Pattern Matching Algorithms for Intrusion Detection and Prevention Systems

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Information Security

Submitted By
Vibha Gupta
(Roll No. 801233028)

Under the supervision of:
Mr. Vinod Kumar Bhalla
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2014

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*Pattern Matching Algorithms for Intrusion Detection and Prevention Systems*" , in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Information Security* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mr. Vinod Kumar Bhalla* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Vibha Gupta

(Vibha Gupta)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mr. Vinod Kumar Bhalla)
Assistant Professor
Thapar University
Patiala

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgment

I express my sincere gratitude to **Dr. Maninder Singh,** Associate Professor, Computer Science and Engineering Department, Thapar University for their valuable guidance, proper advice, generous attitude and constant encouragement during the course of my thesis work.

I would like to thank to my guide **Mr. Vinod K. Bhalla,** Assistant professor, Comptuter Science and Engineering Department, Thapar University for their invaluable co-operation, generous attitude and above all his blessings that have been persistent source of inspiration for me.

I am also grateful to **Dr. Deepak Garg**, Head of Department, Computer Science and Engineering Department, Thapar University for the motivation and inspiration that triggered me for this thesis.

Last but not the least I would like to express my heartfelt thanks to my parents and my friends who with their thought provoking views, veracity and whole hearted co-operation helped me in doing this thesis.

(Vibha Gupta)

Abstract

Intrusion Detection and Prevention Systems (IDPSs) are used to detect malicious activities of intruders and also prevent from the same. These systems use signatures of known attacks to detect them. Signatures are identified through pattern matching algorithm which is the heart of IDPSs. Due to technological advancements, network speed is increasing day by day, so pattern matching algorithm to be used in IDPS should be fast enough so as to match the network speed. Therefore choice of pattern matching algorithm is the critical to the performance of IDS and IPS.

Several pattern matching algorithms exist in literature, but which pattern matching algorithm will give best performance for IDPS is not known at hand. So in this work four pattern matching algorithms namely Brute-force, RabinKarp, Boyer-Moore and Knuth-Morris-Pratt has been selected for the analysis. Pattern matching parallelization algorithm is proposed that use threads for parallelize the combined approach of RabinKarp and Brute force algorithms. It reduces the time complexity equal to size of alphabets. This work is carried out in Java. Performance of various pattern matching algorithms is analyzed in terms of run time by varying number of patterns and by varying size of network captured (pcap) file.

Table of Content

Certificate ..... **I**
Acknowledgment ..... **II**
Abstract ..... **III**
Table of content ..... **IV**
List of Figures ..... **VI**
List of Tables ..... **VII**
List of Acronyms ..... **VIII**

Chapter 1 Introduction **1**
1.1 Intrusions 1
1.1.1 Zero day attacks 2
1.2 Intrusion Detection and Prevention System 2
1.2.1 Promiscuous Mode 3
1.2.2. Inline Mode 4
1.3 IDS, IPS and Firewalls 4
1.3.1. IDS versus IPS 4
1.3.2. IPS versus firewall 5
1.4 Types of alarms 5
1.5 Functions of IDPS 5
1.6 Detection Methodologies 6
1.6.1 Anomaly-based IDPS 6
1.6.2 Signature-based IDPS 7
1.6.3 Stateful Protocol analysis detection 7
1.7 IDPS Techniques 8
1.7.1 Network based IDPS 8
1.7.2 Host-based IDPS 9
1.7.3 Wireless IDPS 10
Chapter 2 Literature Survey **12**
2.1 Snort IDPS Tool 12
2.1.1 Components of Snort IDPS 13
2.2 Pattern Matching 15

Chapter 3 Problem Formulation **24**
3.1 Problem Definition 24
3.2 Objectives 24
Chapter 4 Design and Implementation **25**
4.1 Algorithm used 25
4.1.1 Brute-Force Search 25
4.1.2 Rabin-Karp Algorithm 25
4.1.3 Knuth-Morris-Pratt Algorithm 26
4.1.4 Boyer-Moore Algorithm 27
4.1.5 Proposed Algorithm 29
4.2 Technologies Used 31
4.2.1 Java 31
4.2.2 Wireshark 32
4.2.3 MySql 32
4.3 Implementation of Pattern Matching Algorithms 32
Chapter 5 Experimental Results **39**
Chapter 6 Conclusion and Future Scope **43**
6.1 Conclusion 43
6.2 Future Scope 43
References **44**
List of Publication **47**

