest timestamp; **latest_ts** is its most recent timestamp. Because every data entry in the compressed file cannot possess a pointer that refers back to the node on the indexing tree, a linked list is maintained between the data indexing tree and the compressed file. The leaf nodes on the indexing tree have a one-to-one relationship with the nodes of the linked list. At each node of the list, there are only three attributes that need to be stored: **ptr_node**, **num_byte**, **node_ts** and **prev_ts**. The **ptr_node** is a pointer which points to the leaf node on the indexing tree. The **num_byte** records the beginning location of each data

entry in the compressed file. The **node_ts** denotes the timestamp of the pointing leaf node on the indexing tree. The **next_ts** is a pointer to point to next list node with the same IP address, if any. The indexing list node format is shown below.

```
Class ListNode{

        time_t          node_ts;
        int             num_byte;
        TreeNode        *ptr_node;
        ListNode        *next_ts;

};.
```

Detection tree initialization starts from the root. It proceeds down each non-leaf level and creates the appropriate values for num_child, ptr_parent, ptr_child, least_ts and latest_ts. When it reaches the leaf level, it updates its counter, ptr_parent and latest_ts. Meanwhile, it updates the ptr_node of the corresponding node in the linked list so that a one-to-one relationship is built up. When the sequential new log data are read, the detection tree is updated from the root and goes down each non-leaf to the leaf level. If the expected child does not exist, the tree creates a new tree node with appropriate values for each of its attributes. Every time a leaf node is formed, the relationship between the indexing tree and linked list is again updated.

4.5 Implementation of Successful Login Detection

Even if a successful login occurs, it does not mean that it can be totally trusted. Two possible types of unauthorized logins need to be addressed. One is the same user logging in from a different IP address within a very short time; another is a user's login at an unusual time slot.

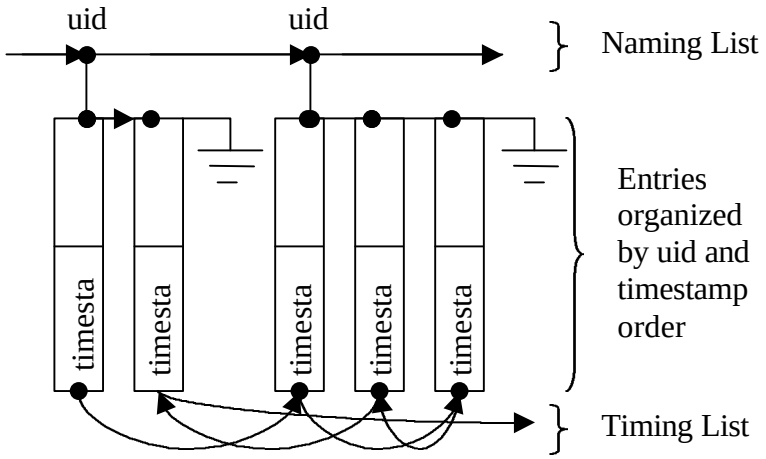

Figure 4.2 Data structure of naming and timing lists

This detection system needs to maintain a profile for each user. The profile contains the basic login information for each user, such as normal login time slots and source IP domain. The user profile is pre-loaded at the beginning of detection to build a user knowledge table in the system. As the detector running, the information on the knowledge table can be refined if requested by the coordinator or response agents.

