PERFORMANCE ENHANCEMENT OF SIGNATURE-BASED NETWORK INTRUSION DETECTION SYSTEM

By

Muhammad Tariq Saeed

A thesis submitted to the faculty of Information Security Department, Military College of Signals, National University of Sciences and Technology, Pakistan in partial fulfillment of the requirements for the degree of MS in Information Security

September 2008

ABSTRACT

Exponential increase in number of vulnerabilities, network traffic and bandwidth pose a serious threat to the performance aspects of Intrusion Detection Systems (IDS). Signature-Based IDS operates by comparing packet payloads against attack signatures. The process of signature matching takes up a lot of processing time and thus overwhelms the efficiency of a single Intrusion Detection System In this work; we propose a function-parallel architecture for enhancing the performance of IDS. The proposed architecture outperforms existing approaches of performance enhancement in terms of speed-up and cost. The parallel implementation has been done in java language on a cluster system comprising of 32 nodes. The cluster consists of dual 3.06 GHz, 1 GB RAM control node, 16 HP and 16 SUN 2.2 GHz compute nodes with 4 GB RAM on each node. Control node runs Red Hat Enterprise Linux AS Operating System whereas compute nodes run the WS version of the same OS. All nodes are interconnected using a Gigabit interconnect through HP ProCurve 2848 switch. The results obtained by parallel implementation of our proposed solution have shown 60 percent improvement in speed up on 32 Intrusion Detection Sensors. The approach has shown the potential to be extended and implemented on reconfigurable hardware for developing a cost-effective and scalable solution for future.

To my loving parents

ACKNOWLEDGMENTS

All praise and thanks to Almighty Allah who has showered me with His invaluable blessings throughout my life, giving me strength and spirit to complete this research work.

I wish to express my sincere gratitude to my thesis advisor AVM(R) Dr. Muhammad Shamim Baig for his continued support and guidance during this project. I have made numerous mistakes and have found him understanding and tolerant. It has been a rewarding experience working with him.

I also gratefully acknowledge the help and guidance provided by my guidance committee members Lt. Col. Attiq Ahmad, Lt. Col. Mofassir-ul-Haque and Air Cdre. Tahir Mahmood Khalid. Without their personal supervision, advice and valuable guidance, completion of this research work would not have been possible.

I am especially thankful to Air Cdre. Habeel Ahmed for his continuous support and encouragement during this research work. I also owe very special thanks to Miss Samin Khaliq, Mr. Muhammad Tariq and Sqn. Ldr. Liaqat Ali Khan for their valuable suggestions.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Introduction

Attacks on computer systems have been increased tremendously with increase in number of vulnerabilities. Over 32000 different vulnerabilities have been reported between 2000 and 2007 [1]. These vulnerabilities are exploited by hackers to gain access to computer systems over the internet. Any violation attempt or actual violation to the security policy committed by an internal or external user is called an Intrusion [2]. The set of techniques used to identify intrusions on computer systems is defined as intrusion detection [3]. Intrusion detection and prevention systems play a crucial role in the security of systems and networks. Intrusion Detection Systems (IDS) are classified mainly on the basis of analysis techniques as Signature-based IDS [4] and Anomaly-based IDS [5]. Security policy in signature-based IDS is implemented by rules [6]. In signature-based IDS, the rules are written in a rule description language which provides different constructs for describing an attack signature. Signature-based IDS needs to check thousands of known attack signatures for every packet. Signature matching of this type imposes significant delays on IDS due to size of network traffic and complexity of policies. Therefore IDS cannot keep pace with the speed of modern networks, resulting in loss of information and performance chokepoints [7]. The existing work to improve the performance focuses on the use of hardware techniques, improvement in software and algorithm and parallelization. This work explores different parallel processing techniques for improving the performance of Signature-Based Network Intrusion Detection System and proposes a function parallel architecture for performance enhancement of Signature-based IDS. The existing work in this regard is based on data parallel approaches and is of very preliminary nature. The data parallel approaches are characterized by several drawbacks

discussed in chapter 4. The proposed architecture employs the concept of function-parallelism and divides overall IDS policy across an array of Intrusion detection sensors. The parallel implementation has been done on a cluster system and experimental results show a comparison of proposed solution with existing work. The proposed solution outperforms existing data parallel approaches and show a significant speed-up which increases linearly with addition in the number of Intrusion Detection sensors. The proposed approach has the potential to be extended and implemented on reconfigurable hardware for developing a cost-effective and scalable solution for the future.

1.2 Problem Statement

Security policy in signature based IDS is implemented by rules. The policy database contains thousands of rules. These rules contain signatures against different attacks. A signature-based NIDS evaluates packet payloads against this huge policy database. Thus making rule matching the most expensive part of signature based IDS in terms of processing and memory resources. Rule matching of this type imposes significant delays on traffic due to complexity and size of polices. Therefore any rule matching processor based on a single intrusion detection component cannot scale over certain thresholds.

1.3 Overall Objective

The motivation behind this work is to explore and develop different techniques used for enhancing the performance of signature-based NIDS and design an architecture that processes network traffic in less time. The architecture will be evaluated by using benchmark data and performance comparison will be carried out with existing data parallel approach. Possibility of expanding the research to Intrusion Prevention Systems will also be studied

1.4 Thesis Organization

The research thesis is organized as given. Chapter 2 discusses the challenges faced by Intrusion Detection Technology in detail from the performance prospective. An in depth review of Intrusion Detection Systems is given in Chapter 3. Different techniques for parallelization of IDS are discussed in Chapter 4. The proposed architecture for performance enhancement is presented in Chapter 5. Issues related to Concurrency and proposed multi-threaded model are discussed in Chapter 6. Implementation of the proposed architecture along with experimental results is presented in chapter 7. Finally conclusion and Future work is discussed in Chapter 8.

1.5 Summary

This chapter has provided a comprehensive introduction to the addressed research work. A brief introduction, problem statement and overall objective of the research work have been discussed. Towards the end of the chapter, overall thesis organization is provided to help the reader in a thorough manner.

Chapter 2

RESEARCH MOTIVATION

2.1 Introduction

The real motivation behind this dissertation is, the continuously pressing performance requirements that modern networks impose on security devices. This chapter discusses some challenges that have become a real matter of concern for performance enhancement of IDS.

2.2 Challenges

Network intrusion detection System (NIDS) detect and investigate security threats on an organization's network. It complements other security devices, such as firewalls, by providing information about the frequency and nature of attacks. The performance of a network intrusion detection system is characterized by the probability that an attack is detected and the number of false alerts. However, the system's ability to process traffic at maximum rate offered by the network is equally important with minimal packet loss. Significant packet loss degrades the overall effectiveness of the system and can leave a number of attacks undetected. A high performance sensor not only improves the quality of detection by reducing the number of false alerts, but also processes packets at a higher rate. Network Intrusion Detection Systems cannot rely on flow control mechanisms and acknowledgments to control the flow of data. Instead, the NIDS must be able to process packets at the maximum rate offered by the network.

2.2.1 Network Speed

The issue of faster data communication is becoming more and more critical with relentless increase in world's internet traffic. Researchers are focusing on improved

techniques to increase data transmission rates in order to cope with phenomenal surge in data traffic, as internet population exceeds a billion users. German and Japanese scientists recently collaborated to achieve just such a quantum leap in obliterating the world record for data transmission. By transmitting a data signal at 2.56 terabits per second over a 160-kilometer link, the researchers broke the old record of 1.28 terabits per second held by a Japanese group. By comparison, the fastest high-speed links currently carry data at a maximum 40 Gbit/s, or around 50 times slower. The researcher assumes the transmission capacity on the large transoceanic traffic links will need to increase to between 50 and 100 terabits per second in ten to 20 years. This kind of capacity will only be feasible with the new high-performance systems. This phenomenal surge of data at very high speed poses a serious threat to the efficiency of Intrusion Detection Systems. The amount of data available within a single host can make the intrusion detection problem computationally very intensive. In the case of network-based intrusion detection, the traffic going through a gateway, for instance, can be prohibitively huge making it practically impossible for a security scanner to check every single packet without dropping some of them. Packet dropping severely damages the effectiveness of this type of protection tools. Similarly, audit facilities can generate very large log files which limit the likelihood of performing real-time processing, which is not as desirable as real-time detection.

2.2.2 Vulnerabilities

Vulnerability can be defined as a coding bug, configuration error, or design flaw that can be exploited to compromise the security of a system [8]. One of the clearest trends in information security is the fact that more and more vulnerabilities are discovered everyday and, consequently, more attacks that exploit them also appear on the scene. New threats appear every day challenging the quality and configuration of all kinds of

platforms ranging from web farms to even wireless telephony. Viruses and worms such as Sasser, Nimda, Code Red, Blaster, and Slammer have hit thousands of computers resulting in millions of people losing a considerable number of effective working hours. When a popular website is compromised by hackers or when a denial of service attack is underway, the reputation of entire institutions may be hurt and the consequences may cause a considerable loss of market share. A system that has not been found to be vulnerable today could be brought down tomorrow. The purpose of the release of vulnerability information is to make users and administrators aware of the existing threats. However, this information is also used by attackers that devise new techniques to abuse systems. The figure 2.1 shows number of vulnerabilities reported each year by CERT [1]. A significant rise in the number of vulnerabilities can be observed with the passage of time.

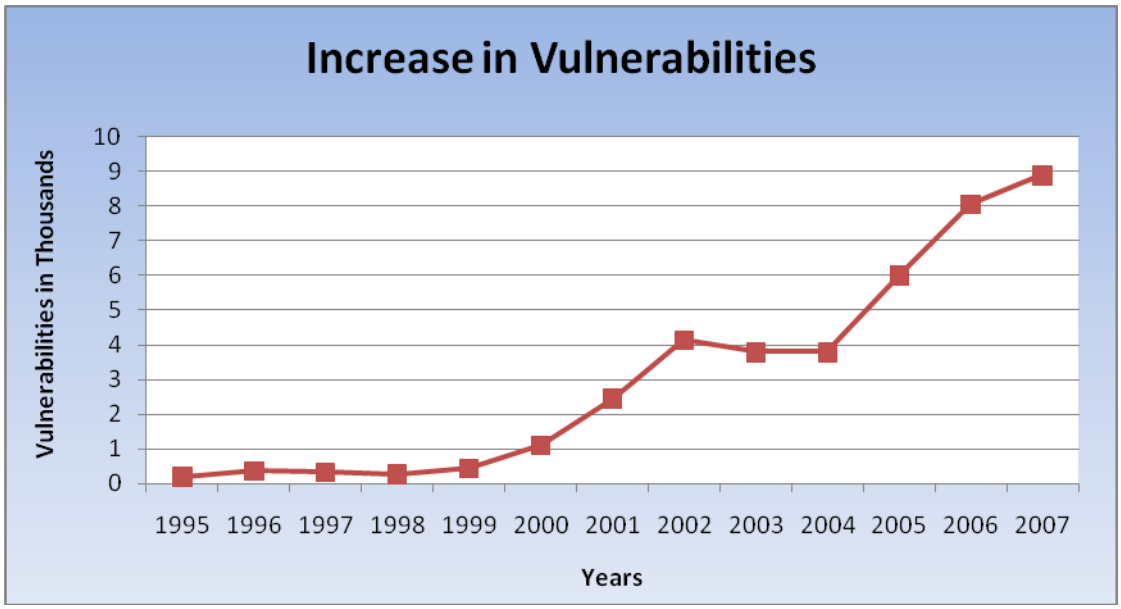

Figure2. 1: Increase in vulnerabilities

2.2.3 Rise in Signature Database

The increase in vulnerabilities has given birth to more security incidents and compromise of millions of resources on internet. The average annual loss reported in the year 2007

survey shot up to $350,424 from $168,000 in 2006 [9]. Percentage of organizations reporting computer intrusions to law enforcement continued upwards after reversing a multi-year decline over the past two years, standing now at 29 percent as compared to 25 percent in last year's report [9]. In Signature-Based IDS, these vulnerabilities are fixed by designing new signatures and adding them to signature database. As a result, numbers of signature are continuously increasing.

In June, 2003 there were only 1500 signatures in default Snort Rule Set whereas in June, 2006 it reached up to 5000 default signature and it is approaching 10,000 now. This increase in signature database means that the Packet payloads have to be evaluated against more signatures, putting too much load on IDS to process huge amount of traffic against thousands of signatures in a fraction of time. Therefore with increase in network speed and signature database, there is pressing requirement for developing performance enhancement methodologies.