List of Figures

Figure 1.1: Zero day attack protection..... 2
Figure 1.2: IDS implemented in Promiscuous Mode 3
Figure 1.3: IDPS sensor implementation in inline mode..... 4
Figure 1.4: Network based IDPS architecture..... 8
Figure 1.5: Host based IDPS architecture..... 10
Figure 1.6 Wireless IDPS architecture 11
Figure 2.1: Block Diagram of Snort NIDPS..... 12
Figure 2.2: Arrangements of Components of Snort 13
Figure 2.3: Preprocessor written in snort configuration file 14
Figure 2.4: Snort rule example..... 14
Figure 2.5: Snort capturing ping 15
Figure 2.6 Snort database contain many tables and show event table information.....15
Figure 2.7: Snort attack output on BASE 16
Figure 4.1: Flow diagrams of IDS..... 33
Figure 4.2: Browse the pcap file 33
Figure 4.3: Select Algorithm..... 34
Figure 4.4: Attack details by RabinKarp 35
Figure 4.5: Attack details by Parallelization..... 35
Figure 4.6: Show content of pcap file..... 36
Figure 4.7: Details of attacks on particular date..... 36
Figure 4.8: Progressive Bar implementation 37
Figure 4.9: Time Comparison graph of elapsed time 37
Figure 4.10: Demonstration of the working of algorithms 38
Figure 4.11: Demonstration of algorithms after searching the pattern..... 38
Figure 5.1: Time comparison graph of algorithm against different file size 40
Figure 5.2: Time comparison graph of algorithms against number of patterns.....41

List of Tables

Table 2.1: Shift table of KMP.....19
Table 2.2: Shift table of Boyer-Moore.....20
Table 5.1: Run time of algorithms against file size and number of patterns.....39
Table 5.2: Comparison of complexities of algorithms.....42

List of Acronyms

IDS	Intrusion Detection System
IPS	Intrusion Prevention System
IDPS	Intrusion Detection and Prevention System
HIDPS	Host-based Intrusion Detection and Prevention System
NIDPS	Network-based Intrusion Detection and Prevention System
WIDPS	Wireless Intrusion Detection and Prevention System
KMP	Knuth-Morris-Pratt Pattern Matching Algorithm
RK	Rabin-Karp Pattern Matching Algorithm
BM	Boyer-Moore Pattern Matching Algorithm
DFA	Deterministic Finite Automata
NDFA	Non- Deterministic Finite Automata

Chapter 1

Introduction

As internet is the best development in the domain of communication industry. It provides a lot of services in terms of tele-working, e-commerce, e-banking, advertisement and manage large database of the confidential information for the companies. With increasing usage and easy accessibility of internet increase its complexity of coding and implementation which causes networks and systems to be vulnerable against devastating threats like denial of service (DoS), Trojans, backdoors and many web application attacks like SQL injection, Cross site scripting, click jacking. These attacks can leads to loss of information, money, services and reputations of the organizations. The need to mitigate and stop these attacks evolved the concept of Intrusion detection and prevention system.

1.1 Intrusions

An intrusion is a series of activities that are performed by an attacker to compromise the confidentiality, integrity and availability of the computer resources through illegally gain access to the system. There are mainly six types of intrusion exist [1]. Attempted break-ins are the attacks to get into the victim system to change or delete the data of system or to plant logic bombs, backdoors and rootkits. These break-ins detected by behavior profiles or policy violation constraints. Masquerade attacks are performed to gain unauthorized access to systems as legitimate author by using false identity. Stealing of passwords, Hijack the session by predicting session id, Cookie stealing, IP spoofing many others techniques are used to pretend as legitimate user. These types of intrusions are detected by violations of security measures or behavior profiles. Penetration of the system attack is performed with intention to find vulnerability of the system like port scanning, banner grabbing. System penetration is detected by identifying specific patterns of activity. Leakage is the unauthorized transmission of information from a user to an external unauthorized user [5]. Information can be leaked by revealing the system data and debug information unintentionally and by sniffing the traffic information leak intentionally. Atypical use of system resources can detect leakage attack. Denial of Service attack is performed to make resources unavailable to its users. Consumption of bandwidth and memory,

disruption of routing information components of network, crash the operating system are the main cause of Denial of Service attack**.** This attack is detected by atypical use of system resources. Malicious use is the technique to use special tools for malicious purpose, like a packet decoded by Wireshark and confidential information revealed to attacker same as cookie is used to hijack session [7]. This attack is detected by atypical behavior profiles, policy violation or use of system privileges.

1.1.1 Zero day attacks

Zero day attack is an attack that exploits a vulnerability of a system on the same day when it is generally known to developers. Some time attacker found the flaw before the programmer. There are zero days between the vulnerability has been found and the attack happen means a developer has zero days to fix this attack [4].