The detection buffer for successful login is implemented in a multi-linked list format that contains the naming link and timing link. Each entry on the list contains a user id, source IP address, timestamp, file location and deviation. The user id is the key for each entry and the naming list is sorted by user id. A sorted, linked list of user ids is built, with each uid node pointing to a linked list of successful logins for that user id in timestamp order. All successful logins are also double threaded based on timestamp order; this is called the timing list. The timing list also has pointers for the head (most recent) and tail (oldest) entry. Each entry on the list also has a file location field to record

its location in the compressed file and a deviation field to record suspect information. The value of the deviation field is inherited from the entries with the same user id. The format of the successful buffer node is given below.

```
Class SListNode{

int         uid;                /* user id */
time_t      node_ts;            /* timestamp */
int         ip;                 /* address ip */
int         location;           /* the starting location in the compressed
                                       file */
SListNode   *ptr_prev_id;       /* pointer to previous different id */
SListNode   *ptr_next_id;       /* pointer to next different id */
SListNode   *ptr_next_siblingid;`/* pointer to next same id */
SListNode   *ptr_prev_ts;       /* pointer to previous timestamp */
SListNode   *ptr_next_ts;       /* pointer to next timestamp */

};.
```

When a new entry is added, it first finds its user id in the naming list and then appends itself to the list of entries for that user id. If there is no matching user id, the detector will create a new user id node and insert it at the right place within the naming list. After the naming list is updated, the timing list will be adjusted accordingly using the head pointer. After finishing the insertion of the node, data analysis is then performed. In every entry's deviation field, there are four basic counters to monitor four aspects of basic detection: normal login times, login times at an unusual time slot from a normal IP domain, login times at the normal time slot but from an unusual IP address and login times from an unusual IP address and at an unusual time slot. In the designed system, there are different threshold levels for each category.

When the checking process starts, the new entry's timestamp and source IP address are first compared with those in the user knowledge table. If an unusual login time slot or source IP is found, its deviation field will record such data. Then, its

timestamp and IP address are compared with the one preceding it in both sub-lists of the naming list and timing list. With this kind of checking system, two kinds of anomalous behaviors easily can be found. The comparisons within the naming list can detect an anomalous login from two different IP sites within very short time. Comparisons within the timing list can detect a user login at an unexpected time.

4.6 Implementation of Multi-thread System Manipulation and Concurrency Control

The designed detection system applies a multi-thread mechanism so that the set of global variables and function calls in the data buffer can be fairly shared and accessed by every thread. A simple approach would be to create a thread for each request received. In this designed system, **pthread** in the GNU thread library is used to create a thread.

There are several situations that will require a thread request. At the stage of the detector initialization, the detector needs to launch two basic threads: a listening thread and a deletion thread. The listening thread functions as a response engine in the data operator being designed. Since the designed detector uses a client/server model in the communication between the detector and the coordinator, the listening thread's responsibility is to receive the message from the coordinator. The listening thread prepares to deal with several messages:

1. When the coordinator requests current detector state, the listening thread will create a thread to obtain the current state and send it to the coordinator.
2. When the listening thread receives a reconfiguration request, it will re-read the configuration file in the assigned location and update the current detection configuration. Since all configuration variables are shared global

variables, this recent modification will immediately reflect on the current detection system.

3. If the listening thread gets updated user information, it will change the appropriate item of that user in the user knowledge table and reset the deviation field of the buffer entry for each user.
4. If the listening thread receives many messages that are not sent by the coordinator, it will report to the coordinator that its port has been attacked.
5. If the coordinator sends a “quit” message, the listening thread will set a global variable, “TERMINATION,” as “true” so that every thread will terminate its execution and the whole detector quits.

Another thread created at the initial stage is called the deletion thread. It implements an aging policy in the detection system. As long as the system is running, the log data will keep increasing. A fixed data buffer can not handle too many data storage requests. It is necessary to delete some out-of-date data.

A sliding window is selected to continue to update the data on the detection buffer, which is moving along the timing linked list of data buffer. A global variable is used to record the moving speed of the sliding window. It controls the moving speed of the sliding window. As the sliding window moves, some old records on the indexing timing list are removed from the data buffer. The deletion thread directs the pointer **ptr_node** to delete the right node and to update its parent’s information. Therefore by such reduction, the size of the detection buffer can be moderately lessened. The original moving speed is read from the configuration file. After that, if the thread finds the data buffer size is reaching a warning limit, it will increase the moving speed. On the other

hand, if the thread finds the current buffer size is far below the warning level, it will decrease the moving speed. Therefore, the implemented aging policy always keeps the data buffer of a reasonable size, which effectively utilizes the capacity of the data buffer.