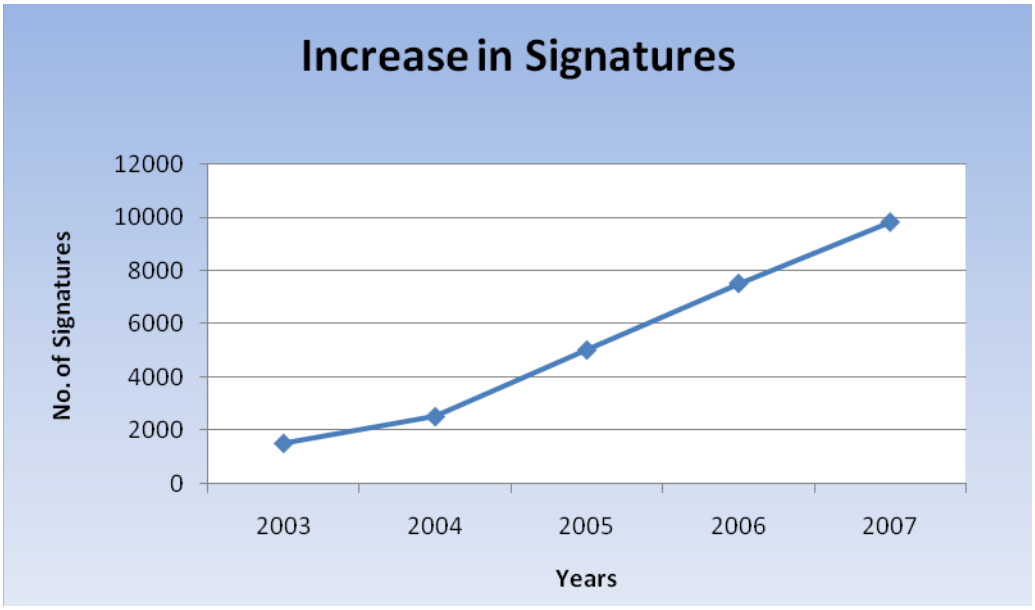

Figure2. 2: Increase in Signature Database

The increase in network speed poses a serious threat to the performance aspects of Intrusion Detection Systems. The process of signature matching is computationally very

intensive and IDS has to process huge amount of traffic in a fraction of time. The performance constraint makes it impossible for IDS to check every packet and it starts dropping the packet. Intrusion Detection Systems also serve as a forensic tool because the information regarding malicious packets is logged. With increase in the amount of network speed, the drop rate also increases and IDS may not catch evidence needed to trace an attacker.

Intrusion Prevention systems are placed inline and the overwhelming load of traffic results in a more devastating situation where rest of the network behind an IPS will experience a denial of service situation. The IPS needs to process the data rapidly and forward traffic to its intended destination.

2.3 Summary

This chapter has presented an overview of the challenges that modern IDS face in the form of growth in vulnerabilities and rise in signature database. Due to increase in the number of vulnerabilities reported per year, signature database of IDS is continuously increasing. As a result, IDS has to evaluate millions of packets against thousands of attacks signatures. Signature matching of this type is computationally very intensive and imposes significant delays on IDS resulting in loss of information.

Chapter 3

INTRUSION DETECTION SYSTEMS

3.1 Introduction

An Intrusion Detection System (IDS) detects unwanted attempts of accessing, manipulating or disabling of computer systems. These attempts are usually made by hackers, malicious programs, or disgruntled employees. Intrusion Detection Systems are used to detect several types of malicious behaviors that can compromise the security and trust of a computer system. It detects network attacks against vulnerable services, data driven attacks on applications, host based attacks such as privilege escalation, unauthorized logins and access to sensitive files.

3.2 Classification of IDS

Currently there are two basic techniques for intrusion detection. The first technique, called *anomaly detection*, defines a normality profile or normal working of systems, and then it detects anything that deviates from this normal activity [8]. It relies on being able to define desired form or behavior of the system and then to distinguish between that and undesired or anomalous behavior. The boundary between acceptable and anomalous form of stored code and data is precisely definable. One bit of difference indicates a problem. The boundary between acceptable and anomalous behavior is much more difficult to define. The second approach, called misuse detection [8], involves characterizing known ways to penetrate a system. Each one is usually described as a pattern. The misuse detection system monitors for explicit patterns. The pattern may be a static bit string, for example a specific virus bit string insertion. Alternatively, the pattern may describe a suspect set or sequence of actions. Patterns take a variety of forms as will be illustrated later. Intrusion detection systems have been built to explore both

approaches – anomaly detection and misuse detection – for the past 15 to 20 years. In some cases, the two kinds of detection are combined in a complementary way in a single system. There is a consensus in the community that both approaches continue to have value. The techniques for single systems have been adapted and scaled to address intrusion in distributed systems and in networks. Efficiency and system control have improved. User interfaces have improved, especially those for specifying new misuse patterns and for interaction with the system security administrator. Essentially all the intrusion detection implementations that will be discussed are extensions of operating systems. They use operating system notions of events, and operating system data collection, particularly audit records, as their base

3.2.1 Anomaly-Based Detection

Anomaly detection is also known as profile-based intrusion detection and statistical intrusion detection. In this technique, a definition of “normality” is created and any abnormal traffic that deviates from that normality profile is declared as intrusive [7].

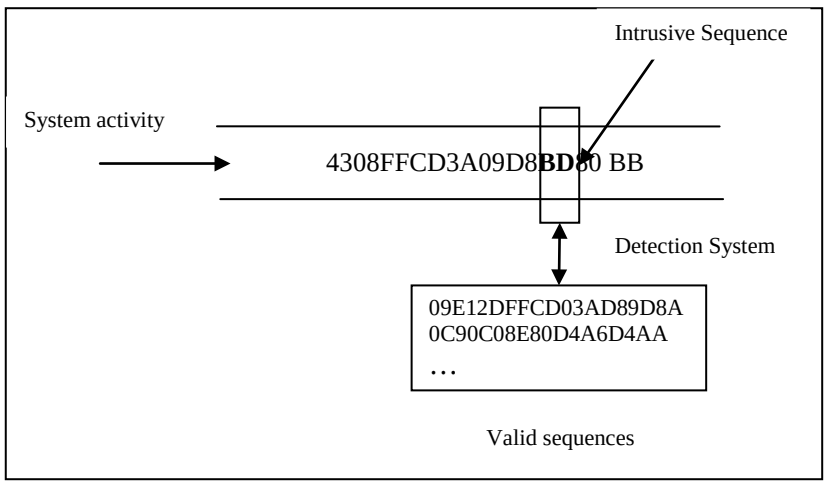

Figure3. 1: Anomaly based Detection

This approach assumes that intrusion constitutes of irregular events. The normal activity of users and applications is monitored and their behavior profiles are created. These profiles are built through training or heuristics [10-15].

Anomaly-Based IDS then monitors the behaviors of users and applications as shown in figure 3.1. Any activity that diverges from the normal profile is considered as intrusive. Differentiating an abnormal activity from a normal one is very difficult task which requires compliance with protocols, understanding the internals of an application, expertise level of users, their preferences, date and time.

A **false positive** is the classification of genuine activity as intrusive whereas a **false negative** is the classification of illegal activity as benevolent – the latter being a much more serious problem as it represents failure to identify potentially harmful events.

Anomaly based intrusion detection systems are difficult to implement as they require not only compliance with protocols and understanding the inner-working of an application but also expertise level of users, their preferences, and the date and time.[16]. Construction and tuning of behavioral models required for anomaly based intrusion detection systems are computationally very intensive.

3.2.2 Signature-Based Detection

Signature-based Detection is also known as misuse detection or pattern-based detection. In this technique, patterns of abnormal activity are defined as shown in figure 3.2. IDS searches to find the occurrence of those patterns in monitored data stream. If an attack pattern is found in data stream, it is the sign of an intrusion [17]. This pattern or fingerprint called signature and it is used to identify and represent an attack [18]. Signature-based IDS will monitor packets on the network and match up to them against a database of signatures or attributes from known malicious threats. Monitoring programs observe different data streams present on the system and match them against a bank of attack signatures [19-21]. This is similar to the way most antivirus software detects malware. The antivirus searches for malicious signatures inside executables of

applications while the signature-based IDS perform the same operation at the network layer.

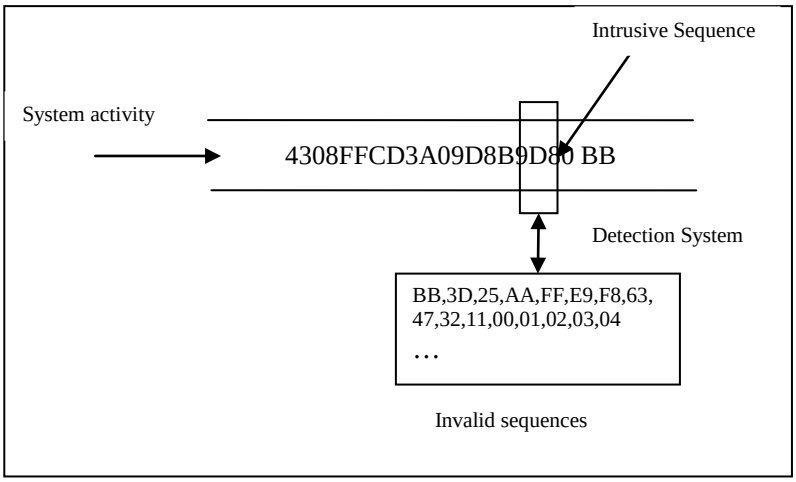

Figure3.2: Signature based Detection

3.3 Functional Decomposition of IDS

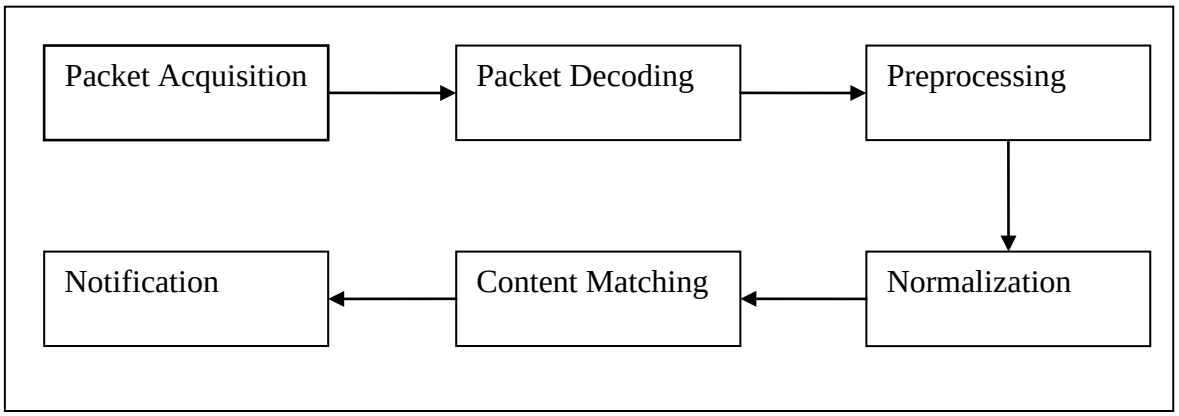

Figure3. 3: Functional decomposition of IDS

3.3.1 Packet Acquisition

After initialization, Signature-based IDS enters into its packet processing function. For passive sniffing (and file read-back) modes this function is *InterfaceThread* in *src/snort.c*. This function utilizes the libpcap library [26] for retrieving packets from the

network device or a trace file. Libpcap is a cross-platform library that provides an API for receiving packets directly from the network. Libpcap provides basic information about each packet, including the time at which the packet was captured from the network, length of the packet on the wire and link layer type of the interface on which the packet was captured.

3.3.2 Packet Decoding

Once the packets are acquired, it passes it into the packet decoder. Exactly where the packet enters the decoder depends on the link layer from which it is being read. IDS support a number of link layers from pcap: Ethernet, 802.11, Token Ring, FDDI, Cisco HDLC, SLIP, PPP, and OpenBSD's PF. It also supports the link layers specific to the APIs used for inline mode. Above the link layer, it supports decoding several other protocols, including IP, Internet Control Message Protocol (ICMP), TCP, and User Datagram Protocol (UDP). Although "decoders" are available for many other protocols (such as Internetwork Packet Exchange [IPX]) within Snort, many of them are just stubs that increment counters to indicate how many packets have been seen. In order to extend Snort to really support these protocols, work should begin in the decoder. You can find the implementation of the decoder in *src/decode.c*. Regardless of which link layer is being used, all of the decoders work in the same general fashion. For the particular layer being decoded, pointers in the packet structure are set to point to various parts of the packet. Based on the decoded information, it calls into appropriate higher-layer decoders until no more decoders are available.