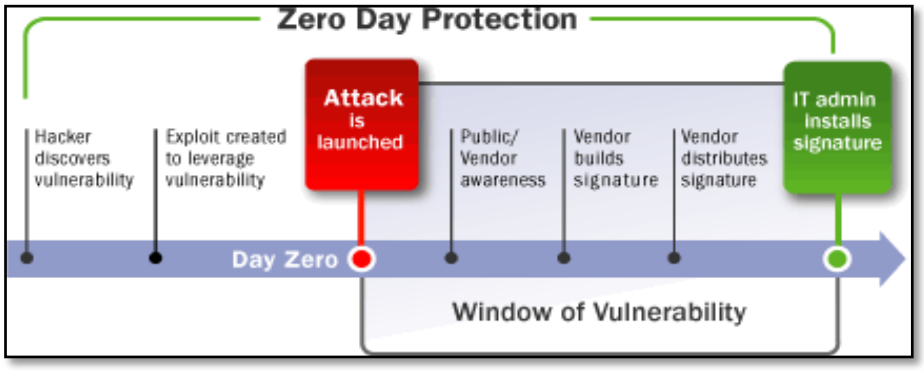


Figure 1.1: Zero day attack protection [4]

1.2 Intrusion Detection and Prevention System

Intrusion Detection System (IDS) is very confident tool for providing security against intrusions that are increasing day by day. The main aim of IDS is to identify misuse, unauthorized access and disruption of information system from both internal and external penetrators. It is done by collecting packets from inflow and outflow traffic and looking for malicious contents and behaviors, after finding attacks generates an alarm for the network security administrator. Whereas Intrusion Prevention System (IPS) has all the capabilities of intrusion detection system but additionally it has capability to drop the malicious packet, change the malicious content to the legitimate content, resetting the sessions or can block the traffic from malicious IP address. IPS is reactive in nature. In many organizations IPSs are combined with Firewalls to provide security against many web based attacks [2]. Advanced IPS technique does

not drop the packet but redirect it to the honeypot to know more about the attacker. Security comes from vision of the attacks and then controls these attacks. Vision of intrusion is provided by IDS by sniffing the traffic and identifies the signatures. While IPS control the attacks by taking preventive actions. Therefore to secure a system or network combined approach of IDS and IPS is used. IDS/IPS can be deployed in two modes, promiscuous mode and inline mode [1] both modes process traffic differently which are defined as follow.

1.2.1 Promiscuous Mode

In this mode, IDS sensor analyses the copy of traffic for intrusion detection. Passive sensors are deployed at the locations where they can monitor divisions and network segments. Promiscuous mode deployment cannot stop the traffic from reaching its destination. In promiscuous mode various methods are used to monitor traffic.

Spanning port is the port on switch that can see all incoming and outgoing network traffic. Therefore IDS sensor is connected to the spanning port. This port mirroring method is inexpensive because this method does not require any additional hardware cost, but when switch is under heavy load or configured incorrectly then spanning port might not able to monitor all traffic [2].

Network Tap is hardware that directly connected between IDS sensor and the physical media such as fiber cable. Network taps have three minimum ports one is monitor port and remaining two ports act as a bridge [2]. The tap forwards copy of all traffic carried by media to the sensor. Taps are immune to heavy traffic. But unlike spanning ports network taps are need to purchase. Installation of a tap and problems with a tap cause downtime to the network.

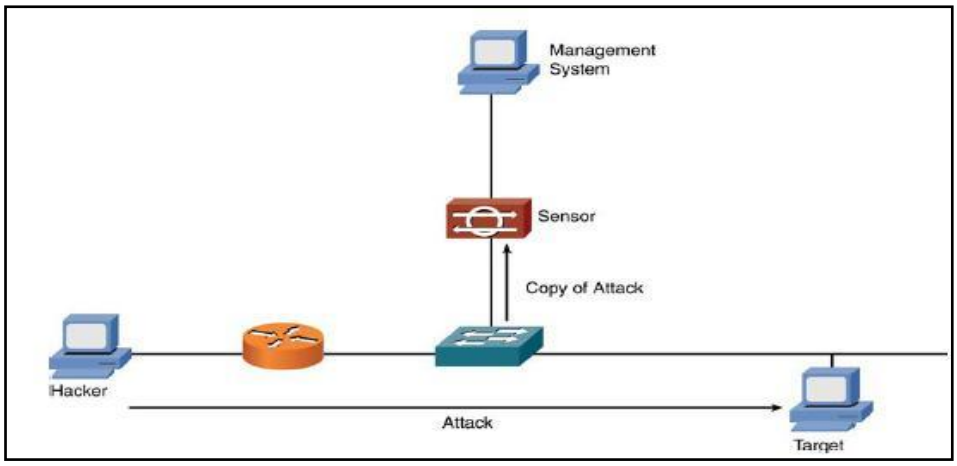


Figure1.2: IDS implemented in Promiscuous Mode

1.2.2. Inline Mode

In inline mode, sensor is deployed directly in the flow of packet. In this mode IDS is configured as transport layer 2 device that passes traffic between two interfaces. It works like layer 2 bridge between network segments. Before reaching to the destination traffic must pass through from the Sensor. By deploying IDS sensor inline to the system with reactive capabilities with attack is known as Intrusion Prevention System (IPS). Inline mode can affect the speed of network traffic because of latency and jitter. Also failure of sensor can cause failure of network.