However, at present, the hackers have become more sophisticated and patient. They can use slow attack strategies to avoid being caught by any fixed length data buffer. The deletion thread has to deal with those attacks. The key is to conduct a post-event periodic analysis. The deletion thread launches a temporary thread to scan the monthly compressed file and builds a temporary buffer for analysis use. If any anomalous intrusion is found based on the assigned detection threshold, it will report it to the coordinator. Once the polling analysis finishes, the temporary thread is terminated and all occupied system resources are released. The detection system can commit the polling analysis in the very early morning so that it does not increase the system load too much during the daytime.

In the process of data analysis, two threads are needed. First, when the insertion engine of the data operator encounters the leaf node of the indexing tree, it finds that the counter of the leaf node has reached a certain warning level's threshold. It needs to launch a communication thread to send the message to the coordinator. Secondly, when the system needs information on a specific IP address, a searching engine thread is created. It traverses the data indexing tree and writes the requested information to a file or sends a message to the requesting coordinator.

Although a multi-thread system enhances the detection system's performance, it also brings up an important concern--concurrency control. Because there are shared variables and memory addresses among the threads previously referred to, it is necessary

to create basic synchronization objects to implement concurrency control. The simplest synchronization object is the *mutex* semaphore, which is used to allow only one thread to access to a resource at a time.

In this thesis work, a simple *mutex* object is applied to control the concurrent accesses in the multi-thread system because insertion and deletion of the data buffer will occur mostly during detection. As a result, the critical section, data buffer, can be accessed by only one thread so that there will be no data inconsistency or erroneous results.

4.7 Summary

This detector is devised as the design required. The use of named pipe makes intrusion detection close to real time so that its use prevents the log message from being modified by an intruder before defection takes place. Also, the detector enhances surveillance and response performance. The configuration file and user profile facilitate configuration and adaptation to the system and users. Reconfiguration is possible as well. The relatively small size of the data buffer ensures a minimal system overhead. Its sliding window's aging policy enhances data buffer utilization. If the detection system crashes, the data in the compressed file can rebuild the detection system from the breakpoint. Successful login detection, bolstered by user profiles, makes the detector more knowledgeable. Finally, with the help of a multi-threaded system and a client/server model, the designed detection system can run continuously without supervision. Meanwhile, the coordinator can control this detection system by using a message passing mechanism.

CHAPTER 5
TESTING

How efficiently the data buffer performs will determine the overall detection and response performance, the testing is mainly focused on the data buffer testing.

Testing on the data buffer is accomplished on the host of 128.227.170.36 (tiger.cons.cise.ufl.edu) , whose system configuration is RedHat Linux 7.0 on Pentium II 266 processor with 96 MB RAM. A sample log file containing 300k log data with fair redundancy is created as an input sample. The performance test is conducted by inserting 10k, 30k, 50k, 100k and 300k bytes data into the data buffer respectively. Then, the deletion and searching testing is applied to those data as well. Since the minimum time scale is one second, a looping testing code is applied to increase the total running time so that the testing time is much longer than the minimum time scale. As a result, the system errors are reduced as much as possible. Beside the total amount of test data, the value of hashing key in data buffer tree is chosen as another important variant on the detection performance testing.

Fast searching and deletion performance: the testing results on the performance of the deletion and searching are shown in Figure 5.1 and 5.2.