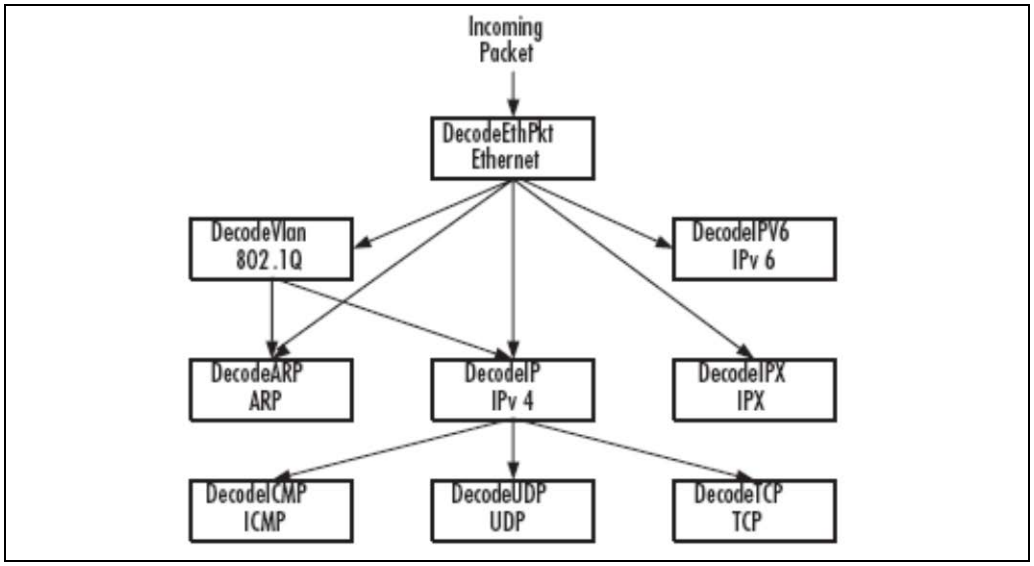

Figure3. 4: Decoders Chain

3.3.3 Preprocessing

After the packet has been decoded, it is passed into the preprocessors. The preprocessors provide a variety of functions, from protocol normalization, to statistics based detection, to non rule-based detection. There is variety of tasks that preprocessors can perform .A preprocessor is code that is compiled into the IDS engine upon build in order to normalize traffic and/or examine the traffic for attacks in a fashion beyond what can be done in normal rules. Although that might seem like an overly simplistic explanation for what these complex pieces of Snort do, it's important to realize their contribution to the overall whole of the intrusion detection system (IDS).

3.3.3.1 Protocol Decoders

Before the preprocessors even see traffic, all traffic must pass through the protocol decoders. Figure 3.5 shows layout of link layer decoders.

3.3.3.2 IP Defragmentation

The IP defragmentation preprocessor(s) reassembles fragmented packets**.**

3.3.3.3 Stateful Inspection

The stream4 preprocessor verifies that packets are part of an established session

3.3.3.4 Stream Reassembly

The stream4_reassemble preprocessor reassembles TCP streams into a “pseudo-packet” for contextual analysis.

3.3.3.5 Application Layer Processors

This step contains a collection of preprocessors (with more coming all the time!) that normalizes complex protocols, and, if needed, generates events on incorrect implementations of the protocol, or an attempted misuse of the protocol.

3.3.3.6 Detection Engine

Finally, after all these steps have taken place, then the packets are passed through to the Snort detection engine.

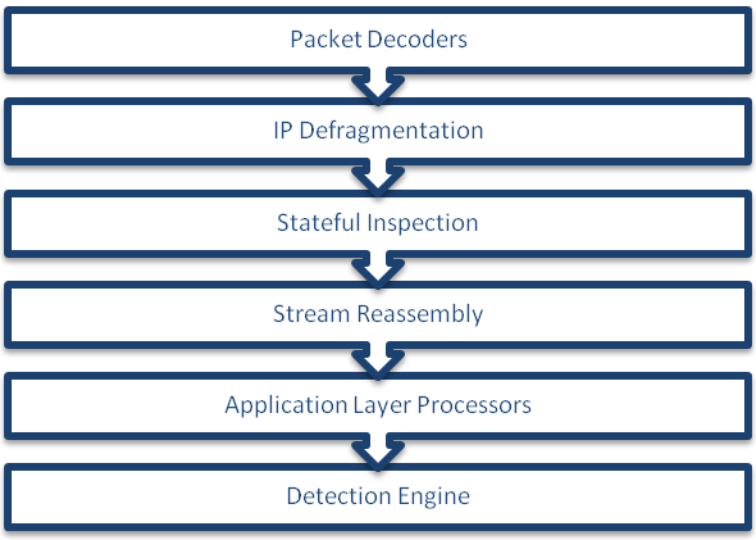

Figure3. 5: Levels of link-layer decoders

3.3.4 Content Matching

Signature-based intrusion detection systems rely on content matching for detecting the anomalous traffic. A great deal of academic research focuses on analysis and design of anomaly based detection techniques but majority of the commercial intrusion detection systems are signature-based due to its high detection accuracy.

During the content matching phase, IDS examines the contents of a network packet and checks for the presence of attack signatures. Each signature is represented by a rule that describes a known intrusion threat. If a packet is found to contain a given signature, then the action specified by a rule associated with the signature is taken. This action triggers an alert and logs the specious packet. Snort uses a simple, lightweight rules description language that is flexible and quite powerful. There are a number of simple guidelines to remember when developing Snort rules. Most Snort rules are written in a single line. This was required in versions prior to 1.8. In current versions of Snort, rules may span multiple lines by adding a backslash \ to the end of the line. Snort rules are divided into two logical sections, the rule header and the rule options. The rule header contains the rule's action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information. The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.

```
alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|";
msg:"mountd access";)
```

Figure3. 6: Header and body part of Rule

The text up to the first parenthesis in figure 3.6 is the rule header and the section enclosed in parenthesis contains the rule options. The words before the colons in the rule options section are called option keywords.

All of the elements in that make up a rule must be true for the indicated rule action to be taken. When taken together, the elements can be considered to form a logical AND statement. At the same time, the various rules in a Snort rules library file can be considered to form a large logical OR statement.

3.3.5 Notification

After the content matching phase the intrusion detection system issues a notification of the events that are produced to inform network administrators about the incident. IDS also store this information in the form of logs that are later used for tracing the perpetrators and detailed forensic analysis.

3.4 Limitations

Intrusion detection technology still suffers from some serious limitations that need to be worked out in order to create effective products. Some of the main challenges relevant to the topic are described next.

3.4.1 False Alarms

Many intrusion detection tools suffer from detection inaccuracy. The high rate of false positives and false negatives that intrusion detection tools generate does not help system administrators simplify their daily tasks. On the contrary, users may eventually opt for disabling this software rather than having to interpret the stream of alarms reported by the security system that may, in fact, not be reflecting the truth. This is a major limitation of intrusion detection technologies and is the priority for multiple data fusion and model-based projects.

3.4.2 Switched Networks

Modern networks utilize more intelligent devices to route packets from one location into the other. Many intrusion detection prototypes were based on the ability to configure a network device in promiscuous mode in a way that the monitoring mechanism could easily see and analyze all Ethernet packets. This allowed the system to correlate packets addressed to different hosts connected to the same segment, and also to protect several hosts at the same time. With modern network technology, this approach may no longer be as successful.

3.4.3 Encryption

Encrypted data cannot be easily interpreted unless a decryption key is available Information technology has witnessed the development of cryptographic protocols and tools that protects the privacy of data. File transfers, interactive sessions, and email can all be protected with encryption. Although this secures the information they handle, it renders further security analysis impossible – necessary in the case they transport illicit material. As more companies become security-conscious the use of encryption will increase. Intrusion detection tools are currently inadequate to deal with this problem as they are blind to encrypted communications.

3.4.4 Performance

Data filtering and screening is a computationally demanding task that is central to intrusion detection. Providing a system with real-time monitoring capabilities implies good search and pattern matching algorithms as well as efficient buffering of network traffic, audit trails, etc. which guarantees no data piece goes through without being inspected. Given the detailed checking needed by intrusion detection, a lot of resources are allocated and consumed and the performance of the hosting system ends up being

impacted. As a consequence, heavy intrusion detection software and business processes cannot cohabit and the former is sometimes disabled to give full privileges to priority services.

3.4.5 Security of Intrusion Detection Technology

Security tools are expected to be secure themselves – otherwise they could not be trusted and their results would be worthless. As software products, intrusion detection systems face the same security challenges other applications do (e.g., poor implementation and weak design). In fact, not much attention has been put into the secure design of this type of software. Only a few systems can be considered moderately secure but the rest is prone to direct or evasion attacks. This is an aspect that cannot be ignored when designing protection systems.

3.4.5 Summary

This chapter has presented an introduction to intrusion detection technology, it functional decomposition and limitations. Intrusion Detection System (IDS) can be classified on the basis of analysis technique into two types; Signature-based detection and Anomaly-based detection. Signature-based IDS are the ones currently dominating the market with high detection accuracy. An IDS performs a series of operation on packet data before it is evaluated against attack signatures. Signature-based IDS also suffer from some serious limitation such as performance drawbacks, false alarms and security of intrusion detection technology.

Chapter 4

PERFORMANCE ENHANCEMENT

4.1 Introduction

Performance Enhancements of Intrusion Detection Systems for high-speed links has been the focus of much debate and research in the recent past. The most common solutions to achieve high performance rely on hardware approaches [24-26], improvement in software and algorithm and parallelization. A common point of view is that high-speed network based Intrusion Detection is not practical by using a single sensor because of the technical difficulties encountered in keeping pace with increasing network speed and enormous surge of internet data. Analysis of network traffic on high-speed links is computationally intensive problem requiring lot of processing time.

The performance enhancement of Intrusion detection techniques is a fertile area of research where different approaches are being used to increase their efficiency. The NIDSs' weak process ability is mainly because that the analysis of network packets needs much computing. So it seems that the problem could be resolved if the processors' speed is increased. Unfortunately, the speed of networks increases faster than the speed of processors. It's impossible to keep up with the speed of networks by just increase the CPU's speed of NIDSs. This chapter presents an overview of existing work in this field and its analysis.

Parallelism is one technique that may be used to help reduce IDS processing time. Parallelization can occur at different levels of the intrusion detection system. For example the entire IDS can be duplicated and arriving packets can be distributed to various systems. The policy of IDS can be duplicated and network traffic is divided by

using a traffic splitter. Likewise same data can be sent to each node and it is evaluated against a subset of the overall IDS policy.

4.2 Data Parallel Approach

Data parallelization is a form of parallelization which is also known as loop level parallelism. It focuses on distributing the data across multiple computing nodes. In data parallelism, each processor performs same computations on different sets of data. For example, if the code is being run on dual-processor system having CPU-A and CPU-B and we have to perform operation on data D, it is possible to tell CPU-A to do that task on one part of data and CPU-B on other part of data. Both operations are performed simultaneously which reduces the execution time of serial program. In data parallel implementation of matrix addition, CPU-A could add elements from top half of the matrices and CPU-B could add all elements from bottom half of matrices. The two processors work in parallel and the job of matrix addition would take one half of the time of performing the same operation on single processor.

There are not many parallel architectures available for improving the performance of IDS and existing work towards parallelization of IDS is based on the concept of data parallelism [9]. In data parallel approach, each node contains the complete IDS policy. A traffic splitter is then used to distribute packets to each node. The primary goal of the traffic splitter is to perform load balancing across all nodes. However, because intrusion detection systems perform stateful analysis a simple round-robin technique for distributing packets is inadequate. It is necessary that the traffic splitter maintain session integrity due to factors like packet reassembly.

The first implementation was given by Christopher Kruegel, in which the traffic is divided into slices and each slice is processed by one or more Intrusion Detection sensors

[24]. The partitioning is done so that a single slice contains all the evidence necessary to detect a specific attack. The architecture of the system is showed in Fig. 4.1. The scatterer only scatters frames in a round-robin fashion to guarantee high speed. The task of the slicers is to route the frames they receive to the sensors that may need them to detect an attack. The reassemblers are responsible to reassemble the possibly disordered frames. The system's throughput nearly reaches 200 Mbps in their experiment. The system consists of a network tap, set of m different slicers, set of n stream assemblers and set of P different Intrusion Detection Sensors. The network tap monitors traffic speed on high-speed monitored link and extracts sequence of frames on the wire during a specific time period. These frames are passed to the scatterer who partitions these frames and divides them into different subsequences.