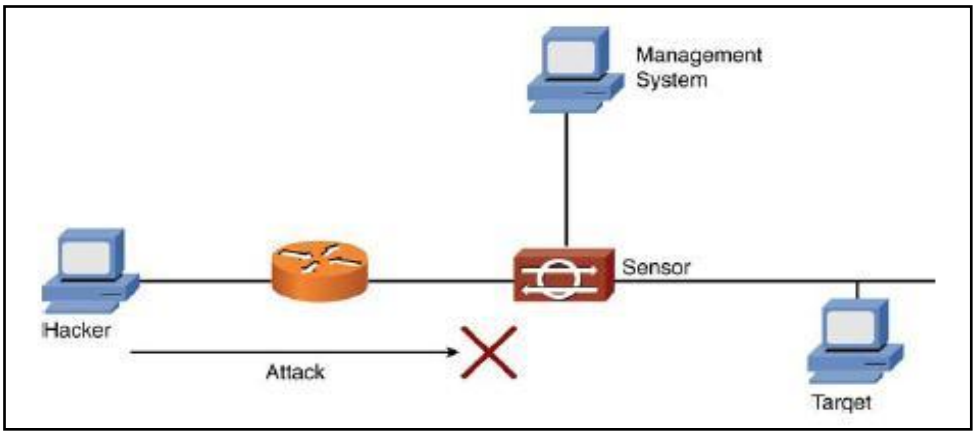


Figure1.3: IDPS sensor implementation in inline mode

1.3 IDS, IPS and Firewalls

IDS, IPS and Firewall provide security to the system but their ways of detecting and preventing from attacks are different.

1.3.1. IDS versus IPS

Intrusion Prevention System (IPS) is an advancement of Intrusion Detection System (IDS). An Intrusion Detection System can only detect an intrusion and report it to the security administrator. But IPS is capable of detecting an intrusion and can also takes preventive measures like dropping of packets that contain legitimate attacks and change the malicious content of legitimate attack [1]. An IPS must be implemented inline so that sensor can stop the attack before reaching to destination. As deployment is inline IPS sensor can degrade the performance of network. Therefore IDS and IPS are combined with the mode selection. These combined systems known as Intrusion Detection Prevention Systems (IDPPSs) [3].

1.3.2. IPS versus firewall

Firewall is used to limiting access to a system or a network of computers by controlling the data which is destined at particular TCP/IP ports. For example, a firewall can stop the traffic which is coming or outgoing to a port 4444 that provide no standard services. However, if an attack is performed through a standard port like 443 (used for HTTPS) there is little that a firewall can do. Firewalls usually check the packets headers and their ports but not the payloads or data of the packet. Some attack like buffer overflow which used payloads to perform an incident is remained unidentified [6]. Firewall use policy if any packet do not follow policy that packet can be dropped. However an IPS can scan the header, flow of the packet and payloads of traffic in real time by matching the data stream of captured packet against a database of known attack patterns called signatures. If any signature matches with the content of packet then IPS can drop or edit the packet content.

1.4 Types of alarms

- **True Positive:** An IPDS raised alarm when actually a legitimate attack occurred on the system or network.
- **False Positive:** An alarm is generated when IDS/IPS incorrectly seeing legitimate requests as intrusion or attack.
- **False Negative:** When IDS/IPS fails to detect attack and generate no alarm for the attack. Sensor does not detect something it should.
- **True Negative:** When IDS/IPS generate no alarm as no attack has occurred to the system. When there is no attack IDPS generate no alarm to administrator.

False Positive and Negatives are the problems of IDPSs. A lot of improvements have done on this. Still these are points of concern.

1.5 Functions of IDPS

There are different types of IDPS on the basis of types of events recognize and methodologies used to identify incidents. Besides monitoring and analyzing undesirable activity, following functions are performed by all types of IDPS [2].

- **Record information about observed event**: Information is recorded locally usually, but it can also be stored on centralized logging servers, enterprise

management systems and security information and event management (SIEM) solutions.

- **Notifying security administrators**: When attack has been detected notification send to administrator known as alert. Alerts can send using several methods including e-mails, message on interface, pages, syslog messages, using programs and scripts. Notification is the basic information provided to administrator, for more information need to access the IDPS database and technology.
- **Producing reports**: Reports provide summarize details on particular events of interest after analyzing monitored events.

Above functions are provided by both IDS and IPS but IPS provide some additional functions to prevent from the attack.