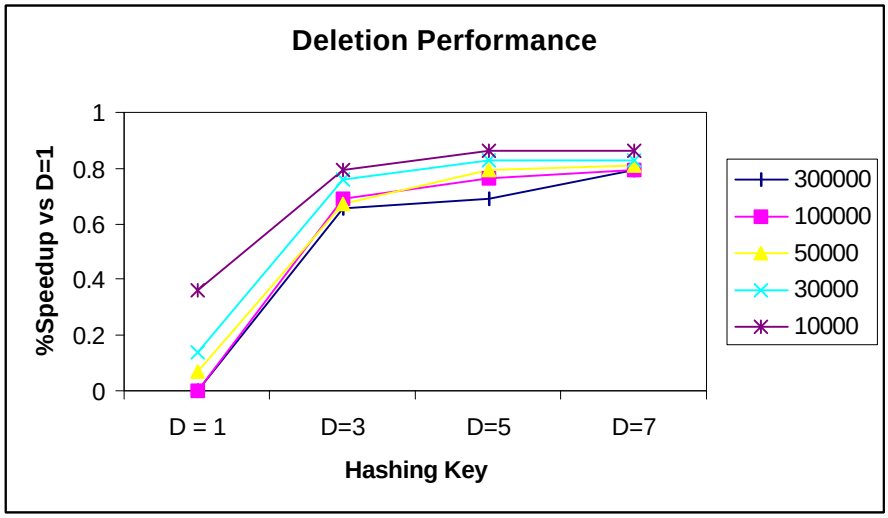

Figure 5.1 Performance of the deletion of the data buffer

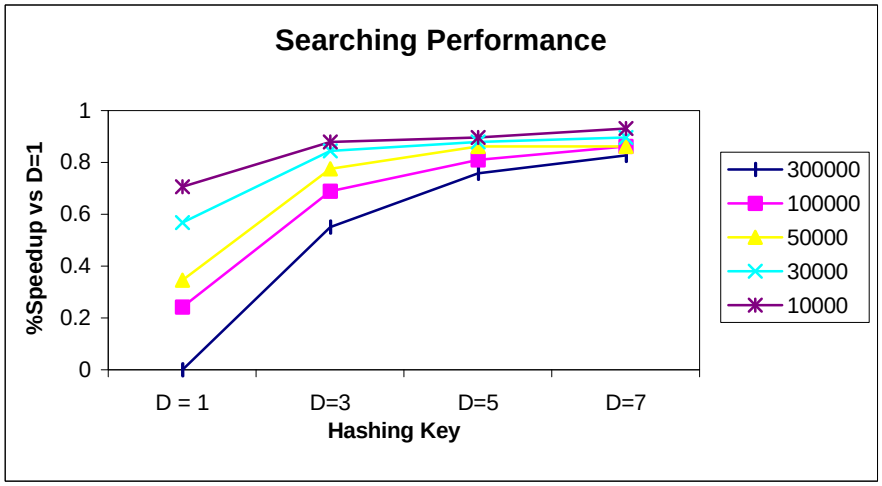

Figure 5.2 Performance of the searching on data buffer

Both of the results indicate that a larger hashing key improves the performance. However, it is found that when the hashing key is bigger than three, the enhancement of the performance is no longer significant. A fast searching ability is helpful to collect the information on a specific IP address and a fast deletion ability is good for performing the sliding window efficiently. Because the shared data part is accessed by many threads, a quick deletion or searching action will not block others to access the data buffer too long.

Fair fast insertion: There is a trade off between the performance of inserting and searching/deletion. A more complicated data structure will be favorable to a better searching/deletion performance. However, the original initialization and insertion will pay for that. Figure 5.3 illustrates the result on insertion performance. The results indicate that a larger hashing key is very beneficial for insertion performance because it reduces the time of searching while inserted, especially when the total number of nodes in the buffer is large. Actually, when the hashing key is equal to 1, it is similar to a linked list. This proves that a tree-like buffer has a better performance than a circular list buffer. Since the insertion time is directly related to the detection response, choosing a hashing key is one of the most important factors.