The splitting of the traffic is done on scatterer and each portion is directed to a different slicer. The traffic slicers route frames to sensors that may need them to detect an attack. The task of frame routing is performed is not performed on the scatterer because frame routing is complex task and will become a bottle neck of scatterer in particular and overall architecture in general. The traffic slicers are connected to a switch port and send frames to a designated stream reassembler through a predefined outgoing channel of the switch. Each channel is also associated with some intrusion detection sensors. The job of reassemblers is to make sure that packets appear on the channel in same order as they appeared on high-speed link. The paper mainly discusses how to divide the network traffic to avoid losing the evidence for intrusion detection. But the author does not present a feasible method to equally divide the traffic although he shows that the load balancing can be done by dynamically change the slices' filters. In the experiment, the traffic is statically divided according to the address range. The architecture uses a static configuration of slicers based upon an offline configuration that is loaded on startup. The

static approach suffers from a major drawback as large percentage of network packets could be forwarded to a single channel resulting in a performance overhead on that sensor. Besides, the system is also complex, which needs many devices and has to reassemble the disordered frames.

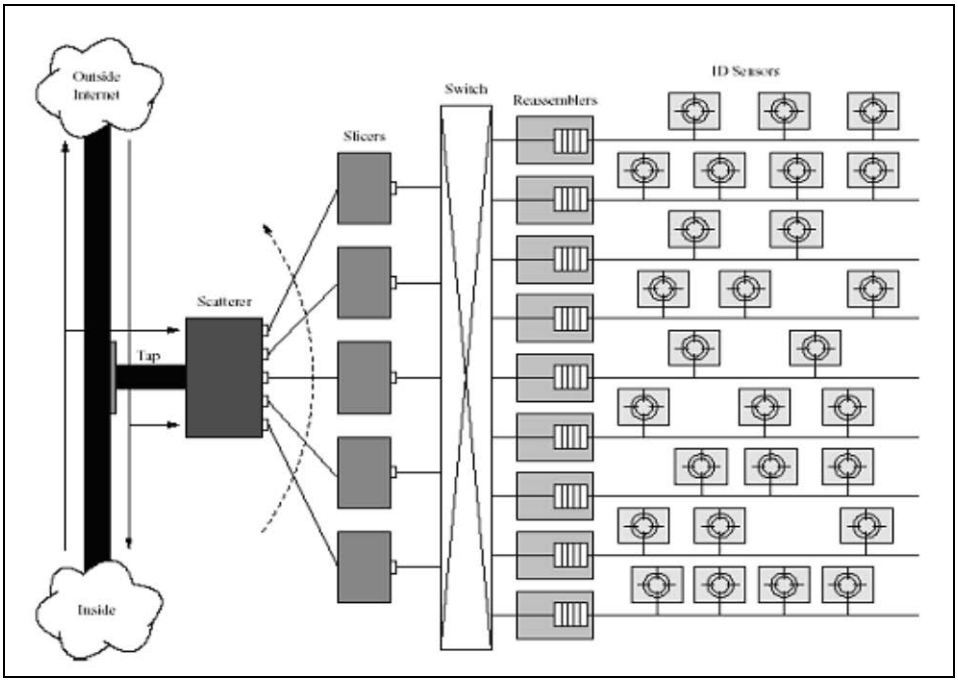

Figure4. 1: High-level architecture of the high-speed intrusion detection system.

Charitakis et al examine a splitter's architecture in their paper [27] and two methods are used to improve system's performance. First method is based on the concept of early filtering, where a portion of packets is processed on the splitter instead of intrusion detection sensors as shown in the figure 4.2. The cost of header preprocessing is significantly cheaper than computationally intensive payload analysis. The NIDS rule set is analyzed and rules that do not require content matching are extracted. It has been observed that a very small portion of the default rule set is only subject to header analysis.

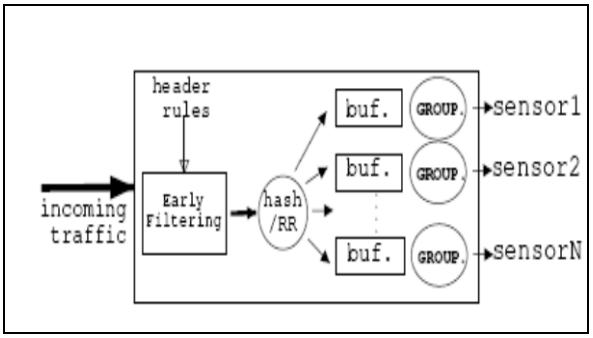

Figure4. 2: Active IDS splitter architecture

These rule sets are referred as Early Filtering rule sets. When a packet arrives at a sensor, it is first evaluated against EF rule set. If no rule is matched and packet contains no payload then the packet is discarded. However if the packet contains payload and no rule is matched, it is retained for payload analysis and forwarded to the sensor. The second method uses locality buffering technique where the splitter records packets in buffers as shown in figure 4.3. The use of locality buffers improves memory access locality on the sensors.

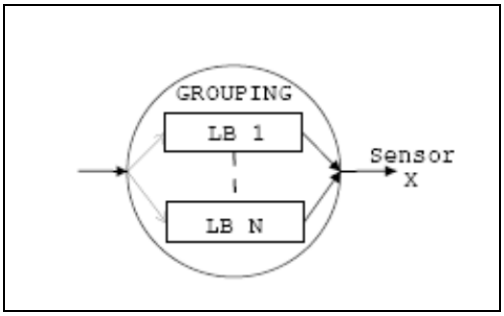

Figure4. 3: Packet grouping using locality buffers

The experiment shows that these methods do improve the performance. However, since early filtering and locality buffering may cause some packets been dropped or reordered, it is impossible for the sensors to do state analysis which is important to improve the detection accuracy. In addition, the author splits the traffic by simply hashing on flow identifiers and does not discuss the problem of load balancing in the paper. The other

interesting parallel architecture presented in [28] is characterized by a custom hardware load balancer that feeds conventional NIDS sensors. To ensure that every packet belonging to a certain flow is analyzed by the same sensor, the hardware load balancer calculates a set of hashing functions on different fields of the packet as shown in figure 4.4 and 4.5.

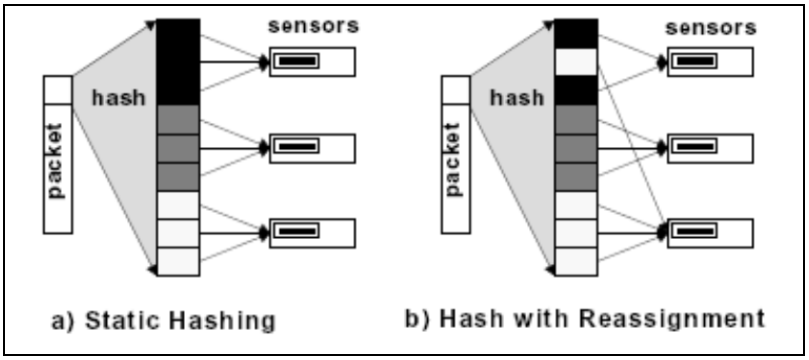

Figure4. 4: Types of hashing

The destination NIDS sensor is selected on the basis of the resulting hashes. However, although the balancer may dynamically adapt to the traffic dispatching rules, the redirection of already established connections to a different sensor makes it impossible to perform a stateful analysis.

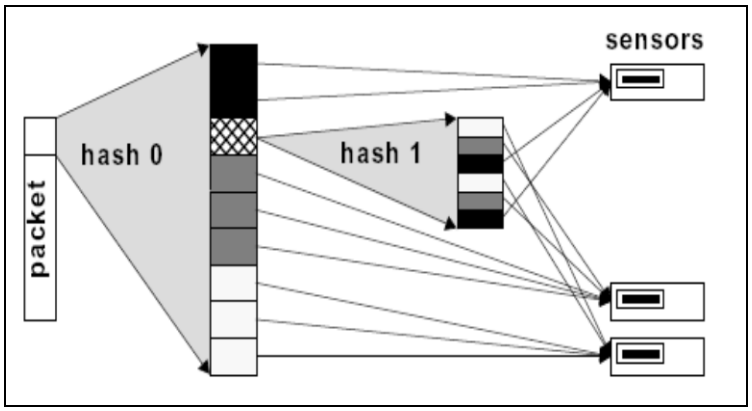

Figure4. 5: Multiple levels of hashing

A serialization of the internal NIDS state was proposed in [29]. This is an important contribution because a serialized representation can be propagated to other NIDS sensors

to achieve better coordination in distributed NIDS architectures. However, this paper does not consider state serialization as a mean to provide Stateful and dynamic load balancing of established connections among different NIDS sensors. Figure 4.6 shows the integration of this concept in architecture of Bro [31].

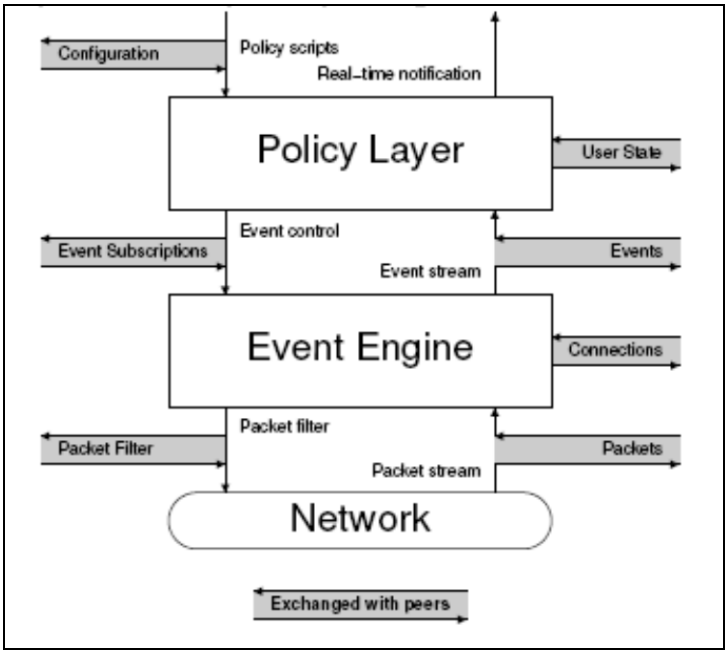

Figure4. 6: Integrating independent state into Bro.

4.3 Function Parallel Approach

Function parallelism is also known as task parallelism and control parallelism**.** It is a form of parallelization of computer code across multiple processors in. Task parallelism focusses on distributing execution processes or threads across different parallel computing nodes. In a multiprocessor system, task parallelism is achieved when each processor executes a different thread or process on the same or different data. The threads may execute the same or different code. In the general case, different execution threads communicate with one another as they work. Communication takes place usually to pass data from one thread to the next as part of a workflow. If we are running code on

a dual processor system in a parallel environment and we wish to do tasks "A" and "B", it is possible to tell CPU-A to do task A and CPU-B to do task B simultaneously and reduce the runtime of the execution.

Function parallelism involves dividing the IDS policy which consists of a set of rule groups each containing some number of rules across nodes. These rules are placed into rule groups based on their source and destination port numbers. For example, rules associated with web traffic are usually placed in the port 80 rule group. One method is to take rules from each rule group and spread them evenly across all nodes. In this case a traffic duplicator is then used to duplicate each incoming packet so that all packets are seen by all nodes. Using this method, which each packet must be examined by each node because each node contains a fraction of the rules which are applicable to all packets.

4.4 Summary

This chapter has presented an overview of performance enhancing techniques, their strengths and weaknesses. The most common solutions to achieve high performance for intrusion detection rely on the use of hardware, improvements in software and use of parallel computing techniques. Application of parallel computing in the domain of intrusion detection seems promising due to high scalability and low cost. Data parallel methods focus on dividing network traffic through splitter while function parallel methods divide policy database of signature-based intrusion detection system across an array of nodes.

Chapter 5

PROPOSED FUNCTION PARALLEL ARCHITECTURE

5.1 Introduction

This chapter presents our proposed architecture in detail, also differentiating it from existing work. The data parallel approach for performance enhancement is characterized by several drawbacks discussed in Chapter 2. The proposed architecture employs the concept of function parallelism for improving the performance of Signature-Based Network Intrusion Detection System. Instead of dividing the data, the security policy of IDS is divided across different nodes. This security policy of IDS consists of different rules where each rule contains a signature and the action against a known attack. Each node in this architecture receives same data packets and compares them against different rule sets.