- **The IPS stops the attacks:** The IPS can stop the attack by resetting or terminate currently used connection or session. IPS can block access from the offending IP address, user account to the targeted host, services and applications.
- **The IPS changes the security environment**: The IPS can reconfigure a network device like firewall, router, and switch to block the access. Some IPSs are able to apply the patches after detecting the vulnerability.
- **The IPS change the attack's content:** IPSs can change or delete the malicious content of the network traffic to make it benign. For example IPS can remove the malicious file attachment from the e-mail.

1.6 Detection Methodologies

IDPS use many methodologies but main three detection methodologies are used anomaly based, signature based and Stateful protocol analysis.

1.6.1 Anomaly-based IDPS

Anomaly-based detection method compares definitions and data of legitimate user with events that show deviations from their normal behavior. Anomaly-based IDSs generate profiles by monitoring the characteristics of network traffic and activity over a period of time known as machine learning techniques [2]. These systems detect intrusions by comparing characteristics of activity with the thresholds belongs to that

profile, like detecting when web activity uses more bandwidth than expected bandwidth, user access a single database record is legitimate but IDPS generate an alarm of anomaly to administrator when more number of records are accessed. Advantage is that Anomaly-based IDPS are able to detect Zero-day attacks. [1] But these systems must be configured carefully to recognize expected behavior. This method has high false positive rate and suffer from slow speeds.

1.6.2 Signature-based IDPS

A signature is a pattern of attack that is predetermined. Signature-based detection systems compare signatures with the monitored network traffic. After finding a match IDPSs generate alarm and sometime take appropriate action. Signature can be of two types whether vulnerability-based or exploit-based. Vulnerabilities based analyze vulnerability in program execution and conditions needed while exploit-based analyze patterns of known attacks. Signatures can be present any part of the network packet means on any protocol header according to the nature of attack. Most of the IDPS are implemented on signature-based techniques because of its low false positive rate, fast speed and accuracy [1]. In addition, signatures are created manually by the security analyst who makes it more accurate. But they are not able to detect new emerging attacks means zero-day attacks. Continuous creating, updating and Tuning are required in this technique [3].

1.6.3 Stateful Protocol analysis detection

Stateful protocol analysis is process of examining protocols of network, transport and application layers like TCP and UDP protocols. These protocols contain HTTP, FTP, SMTP and SNMP protocols. These systems must know how these protocols are supposed to work [2]. It can track the state of the protocols. For example in FTP protocol user start with unauthenticated state so these IDS keep on checking the privileges to the unauthenticated users. These are also capable to detect attack with in protocol application data. Example HTTP protocol analysis helps to detect URL obfuscation in which URL uses hex encoding to look different but has same meaning. By using hex decoding operations examine the URL. Disadvantage is that these are resource intensive because of lots of overheads for state tracking [1]. These are not able to detect attacks that show normal characteristics of accepted protocol behavior.

1.7 IDPS Techniques

IDPS techniques can be classified into various types based on events monitored and the ways of deployment.

1.7.1 Network based IDPS

Network-based IDPS sensors capture traffic of particular network segment or device to analyze the content and monitor application protocol activity to detect malicious activity. It identifies and stops known and unknown attacks. DDoS and Unauthorized access are the main threats to networks. Network-based IDPS analyses the activities to find threats at application layer, transport layer and network layer efficiently. At application layer it can analyze various protocols like DHCP, FTP, HTTP, Telnet, Server Message Block and SMTP as well many database protocols. At transport layer it can identify port scanning, packet fragmentation and SYN flooding attack [6]. At network layer it can prevent from IP spoofing, Illegal IP header value by in-depth analysis. IDPS sensors commonly deployed at a division between networks, such as border firewalls or routers, remote access servers, and virtual private network (VPN) servers. Network-based IDPS is cost-effective and it is not visible on the network. NIDPSs are the operating system independent. Snort is the example of Network-based intrusion detection and prevention systems.