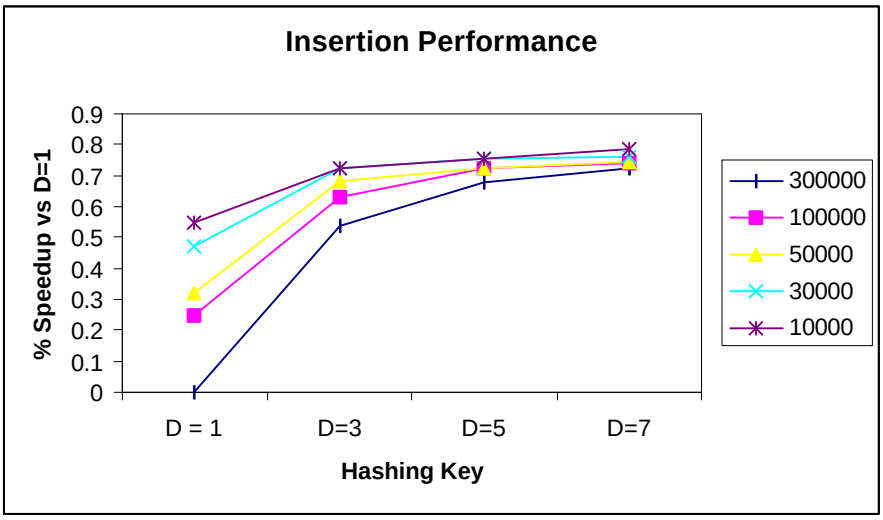

Figure 5.3 Performance of the insertion on data buffer

Memory Utilization: Another trade off in the data buffer is between hashing key selection and memory utilization. Figure 5.4 depicts the memory utilization under different hashing keys and different total amount of data. The data in Figure 5.4 has

been normalized to utilization with D = 1. As the total number of nodes increases in the buffer, the utilization becomes better. However, with the growth of the hashing key, the utilization decreases linearly. For the first phenomenon, a larger number of data increases the chance of data redundancy. Consequentially, each node in buffer can represents several different events with the same source IP address. As for the second one, it is because more hashing buckets occupy more memory space. Therefore, the utilization percentage is lowered in that case.

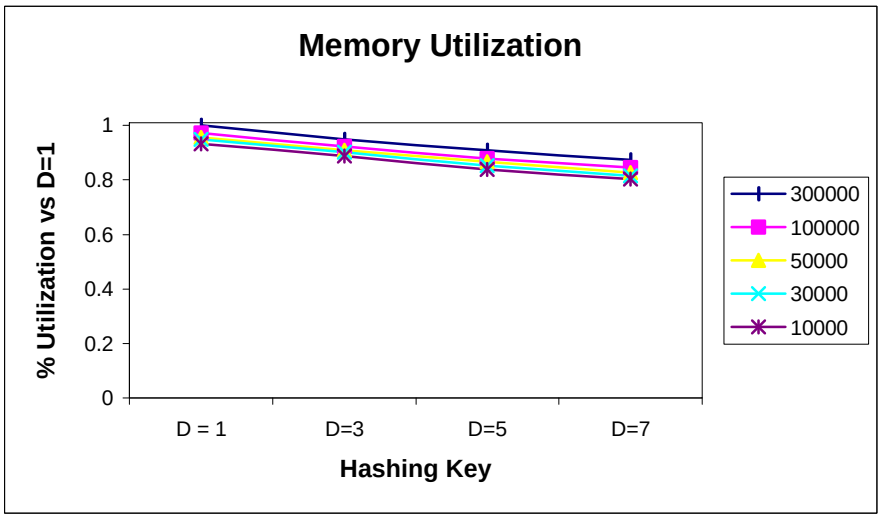

Figure 5.4 Memory Utilization

Based on the testing results, it indicates that the detector detection and response time is in a desired range. Overall detection performance of the tree-like data buffer is better than that of a circular list buffer, though it has a little higher memory usage. When the data has a high redundancy frequency, like in the case of login failure, its memory utilization will be comparable with the circular list buffer. An important decision is choice of a fair hashing key. A range between three and seven will be good enough,

because a bigger one will cause a lower utilization of memory and a smaller one will cause inefficient performance.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

This thesis implements a near real time host-based and event-based intrusion detector. It satisfies the desirable characters of an intrusion detection system. The detector configuration file allows the coordinator easily to reconfigure the detector so that the coordinator can run the detector as needed. The use of the user profile contributes a knowledge system to the detector. After a period of "learning," the detector can obtain more accurate knowledge of local users.