5.2 Design Goals

The proposed architecture considers following design goals

5.2.1 Uniform Division

The security policy of IDS is divided across multiple sensors such that the workload distribution on each sensor is uniform and performance is maximized. This requirement is the key to guarantee system's performance. If one slice is much larger than others, the intrusion detection sensor that processes the particular slice will become a bottleneck of the system and waste other sensors abilities of processing traffic.

5.2.2 Efficiency

The load-balancing algorithm should be simple and efficient enough to operate at high speed networks. This requirement assures that each sensor can detect attacks without any interaction so that the system's complexity is greatly reduced.

5.2.3 Adaptability

The architecture should be dynamically adaptable to varying traffic loads and compositions. This requirement is also for the performance. It is obvious that a simple and efficient algorithm is easy to archive a high throughput. Unfortunately, in practice, it is very hard to satisfy all of these requirements. For example, the round-robin partition algorithm, which sends the packets to the sensors in turn, completely satisfies requirement 1 and requirement 3. However, because the algorithm does not consider any character of the packets, it is very possible that several packets relevant to the same attack are sent to different sensors. As a result, no sensor has enough information to detect the attack. The round-robin algorithm does not satisfy requirement 2. On the other hand, to satisfy requirement 2, it is difficult to partition the traffic equally and keep the algorithm simple and efficient.

After studying the network attacks, we design a simple architecture that satisfies requirement 2 and guarantee that the slices have the nearly same sizes. There are two kinds of network attacks. The attacks in the first category are those that can be detected after inspecting all the packets belonging to one TCP/UDP connection. Most attacks fall into this category, such as the attacks making use of the bugs of the programs. On the contrary, several connections have to be inspected to detect the attacks in the second category. Scan and DoS fall into this category. For the first kind of attacks, the sensor would be able to detect the possible attacks without other information if all packets

belonging to the same TCP/UDP connection are sent to it. So, no sensor-to-sensor interaction would be needed if partitioning according to TCP/UDP connections.

5.3 Functional Components

The architecture consists of following components as described in figure5.1.

5.3.1 Network Source

It is installed on a high speed monitored link and it feeds traffic to rest of the system. A network source can be implemented by using a simulator, network tap or switch with span port.

5.3.2 Duplicator

Traffic duplicator is used to duplicate each incoming packet so that all packet s are seen by all nodes. The functionality of a duplicator can be implemented by a simple broadcast or multicast to some selected nodes.

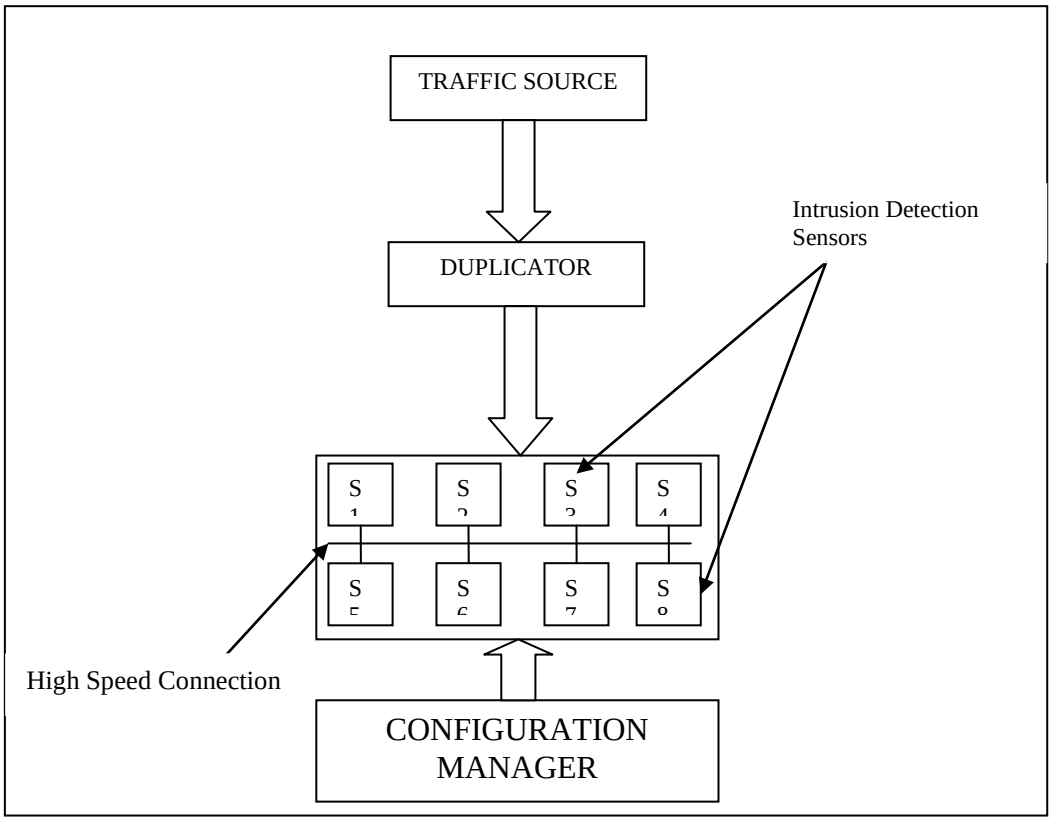

Figure5. 1: Functional Components of proposed architecture

5.3.3 Configuration Manager

It controls and updates the configuration of rules on each sensor. Every sensor contains complete rule set which is divided according to configuration file placed on each intrusion detection sensor. Configuration manager maintains and updates the contents of this configuration file on each sensor.

5.3.4 Sensor Array

It refers to a collection of systems on which signature based IDS is installed. Sensor array can be implemented by installing signature based IDS on individual computers. Each sensor is equipped with a policy database and a local configuration file which enables a certain rule set on that sensor. The configuration manager can update this file dynamically to regulate the processing load on that sensor. Each intrusion detection sensor will match incoming traffic against those rules which are

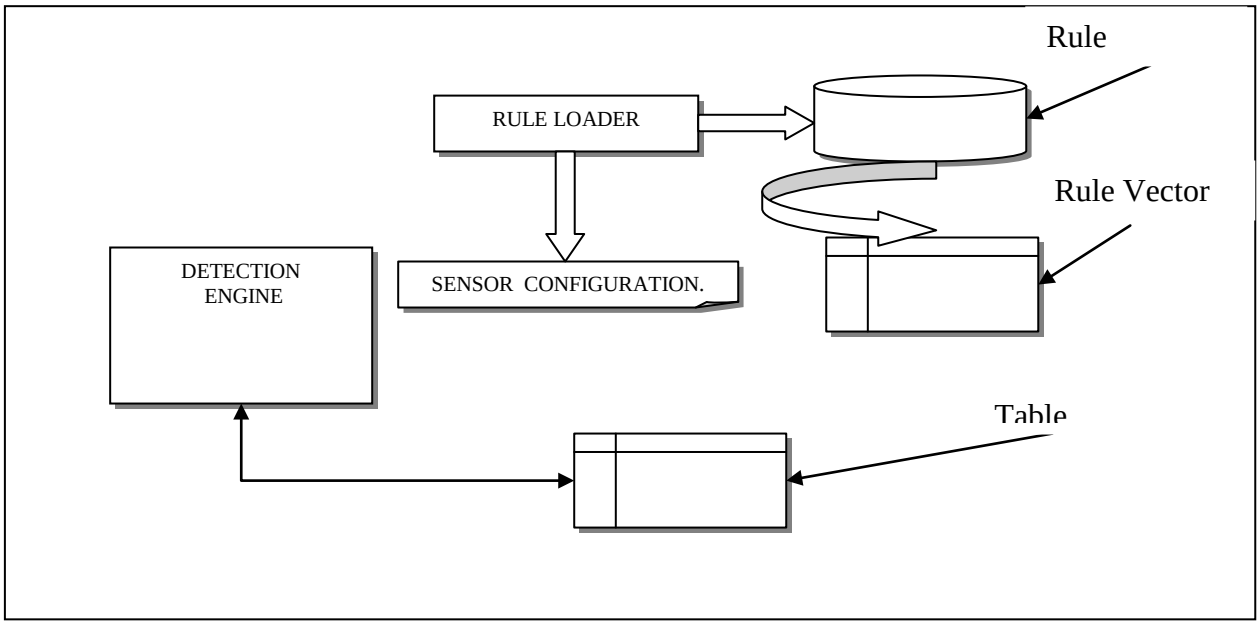

Figure5. 2: Functional Decomposition of a Sensor

enabled in file. The proposed architecture uses multiple sensors for signature matching. Each sensor is equipped with different components as shown in the figure 5.2.

5.3.4.1 Detection Engine

Each sensor is equipped with a detection engine which is installed on it. This detection engine can be an existing signature-based intrusion detection system.

5.3.4.2 Rule Loader

This component reads Sensor.conf file and updates the entries in rule vector as well as Table '*S*'

5.3.4.3 Mapping Table

Each Sensor contains a table "***S"*** that maps hash value ***k*** to the Rule-set ***R***. The hash value is calculated as a function of source and destination port numbers.

5.3.4.4 Rule Vector

The rules are loaded and placed in a data structure designed especially to accommodate rules. This data structure can be implemented in the form of a vector.

5.4 Working

Each sensor upon startup read its configuration. The format of configuration file is shown in figure 5.3. The entries of the file can be manually or automatically updated by using configuration manager.

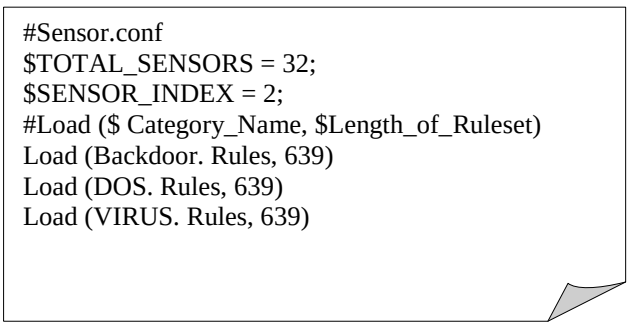

Figure5. 3 Format of Sensor.conf file

Each sensor is assigned an index number on the basis of which rules are loaded from specific rule sets. Sensor index of each intrusion detection sensor is specified in the configuration file along with total number of sensors used by parallel IDS. The configuration file also specifies routines or functions for loading rules from a specific category of rule sets. Each Load routine takes two arguments; rule set category and total number of rules. $LENGTH describes the length of a particular rule set. Each sensor reads configuration file and loads a certain subset of the overall IDS policy according to a predefined algorithm shown in Figure 5.2. Given the sensor index, total sensors and length of a rule set, the algorithm tries to find indexes that describe which rules from a specific rule set are to be loaded. The algorithm divides length of a rule set by number of sensors used. The values of quotient and remainder are stored. If rules are not perfectly divisible by total sensors, the remaining rules are divided on *P-R* processors as shown in figure 5.4. For instance If 639 rules are to be divided across 32 nodes, the algorithm proceeds as shown in figure 5.5.

```
Func (SENSOR_INDEX, RULESET, TOTAL_SENSORS)
Begin
        Quotient = 0;
        Remainder = 0;
        Size = 0;
        Quotient = RULESET/TOTAL_SENSORS;
        Remainder = RULESET%TOTAL_SENSORS;
        Size = Quotient;
        If (SENSOR_INDEX < Remainder)
                SIZE++;Int ARRAY [SIZE];
        Int COUNT = 0;
        FOR (INT I = 0; I < SIZE; I++)
                ARRAY [I] = INDEX + COUNT;
                COUNT = COUNT + TOTAL_SENSORS
End
```

Figure5. 4: Rule Loading Algorithm

By introducing this mechanism of rule division, we ensure the portability of proposed architecture where the changes have to be made only on configuration manager and not on every sensor. Any change made by the user or the system is automatically picked by each sensor as configuration manager updates the entries by updating sensor configuration file on each sensor.

```
Given
        SENSOR_INDEX = 2
        RULESET = 639
        TOTAL_SENSORS = 32
==============================================
        Quotient = 639/32 = 19
        Remainder = 639%32 = 31
        Size = 19
        AS      SENSOR_INDEX < Remainder
                SIZE++; SIZE = 20;
        Int ARRAY [20];
        Int COUNT = 0;
        FOR (INT I = 0; I < SIZE; I++)
                ARRAY [I] =SENSOR_INDEX + COUNT;
                COUNT = COUNT + TOTAL_SENSORS
```

Figure5. 5: Rule Loading

Each sensor maintains a table ***'S'*** which is used for the selection of a rule set for a packet. It maintains information about the location of rules in RAM belonging to particular category of hash value. The hash value can be calculated as function of source and destination port numbers. The format of this table is shown in figure 5.6. After a Sensor has loaded a rule set, it records its entry in this table.