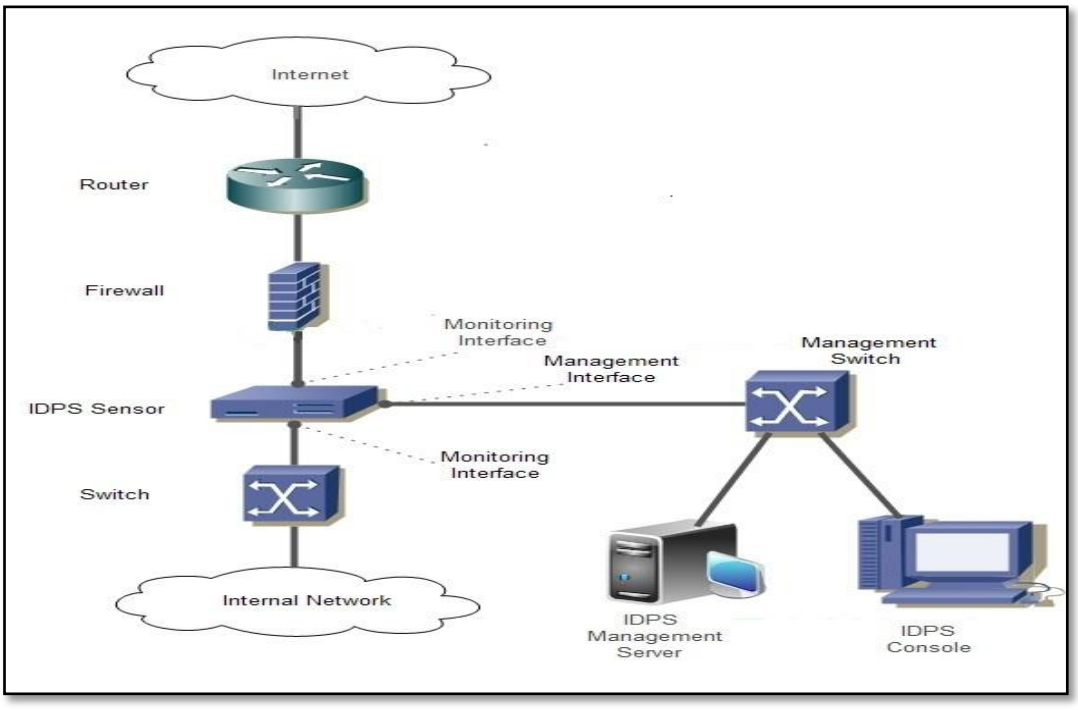


Figure1.4: Network based IDPS architecture

Network based intrusion detection systems have some *limitations.* Attacks within encrypted network traffic cannot be detected by NIDPS. For this detection either IDPS should analyze traffic before encryption of after the decryption. It is unable to handle or analyze high loads. They drop some low priority packets to handle high loads and attack goes undetected [1]. NIDPSs are also susceptible to DDoS attacks, anomalous attacks to crash the sensor system and blinding attacks that generate many alerts in very short time against themselves.

1.7.2 Host-based IDPS

A host-based IDPS agent employed to protect computer systems containing crucial data against threats like viruses and malware. HIDPS detect both known and unknown attacks against a particular host. Host-based IDS agents are installed on every physical device in the network. HIDPS monitor wired and wireless traffic, file access, log files, running processes and any changes in system configurations [2]. HIDPS keep on checking the characteristics of host and the various activities that occur within the host. It can be implemented on servers, workstations, and computers. For every object in question, the HIPS remember each object's attributes and creates a checksum for the contents [8]. This information gets stored in for later comparison. The system also checks whether appropriate regions of memory have not been modified. Host-Based IDPS provides a several security capabilities. It uses combinations of several detection and prevention techniques and it is capable of achieving more accuracy as each technique can monitor different characteristics of the host. Whether the attack is successful or failure can be determined by host-based IDS. HIDPS does not susceptible to fragmentation attacks or variable Time to Live (TTL) attacks. HIDPS can detect attack within HTTPS, VPN, SSH encrypted protocols because it has access to the traffic after decryption.

Host-Based IDPS has some *limitations.* Some techniques are performed only periodically such as hourly or a few times a day. It forwards its alert data to the management server which can result in a delay to respond. It may impact the host’s performance as Host-Based Intrusion Prevention System is running on the host and use the host’s resources [1]. Furthermore, the installation of an agent may cause conflicts with existing host security controls. Host-Based Intrusion Detection and Prevention Systems do not provide a complete picture of the network.