The relatively small size of the data analysis buffer not only ensures a small system overhead but also provides quick detection and response. The aging policy implemented by the sliding window enhances the efficiency of the data analysis buffer and the capability of its continued running. The compressed file makes periodic polling analysis and system recovery possible.

A multi-threaded system allows the detector to perform different tasks concurrently. A distributed detection system is feasible by using a client/server model. Finally, an object-oriented programming technique can more easily extend the current data structures for further development.

6.2 Future Work

Future work can be in following fields:

The basic data structure can be made to be reusable and flexible. Vectors of counters in the data indexing tree can be built in the tree. Each vector has different threshold and alarm function. The key of tree node, hashing function and the depth of the tree can be varied. For the successful login data structure, the key of each entry can be parameterized so that the data structure can be configurable.

The designed detector has good scalability and adaptability. Multi-level analysis based on the data buffer and the compressed file is possible. Any future specific searching request can be conducted as a thread to access the existing data analysis buffer.

Machine learning and artificial intelligence research can enhance the detection on local users' abuse attacks. Accordingly, the user profile could be built more accurately.

REFERENCES

[1] Bill Boni. "Hackers, Crakers, Lawyers and Other Dangers," *Network Security*, 4: 19-20, 2000.

[2] L. Todd Heberlein, Gihan V. Dias, Karl N. Levitt, "A Network Security Monitor," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 296-304, 1990.

[3] B.Harris, R. Hunt, "TCP/IP Security Threats and Attack Method," *Computer Communication*, 22: 885-897, 1999.

[4] CNET News.com Staff, "How a "Denial of Service" Attack Works," http://news.cnet.com/news/0-1007-200-1546362.html, 2000.

[5] CERT Coordination Center, "Denial of Service Attacks," Software Engineering Institute, Carnegie Mellon University, http://www.cert.org/tech_tips/denial_of_service.html, 1999.

[6] Brooke Paul, "DDoS: Internet Weapons of Mass Destruction," http://www.networkcomputing.com/1201/1201f1c1.html, 2001.

[7] Guy Helmer, Johnny Wong, Subhasri Madaka, "Anomalous Intrusion Detection System for Hostile Java Applets," *The Journal of Systems and Software*, 55: 273-286, 2001.

[8] B. Mukherjee, T.L. Heberlein, K.N. Levitt, "Network Intrusion Detection," *IEEE Network*, 8 (3): 26-41, 1994.

[9] Eugene H. Spafford and Diego Zamboni, "Intrusion Detection Using Autonomous Agents," *Computer Networks*, 34(4): 547-570, October 2000.

[10] M. Wood, M. Erlinger, Intrusion Detection Message Exchange Requirments, IETF (Internet Engineering Task Force) model, Internet Security Systems, http://www.ietf.org/internet-drafts/draft-ietf-idwg-requirements-05.txt, 2001.

[11] Herve Debar, Marc Dacier, Andreas Wespi, "Towards a Taxonomy of Intrusion-Detection Systems," *Computer Networks*, 31: 805-822, 1999.

[12] Sandeep Kumar and Eugene H. Spafford, "A Pattern Matching Model for Misuse Intrusion Detection," *Proceedings of the 17th National Computer Security Conference*, pages 11-21, Baltimore, MD, USA, October 1994.