Hash-Value	Category	Offset	Length
8568	SNMP	1	60
0023	TELNET	61	85
0021	FTP	86	115
0080	HTTP	116	217
3600	DNS	218	290

Figure5. 6: Format of Table *'S'*

These entries are then used in the later stage when the sensor performs a decision based upon the packet type that which rules are to be loaded. After the Sensor has loaded the rules and maintained the Table 'S' and all the entries of rule vector are populated , it is set to receive traffic from the splitter. When the sensor receives a packet ***P***, the sensor has to decide that the packet has to be evaluated against which type of traffic. The decision depends upon the type of packet and as the rules are categorized on the basis of porn numbers hash values , therefore the Sensor will perform a simple hash function on port number and will map the result ***'K'*** onto the Table ***'S'.***

Step1: Calculate ***src_port||dest_port*** *hash value* ***'k'***
Step2: Search for the Rule-set ***R*** to which ***k*** maps in ***S***
Step 3: Read offset of R in **S**
Step 4: Read length ***l*** of the Rule-set ***R*** in ***S***
Step 5: For index = offset; index < l; index++
Intrusive = Detection-engine(P,R[index]);
If (Intrusive)
Log (P, R[index])

Figure5. 7: Rule set selection algorithm

Depending upon the type of hash value, the rule set is selected and the detection engine starts signature matching on the packet as shown in figure 5.7.

5.5 Error Reporting

Error reporting or taking any action against an Intrusion is an important issue in parallel IDS. In centralized architecture, single intrusion detection sensor performs all the matching and produces an alert and the availability of rules and signature matching is central. But in parallel architecture, each sensor is evaluating a particular packet against a different rule set and there may be more than one match on a single packet. The information coming from several intrusion detection sensors must be combined and correlated by a central process. The proposed Architecture uses a simple and efficient

way of combining and correlating this information as shown in figure 5.8. Each sensor evaluates packet payloads against attack signatures. Whenever a malicious packet is found, the sensor records information related to the type of attack in a table.

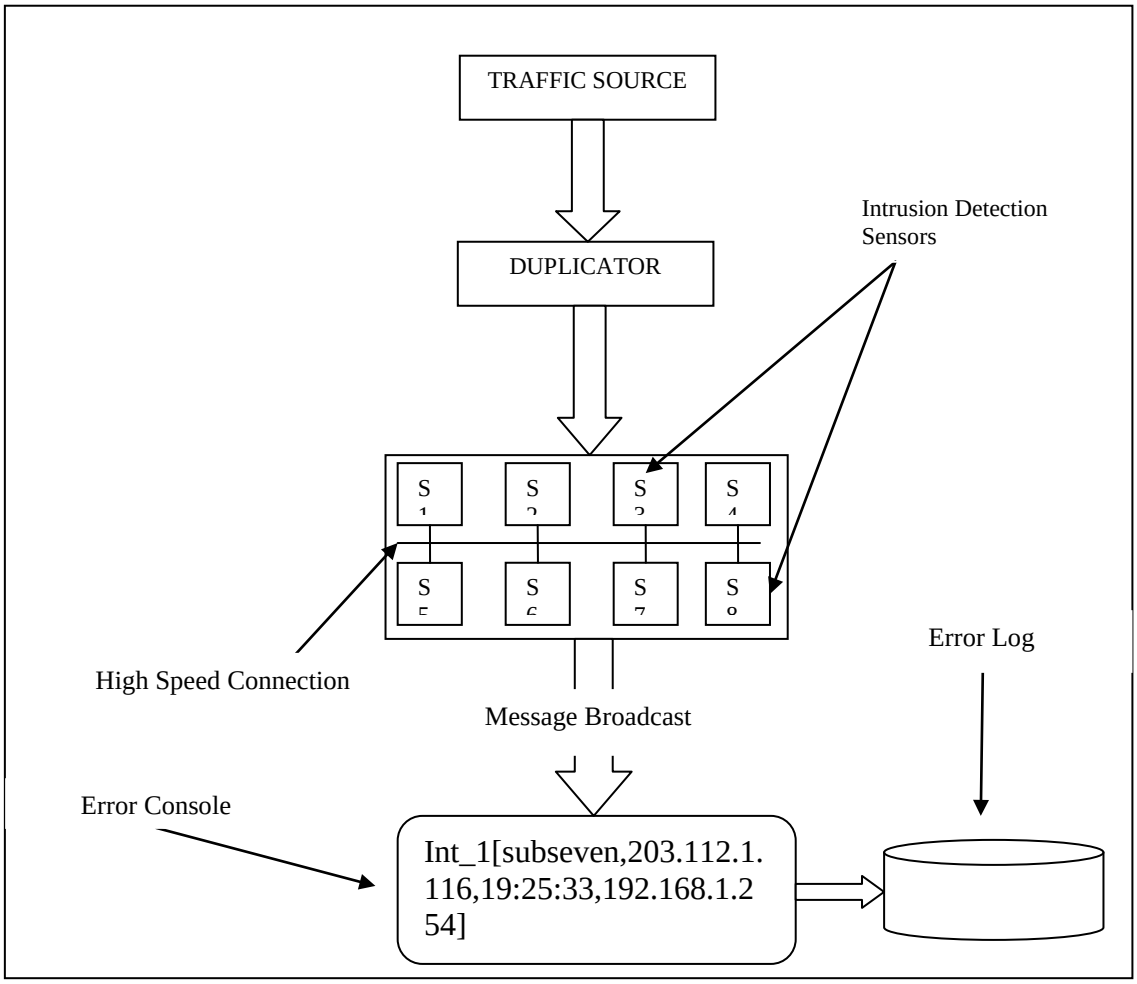

Figure5. 8: Correlation mechanism for parallel intrusion detection system

The table has a specific entry for each sensor to avoid mutual exclusion. Error Console is a lightweight thread that monitors the status of the table and whenever an update is made by a sensor, error console reads the entry and produces a notification of that intrusion. The information is also logged in database for further use.

5.6 Summary

This chapter has presented our proposed architecture in detail. This architecture employs the concept of function parallelism for increasing the performance of signature-based IDS. Instead of dividing the data, it divides rules on the basis of rule groups. Each intrusion detection sensor receives same data and evaluates it against different signatures. Finally the results are combined using a broadcast message. The message is received by error console which displays the intrusion on the screen and logs it in database.

Chapter 6

PROPOSED MULTI-THREADED ARCHITECTURE FOR HIGH SPEED IDS

6.1 Introduction

We have presented our proposed architecture based on the concept of function parallelism in Chapter 5. The architecture has the potential to perform well on high performance computing platforms where Intrusion Detection Systems are installed on general purpose central processing units. But the hardware architecture has been changing during the last two decades and the biggest players in processor manufacturing are relying on multiprocessors. Rather than increasing the speed of single processor, multiple processors are stamped on a single chip. Therefore an application has to be implemented in such a way that it gains benefit from the underlying architecture in order to perform well. The improvements in the fabrication of semi-conductors and processor implementation steadily increased the performance of sequential computer programs for the past three decades. The performance gain through the inception of multi-core architectures represents a fundamental shift in computing. The normal desktop applications with sequential execution cannot gain any benefit from multi-core architectures .The individual cores will become much simpler and run at lower clock speed in order to reduce power consumption on dense multi-core processors.

The architectural changes in multi-core processors benefit only concurrent applications and therefore have little value for most existing mainstream software [9]. These changes mark a fundamental turning point for mainstream software development industry. New

applications cannot simply rely on increasing clock speed. For high performance gains, the applications have to be highly concurrent.

In this chapter we present a lightweight multi-threaded design model that gains advantage of multiprocessor architectures. This allows spreading the signature matching workload over multiple CPUs of the same machine and performance to scale beyond single-CPU machines.

6.2 Design Goals

The proposed architecture considers some design issues in order to maximize the performance of proposed architecture. The algorithm should divide the whole workload into slices with equal sizes. This requirement is the key to guarantee the system's performance. If workload on a single thread is larger than others, the thread that processes that workload will become the bottleneck of the system and waste other threads' process abilities. Each slice contains all the evidence necessary to detect a specific attack. This requirement assures that each thread can detect attacks without any interaction so that the system's complexity is greatly reduced.

The proposed model divides overall IDS policy across multiple threads. We argue that data parallel approach for dividing the workload is not suitable as it is characterized by some drawbacks such as connection state maintenance and tcp reassembly. Moreover by assigning individual connections to separate threads, the workload cannot be divided uniformly due to variation in traffic type and volume. The proposed model therefore divides the policy across multiple threads and performs a uniform distribution of workload. The load-balancer copies each packet to trace tables, which are specially designed data structures to accommodate network traffic and IDS rules.

6.3 Functional Components

The proposed multithreaded model for intrusion detection consists of different components which are illustrated in the figure 6.1.

6.3.1 Load Balancer

Instead of separating the individual phases from the serial execution and including it in multi-threaded path, we introduce a sophisticated and easy to implement load balancer. In case of shared memory architecture, the load balancer does not need to handle issues like connection state monitoring. From the packet capturing phase the packets are forwarded to the load balancer, which initializes the main thread and also keeps track of the maximum number of threads that are currently active. Load-balancer can also be implemented as a part of the packet capturing module. When a packet ***'p'*** is received by the load-balancer, it duplicates the packet onto trace tables and notifies all the threads about the arrival of new packet so that they are consumed by daemon threads for signature-matching process. Load-balancer also updates and divides IDS policy on every table. The policies are updated and loaded on system start-up. Whenever a new rule is entered by a user or an application, load-balancer updates the copies on every trace table.

6.3.2 Daemon Thread

Daemon thread acts as a child thread for the load-balancer. Each daemon has a unique id and based upon that ID, it loads a certain rule set and places it in the trace table. Whenever a new packet '***P***' arrives, load-balancer notifies every daemon about the arrival of the packet. Each daemon compares it against it against a subset of the overall policy and if it finds any intrusion, it notifies the load-balancer about the malicious packet.

6.3.3 O/P Thread

The architecture doesn't allocate any special thread for producing any output. We believe that majority of the traffic in a network is clean and very few packets are malicious. Therefore assigning a separate o/p thread for producing alerts can significantly reduce the efficiency of the system. The proposed architecture presents different alternatives for assigning a separate thread to show alerts. Load-balancer checks the availability of any daemon thread and assigns it, the task of logging the malicious packet and producing alert.