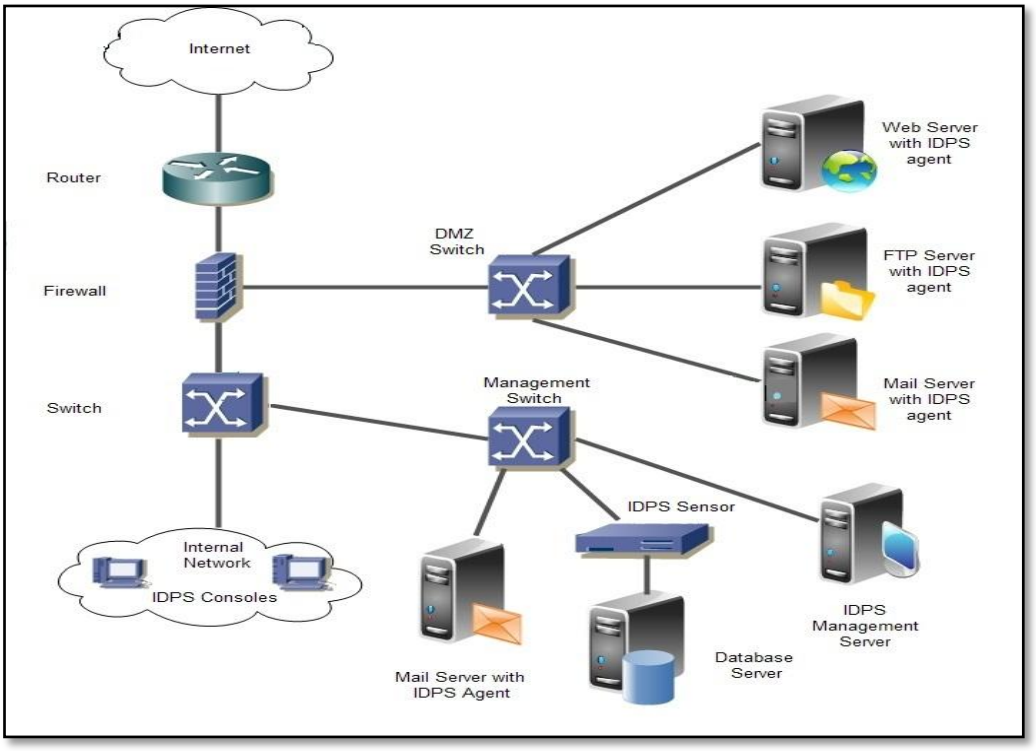


Figure1.5: Host based IDPS architecture

1.7.3 Wireless IDPS

A wireless intrusion detection and prevention system (WIDPS) monitors the radio spectrum means wireless media to detect the access points that are not authorized for the network. WIDS analyze wireless traffic and wireless protocols. Wireless is not similar to the wired network because of its shared nature wireless is more susceptible to the security threats. Wireless local area network (WLAN) uses the IEEE 802.11 standard. These standard WLANs have two components, Stations (STA) and Access Point (AP). Stations are the endpoints of wireless network like laptops, mobile phones, Tablets, computer and PDA. Access points are used to connect stations with a distribution system (DS). Further DS connect the stations to the external network and with the internet.

IEEE 802.11 defines two WLAN architectures, Ad Hoc Mode and Infrastructure Mode. In ad hoc mode APs are not used. All stations communicate directly with each other. One station act like AP to other station. It is also known as peer-to-peer connection. Infrastructure mode use APs to connect STAs with DS. Every endpoint device identified by its MAC address and wireless network with a service set

identifier (SSID). There are many threats to the wireless network. Jamming and DDoS attacks are identified by Stateful protocol analysis or by anomaly based techniques [2]. To detect these attacks WIDPS sensor is deployed either with AP or with Wireless switch. Man-in-the-middle attack is performed through deploy a rogue access point. WIDPS are able to detect Rogue AP that is implemented on the network where it is not authorized to work. WIDPS with the help of information gathering can identify the difference between external AP and Rogue AP.