[13] Mandar Raje, "Behavior-based Confinement of Untrusted Applications," Technical Report TRCS99-12, April 1999. Computer Science Department, University of California at Santa Barbara.

[14] Internet Security System, Network- vs. Host-based Intrusion Detection, A Guide to Intrusion Detection Technology, http://secinf.net/info/ids/nvh_ids/, 1998

[15] Aaron Schwartzbard and Anup K. Ghosh, "A Study in the Feasibility of Performing Host-based Anomaly Detection on Windows NT," *2nd International Workshop on Recent Advances in Intrusion Detection--RAID99*.

[16] Yang, Bo, "Design and Implementation of a Generic Port-based and Event-based Detection System," Master of Science Thesis, Computer and Information Sciences and Engineering, University of Florida, Gainesville, 1998.

[17] G. Vigna and R.A. Kemmerer, "NetSTAT: A Network-based Intrusion Detection System," *Journal of Computer Security*, 7(1): 37-71, 1999.

[18] M. Crosbie, E. Spafford, "Defending a Computer System using Autonomous Agents," *Proceedings of the 18th National Information Systems Security Conference*, pages 549-558, Baltimore, MD, October 1995.

[19] Jai Balasubramaniyan, Jose Omar Garcia-Fernandez, E. H. Spafford, and Diego Zamboni, "An Architecture for Intrusion Detection using Autonomous Agents," Coast TR 98-05, COAST Laboratory, Purdue University, West Lafayette, IN, May 1998.

[20] Guy Helmer, Johnny S. K. Wong, Vasant Honavar, and Les Miller, "Intelligent Agents for Intrusion Detection," *Proceedings of the IEEE Information Technology Conference*, pages 121-124, Syracuse, NY, September 1998.

[21] P. Neumann and P Porras, "Experience with EMERALD to Date," *1st USENIX Workshop on Intrusion Detection and Network Monitoring*, pages 73-80, Santa Clara California, April 11-12, 1999.

[22] P. A. Porras and P. G Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," *Proceedings of the 20th National Information Systems Security Conference*, pages 353-356, Baltimore, Maryland, Oct. 7-10 1997. National Institute of Standards and Technology/National Computer Security Center.

[23] 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*, pages 146-161, Oakland, CA, May 1999.

[24] S. Staniford-chen, S. Cheung, R. Crawford, M.Dilger, J.Frank, J.Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle, "GrIDS--A Graph-Based Intrusion Detection System for Large Networks," *Proceedings of the 19th National Information Systems Security Conference*, volume1, pages 361-370, Baltimore, Maryland, October 1996.

[25] S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, J. Rowe, S. Staniford-Chen, R. Yip, and D. Zerkle, "The Design of GrIDS. A Graph-Based Intrusion Detection System," Technical Report CSE-99-2, Computer Science Department, The University of California, Davis, January 1999.

[26] James A. Hoagland, Raju Pandey, and Karl N. Levitt, "Security Policy Specification using a Graphical Approach," Technical Report CSE-98-3, Department of Computer Science, The University of California, Davis, CA, July 1998.

[27] Seetharaman Balasubramanian, "An Architecture for Protection of Network Hosts from Denial of Service Attacks," Master of Science Thesis, Computer and Information Sciences and Engineering, University of Florida, Gainesville, 2000.

[28] CERT Coordination Center, "Configuring and Using Syslogd to Collect Logging Messages on Systems Running Solaris 2.x," Software Engineering Institute, Carnegie Mellon University, http://www.cert.org/security-improvement/implementations/i041.08.html, 2001.

BIOGRAPHICAL SKETCH

Jin Chen was born in Kaifeng, Henan, China. He received the Bachelor of Science and Master of Science degrees from Nankai University, Tianjin, China, in July 1995 and New Mexico Highlands University, Las Vegas, in August 1999 respectively. He will receive his Master of Science degree in computer and information science and engineering from the University of Florida, Gainesville, in August 2001. His research interests include computer network and database.