6.3.4 Trace Tables

These are used as data structures for holding incoming packets and rule files. Each table has two portions, one for accommodating the rule set loaded by the daemon threads and the other which provide storage for the incoming packets. The rule sets are placed on the basis of a simple hash-function of source and destination port numbers. When a packet

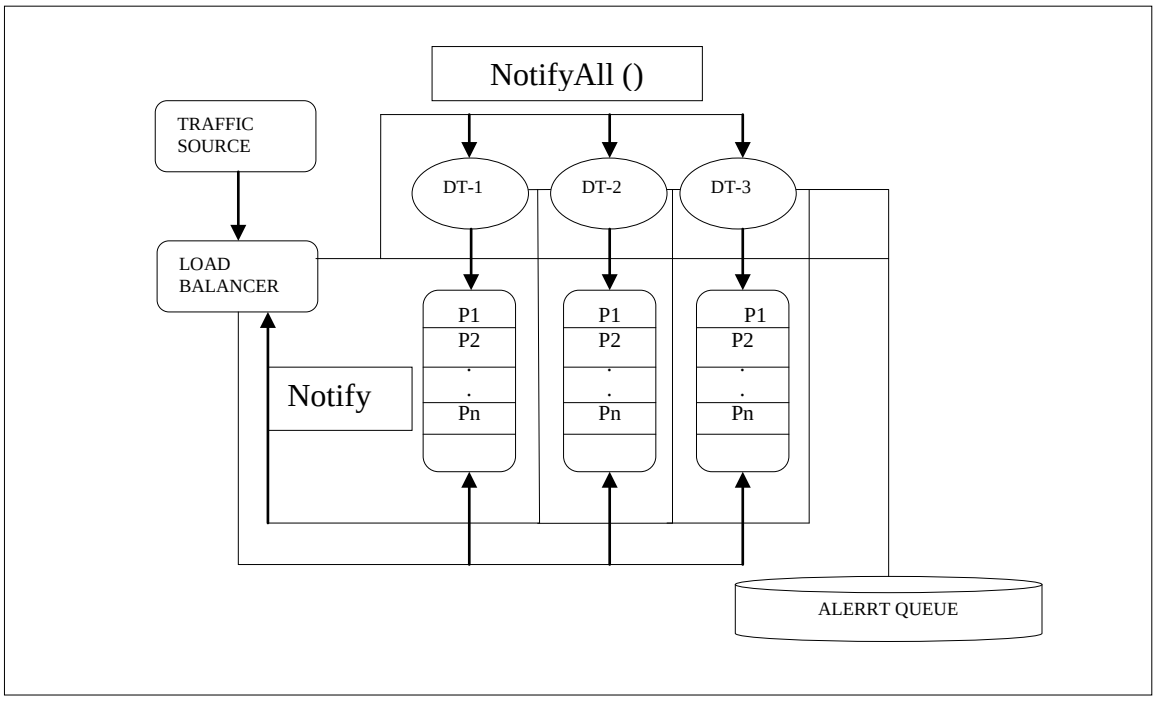

Figure6. 1: Multi threaded architecture for intrusion detection

'P' is received by a daemon thread, it performs a hash-function on source and destination port numbers and maps the resultant hash value onto the trace table which keeps an entry of a unique rule set for each hash value. The trace tables provide very simple and efficient mechanism for storing packet payloads and the rule sets

6.4 Summary

Due to the emergence of multi-core processors, application software cannot rely on increasing clock speed. Applications have to be highly concurrent in order to achieve high performance gains. This chapter has presented a concurrent architecture of signature-based IDS which uses multiple threads for signature matching. Instead of allocating a separate output thread, dynamic load balancing mechanism is introduced which ensures maximum utilization of hardware resources.

Chapter 7

IMPLEMENTATION

7.1 Introduction

The implementation of function parallel architecture has been done using default signature set of snort. The implementation of duplicator, configuration manager has been done in java using jdk1.4 and Jpcap for packet capturing. This chapter presents the performance evaluation of function parallel architecture on the basis of different parameters like execution time, speed up factor, cost, efficiency and effectiveness.

7.2 Experimental Setup

This section presents the experimental results for the parallel algorithm. The results were taken on a High Performance Computing Cluster. The cluster consists of dual 3.06 GHz, 1 GB RAM control node, 16 HP and 16 SUN 2.2 GHz compute nodes with 4 GB RAM on each node. Control node runs Red Hat Enterprise Linux AS Operating System (Release 3 Update 6) whereas compute nodes run the WS version of the same OS. All nodes are interconnected using a Gigabit interconnect through HP ProCurve 2848 switch. The implementation has been done in Java. MPJ Express [18] has been used for code parallelization of a prototype implementation of a signature-based NIDS. The configuration manager and the splitter are installed on the control node and snort signatures are placed on each node for intrusion detection.

Snort uses a simple, lightweight rules description language that is flexible and quite powerful. There are a number of simple guidelines to remember when developing Snort rules. Most Snort rules are written in a single line. This was required in versions prior to 1.8. In current versions of Snort, rules may span multiple lines by adding a backslash \ to

the end of the line. Snort rules are divided into two logical sections, the rule header and the rule options. The rule header contains the rule's action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information. The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken. The text up to the first parenthesis is the rule header and the section enclosed in parenthesis contains the rule options. The words before the colons in the rule options section are called option *keywords*. All of the elements in that make up a rule must be true for the indicated rule action to be taken. When taken together, the elements can be considered to form a logical AND statement. At the same time, the various rules in a Snort rules library file can be considered to form a large logical OR statement. The rule header contains the information that defines the who, where, and what of a packet, as well as what to do in the event that a packet with all the attributes indicated in the rule should show up. The first item in a rule is the rule action. The rule action tells Snort what to do when it finds a packet that matches the rule criteria. There are 5 available default actions in Snort, alert, log, pass, activate, and dynamic. In addition, if you are running Snort in inline mode, you have additional options which include drop, reject, and sdrop.

7.2 Results

The experimental results were obtained on 1, 4, 8, 12, 16, 20, 24, 28 and 32 intrusion detection sensors. The experiment was carried out with 7458 default rules of Snort 2.6 release. The alerts were directed to standard output. Traffic is taken from the Lincoln's lab DARPA Intrusion detection evaluation data sets [19, 20] specifically the outside tcpdump data of week 1 and 4. Care was taken that dump file was completely cached in RAM at the start of experiment. The results were evaluated on the basis of following parameters.

7.2.1 Execution Speed

The experiment was performed 20 times and run times were averaged. A comparison of execution time is presented in figure 7.1. It is clear that function parallel method reduces the overall time and sticks closely with theoretical model.

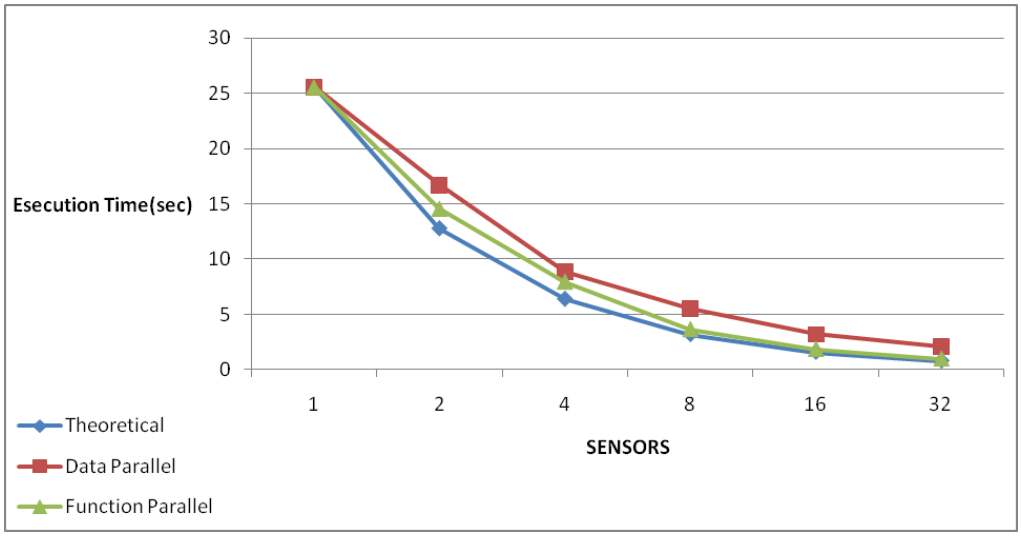

Figure7. 1: Execution Time (sec)

7.2.2 Speed-up Factor

A measure of relative performance between a multiprocessor system and a single processor system is the speedup factors (n), defined as

$$S(n) = \frac{Execution\ time(single\ processor)}{Executiontime(multiprocessor)} = \frac{ts}{tp}$$

Where t_s is the execution time on a single processor and t_p is the execution time on a multiprocessor. S(n) gives the increase in speed in using a multiprocessor. The

experimental results show a significant rise in speed-up with increase in the number of sensors. The observed speed-up sticks with theoretical speed-up and shows a performance gain over classical data parallel approach for intrusion detection systems as shown in figure 7.2.

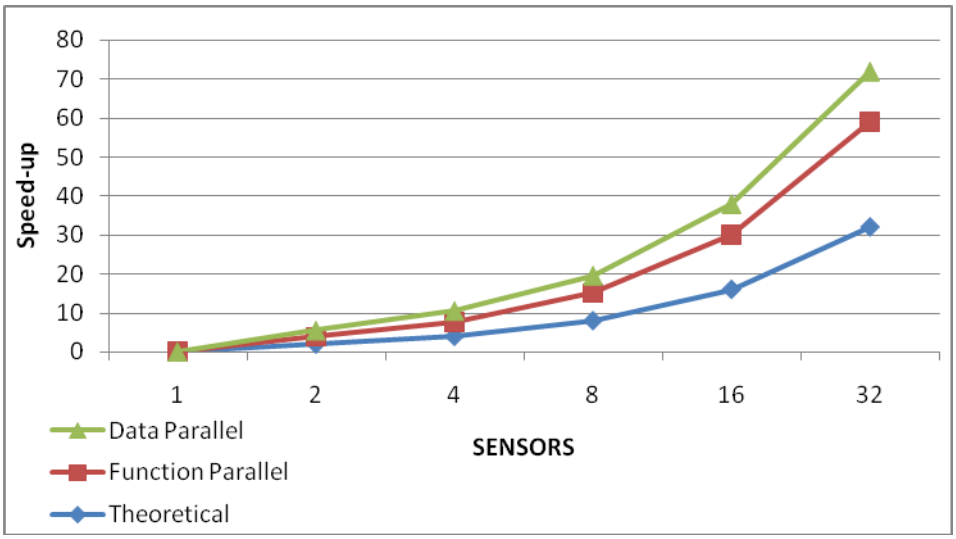

Figure7. 2 : Speed-up Factor

7.2.2 Effectiveness

Effectiveness is a measure of load distribution on multiple sensors. The data parallel approach cannot regulate the load balancing due to issues like connection state maintenance and TCP reassembly. The proposed architecture divides workload by evenly distributing the overall policy across multiple sensors. Figure 7.3 shows the distribution of rules on ten different sensors for a Trojan Scan.

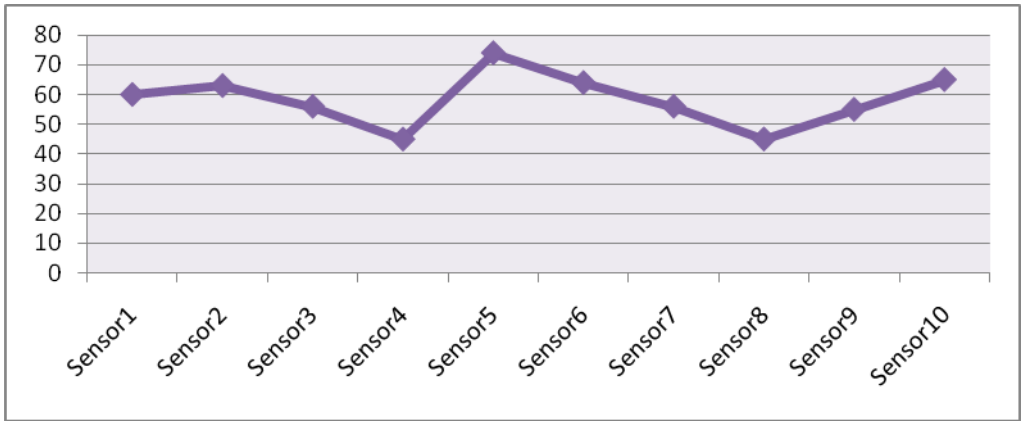

Figure7. 3: Effectiveness on ten IDS sensors

7.2.4 Cost

The cost of parallel architecture is defined as the product of running time and the number intrusion detection sensors.

NODES	Cost-Data Parallel	Cost-Function Parallel
1	33.54	29.12
2	33.54	28.92
4	39.44	29.23
8	44.32	29.92
16	65.46	27.52
32	78.34	27.21

Figure7. 4: Cost of data and function parallelism methods

Figure 7.4 indicates that proposed solution is cost effective as compared to data parallel approach which suffers mainly due to diversity of traffic, making it impossible for the splitter to divide uniformly.

7.3 Summary

This chapter has presented results of our parallel implementation on cluster system. Implementation has been in java language and JPCAP has been used as packet capturing library. The performance of proposed architecture has been evaluated mainly in terms of speed-up factor and cost. Proposed solution for high speed intrusion detection is cost effective and processes more traffic in less time as compared to classical data parallel approach for IDS.

Chapter 8

FUTURE WORK

8.1 Overview

The performance requirements of Intrusion Detection Systems are continuously increasing with increase in network speed and traffic loads .The ability of the system to handle increasing traffic loads becomes even more crucial when IDS are placed in line, just becoming an IPS. It is severe enough to have overloaded IDS which drops packets as this can lead to missed intrusions. But when an IPS is overloaded it is not only the system itself which faces a denial-of-service, but all the users whose traffic passes through the system. Numerous approaches to improve the performance of intrusion detection and prevention systems have been suggested by researchers in all areas of computer science. Some approaches focus on improving software and algorithms, others introduce new hardware techniques, and still others examine the use of parallelism in intrusion detection. Our current work has presented a function-parallel architecture for enhancing the performance of Intrusion Detection Systems. We argue that data parallel approaches are categorized by some drawbacks which can be overcome by using our proposed Function-Parallel Architecture. The proposed architecture outperforms existing data parallel approaches and results obtained by our parallel implementation have shown significant speed. The approach has also shown the potential to be extended and implemented on reconfigurable hardware to develop a cost-effective and scalable solution for the future.

We have also discussed the importance of concurrency in the context of Intrusion Detection Systems. The introduction of multi-core architecture has revolutionized the approaches towards speed up. Now an application cannot perform well until its design

considers architectural considerations right from the start. We have discussed our proposed Multi-threaded model for Intrusion detection system and carried out its comparison with single-threaded implementations.

8.2 Future Work

There are many interesting avenues for future work. The function parallel and data parallel approaches can be combined in an effective manner to gain advantage on high speed network. The increase in the amount of data is tremendous and this trend will continue in the future. Data Parallel technique can be used to divide the data in slices and then for processing each slice, the concept of function-parallelism can be employed. We showed that Function-Parallel implementation of a signature-based NIDS sensor is able to outperform existing data parallel approach. This result provides a new approach to increase the performance of individual sensors. We were only able to evaluate our designs on 32 nodes. It would be very interesting to evaluate its performance on more nodes to develop a better understanding of the proposed architecture.

The idea can be further extended to parallelize different stages of intrusion detection. For future NIDS implementations, we believe that Function-Parallel approach should be seriously considered. We were able to achieve a performance increase through modification of existing code, but we think that a complete new implementation that takes function parallelization into account from the start would suffer less overhead and perform even better.

BIBLIOGRAPHY

[1] Computer Emergency Response Team, "Vulnerability Statistics", *Computer Emergency Response Team, Carnegie Mellon University,* 1995-2008. [Online]. Available: http://www.cert.org/stats/fullstats.html. [Accessed: Mar 5, 2008].

[2] McHugh, J., *"Intrusion and Intrusion Detection"*, International Journal of Information Security 2001

[3] Escamilla, T. *"Intrusion Detection: Network Security beyond the Firewall"*. John Wiley & Sons Inc, New York, 1998.

[4] Roesch, M. "Snort: Lightweight Intrusion Detection for Networks", 13th Conference on Systems Administration, Berkeley, CA, 1999

[5] Hofmeyr, S. A., Forrest, S., and Somayaji, A., "*Intrusion Detection using Sequences of System Calls"* Journal of Computer Security, 1998.

[6] Ramalingam, G. "*Context-sensitive Synchronization-Sensitive Analysis is undecidable"*, ACM Transactions on Programming Languages and Systems

[7] Patrick Wheeler, Errin Fulp,"*Taxonomy of Parallel Techniques for Intrusion Detection"*, *ACMSE 2007* March 23-24, 2007

[8] Axelsson, S., *"Research in Intrusion-Detection Systems: A Survey"*. Technical Report, Department of Computer Engineering, Chalmers University of Technology, Sweden, 1998.

[9] http://www.gocsi.com, Computer Security Institute (CSI) FBI Survey 2007

[10] Paxson, V. "*Bro: A System for Detecting Network Intruders in Real-time*" Elsevier Computer Networks, 1998

[11] Lane, T., Brodley, C. E., *"Temporal Sequence Learning for Anomaly Detection"* 5th ACM Conference on Computer and Communications Security, 1998

[12] Hofmeyr, S. A., Forrest, S., and Somayaji, A., "*Intrusion Detection using Sequences of System Calls"* Journal of Computer Security, 1998

[13] Samaha, S. E. Haystack: *"An Intrusion Detection System"*, 4th Aerospace Computer Security Applications Conference, IEEE Computer Society Press, 1988

[14] Lane, T., and Brodley, C. E. *"Temporal Sequence Learning and Data Reduction for Anomaly Detection"*, ACM Transactions on Information Systems Security, 1999

[15] Forrest, S., Hofmer, S. A., Somayaji, A., and Longstaff, T. A. *"A Sense of Self for UNIX Processes",*. IEEE Symposium on Security and Privacy, IEEE Press, Oakland, May 1996

[16] Errin W. Fulp and Ryan J. Farley, *"A Function- Parallel Architecture for High-Speed Firewalls"*, Proceedings of ICC IEEE 2006

[17] Anita K. Jones and Robert S. Sielken ,"*Computer System Intrusion Detection: A Survey"*

[18] Northcutt, S., Cooper, M., Fearnow, M., and Frederick, K. Intrusion Signatures and Analysis, 1st ed. New Riders, SANS GIAC. Indianapolis, IN, January 2001

[19] Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R., and Zerkle, D., *"GrIDS – A Graphbased Intrusion Detection System for Large Networks"*, 19th National Intrusion Detection Working Group – Information Systems Security Conference, October 1996

[20] Dowel C, R. "The Computer Watch Data Reduction Tool", 13th National Computer Security Conference, Washington, October 1990

[21] Goldberg, I., Wagner, D., Thomas, R., and Brewer, E. A.," *A Secure Environment for untrusted Helper Applications"*, 6th USENIX Security Symposium, July 1996.

[22] http://sourceforge.net/projects/libpcap

[23] H. Song, T. Sproull, M. Attig, and J. Lockwood, "*Snort offloader: A reconfigurable hardware NIDS filter*", 15th International Conference on Field Programmable Logic and Applications (FPL), Tampere, Finland, Aug. 2005

[24] H. Song and J. W. Lockwood, *"Efficient packet classification for network intrusion detection using fpga,"* Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field- programmable gate arrays. New York, NY, USA: ACM Press

[25] Christopher R. Clark, "*Design of efficient fpga circuits for matching complex Patterns in network intrusion detection systems"*, Master's thesis, Georgia Institute of Technology, December 2003

[26] Christopher Kruegel Fredrik Valeur Giovanni Vigna Richard Kemmerer, "*Stateful Intrusion Detection for High Speed Networks*", Proceedings of IEEE symposium on Security and Privacy, 2002

[27] Charitakis, K. Anagnostakis, E. Markatos,"*An Active Traffic Splitter Architecture for Intrusion Detection"* 11th IEEE/ACM International Symposium onAnalysis and Simulation of Computer Telecommunications Systems, 2003

[28] Lambert Schaelicke, Kyle Wheeler, Curt Freeland, SPANIDS: *"A Scalable Network Intrusion Detection Load balancer"* Sixth IEEE International Symposium on Network Computing and Applications, 2007

[29] Charitakis, K. Anagnostakis, E. Markatos," *An Active Traffic Splitter Architecture for Intrusion Detection"* IEEE Transactions on Dependable and Secure Computing, Jan-Mar 2006

[30] Lambert Schaelicke, Kyle Wheeler, Curt Freeland, "SPANIDS:*A Scalable Network Intrusion Detection Load balancer"* Sixth IEEE International Symposium on Network Computing and Applications, 2007

[31] Robin Sommer, Vern Paxson, *"Exploiting Independent State for Network Intrusion Detection"*, Proceedings of the 13th ACM conference on Computer and communications security, 2006

[32] Bro Intrusion Detection System, [Online]. Available: www.**bro-ids**.org. [Accessed: Mar 5, 2008]

Appendix "A" – Code Listing

```
import java.io.*;
import java.util.*;
import java.net.*;
public class FPAIDSDaemon {
  private int total_sensors = 0;
  private int sensor_index = 0;
  private String [] rulefiles = null;
  private String configurationFilePath = null;
  private ServerSocket server = null;
  private int serverport = 3000;
  private int indexes[] = null;
  private Rule[] IDSPolicy = null;
  private Rule[] sensorPolicy = null;
  public void updateSensorConfiguration( String configurationFilePath )
  {
    try
    {
      FileInputStream fin = new FileInputStream(configurationFilePath);
      ObjectInputStream ob_in = new ObjectInputStream(fin);
      Configuration sensor_configuration =
      (Configuration)ob_in.readObject();
      this.sensor_index = sensor_configuration.getSensor_index();
      this.total_sensors = sensor_configuration.getTotal_sensors();
      Vector v = sensor_configuration.getRulespath();
      this.rulefiles = new String[v.size()];
      for ( int i = 0 ; i < rulefiles.length ; i++ )
      {
        this.rulefiles[i] = (String)v.elementAt(i);
       }//end for
```

```
   }//end try
    catch (FileNotFoundException ex)
    {
      System.out.println(ex.getMessage());
      ex.printStackTrace();
    }//end catch
    catch (IOException ex)
    {
      System.out.println(ex.getMessage());
      ex.printStackTrace();
    }//end catch
    catch (ClassNotFoundException ex)
    {
      System.out.println(ex.getMessage());
      ex.printStackTrace();
    }//end catch
   }//end updateSensorConfiguration method
  public int[] calculateRuleIndexes( int total_rules , int sensor_index
, int total_sensors )
  {
    int indexes[] = null;
    int remainder = 0;
    int q = total_rules/total_sensors;
    indexes = new int[q+1];
    int index = 0;
    remainder = total_rules%total_sensors;
    for ( int i = 0 ; i <= q ; i++ )
    {
     index = sensor_index + (total_sensors*i);
     if ( index >total_rules-1){
       break;}
```

```
     else
     {
         indexes[i] = index;
     }
     //System.out.println(indexes[i]);
    }//end for
   return indexes;
  }
public Rule[] readSerializedRules( String serializedRuleFilePath )
{
  Rule rules[] = null;
  FileInputStream fin = null;
  ObjectInputStream ois = null;
  Object temp[] = null;
  try {
      fin = new FileInputStream(serializedRuleFilePath);
      ois = new ObjectInputStream(fin);
      Vector v = (Vector)ois.readObject();
      temp = v.toArray();
      rules = new Rule[temp.length];
      for ( int i = 0 ; i < rules.length ; i++ )
      {
        rules[i] = (Rule)temp[i];
      }//end for
    }//end try
    catch (FileNotFoundException ex) {
      ex.printStackTrace();
    }//end catch
    catch (IOException ex) {
      ex.printStackTrace();
    }//end catch
```

```
    catch (ClassNotFoundException ex) {
      ex.printStackTrace();
    }//end catch
  return rules;
}//end method readSerializedRules
public void printIDSPolicy ()
{
  for ( int i = 0 ; i < IDSPolicy.length ; i++ )
 {
     System.out.println(IDSPolicy[i].toString());
   }//end for
}//end printIDSPolicy
public void printSensorPolicy()
{
  for ( int i = 0 ; i < this.sensorPolicy.length ; i++ )
  {
    System.out.println(sensorPolicy[i].toString());
  }//end for
}//end method
public void sleep( int interval)
{
  try
    {
      for ( int i = 0 ; i < 50 ; i++ )
      {
        Thread.sleep(interval);
        System.out.print(".");
      }
      System.out.println();
     }
    catch (InterruptedException ex1) {
```

```
      ex1.printStackTrace();
     }
}
  public FPAIDSDaemon()
  {
  }//end constructor
  public FPAIDSDaemon(String configurationFilePath)
  {
    this.configurationFilePath = configurationFilePath;
    System.out.println("Reading Configuration");
    this.sleep(20);
    this.updateSensorConfiguration(this.configurationFilePath);
    System.out.println("updating Configuration");
    this.sleep(30);
    System.out.println("Loading IDS Policy");
    this.sleep(100);
    this.IDSPolicy = this.readSerializedRules(this.rulefiles[0]);
    System.out.println("calulating indexes...");
    this.sleep(10);
    this.indexes =
this.calculateRuleIndexes(this.IDSPolicy.length,this.sensor_index,this.
total_sensors);
    System.out.println("Extracting policy based upon indexed values");
    this.sleep(100);
    this.sensorPolicy = new Rule[this.indexes.length];
    for ( int i = 0 ; i < sensorPolicy.length ; i++ )
    {
      sensorPolicy[i] = IDSPolicy[indexes[i]];
    }//end for
    this.printIDSPolicy();
    this.printSensorPolicy();
```

```
    try {


      this.server = new ServerSocket(this.serverport);
      System.out.println("FPAIDS DAEMON listening on port " +
                              server.getLocalPort());
      Socket client  = server.accept();
      DataInputStream din = new
DataInputStream(client.getInputStream());
      String message = din.readUTF();
      System.out.println(message);
      }//end try
    catch (IOException ex) {
      ex.printStackTrace();
    }
  public static void main(String[] args)
  {
    FPAIDSDaemon FPAIDSDaemon1 = new
FPAIDSDaemon("C:\\FPAIDS\\sensor.conf");
  }//end main
}//end class
```