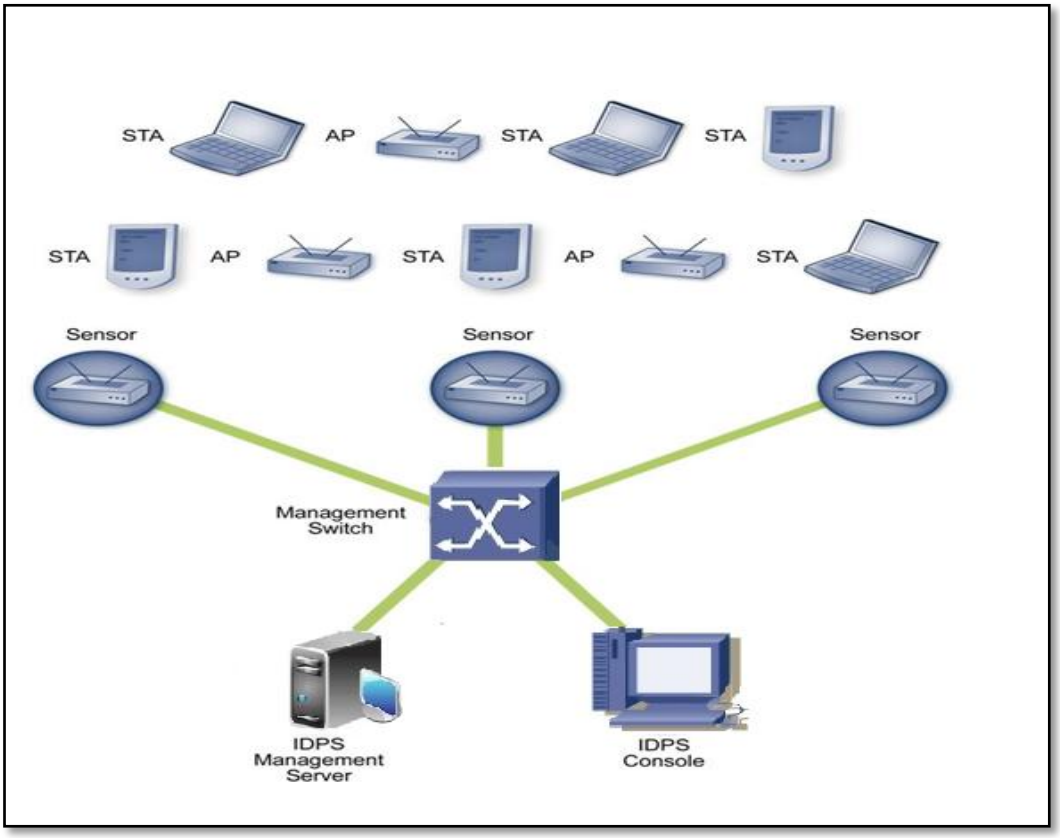


Figure1.6 Wireless IDPS architecture

Wireless network is unable to detect wireless protocol attack and evasion techniques. If an attacker attack two channel at the same time then the IDPS sensor detect attack on one channel but another attack goes undetected. Wireless Intrusion Detection and Prevention Systems (WIDPS) are also susceptible to the threats to itself. DDoS attack is very common to Wireless Intrusion Detection System. DDoS is the attack on availability of the IDPS.

Chapter 2

Literature Survey

Rapid increase in network traffic for past few years leads to more security breach in network. Previous protective measure like Packet filtering Firewall is not capable against such network security attacks. Therefore many Intrusion Detection and Prevention Systems are developed among them Snort is world widely used IDPS.

2.1 Snort IDPS Tool

Sourcefire has developed a network based intrusion detection and prevention system (IDS/IPS) named as Snort which is an open source program [10]. As Snort combines the advantages of protocol, signature and anomaly-based inspection, it is most popular worldwide. Snort uses rules that can be modified. Rules are stored in separate files according to category of rule. Snort has three main modes to be configured: sniffer, packet logger, and intrusion detection. In sniffer mode, Snort captures the traffic and display the content to the console. In logger mode, Snort creates logs of packets to the disk. In intrusion detection mode, Sensor monitors the network traffic and compares the content against a rule set [3]. On the basis of detection Snort performs a specific action. Snort compare malicious activities with these rules and store output on MYSQL database, and use Apache web server and PHP script as shown in Figure 2.1.

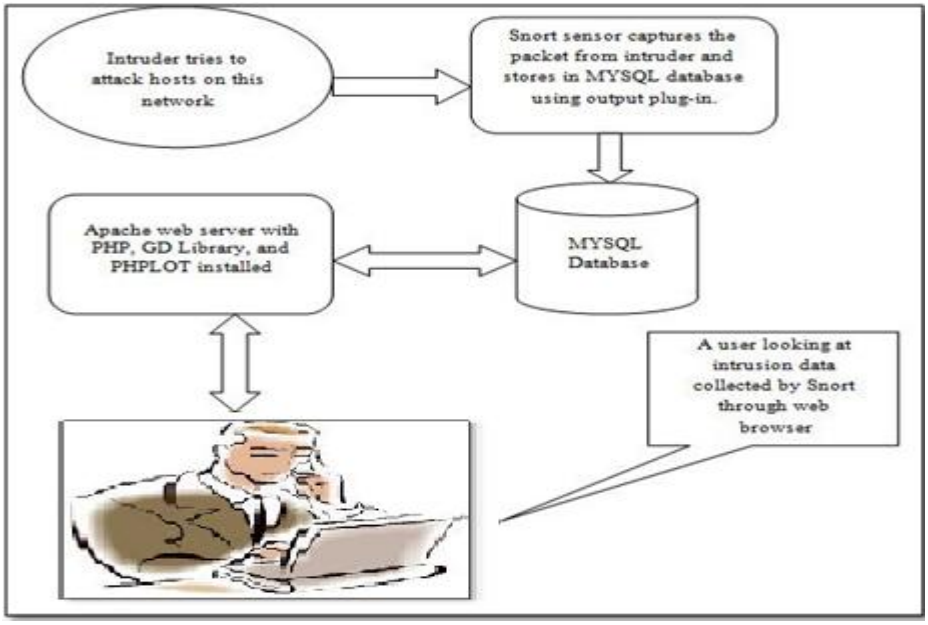


Figure 2.1 Block Diagram of Snort NIDPS

2.1.1 Components of Snort IDPS

Snort is divided into different components. For intrusion detection all these components work together and generate output. First of all packet decoder receives the data packet coming from the web servers. Based on Detection engine analysis packet is either dropped, logged or generate an alarm on its way towards the output modules. Arrangement of these components is shown in Figure 2.2.

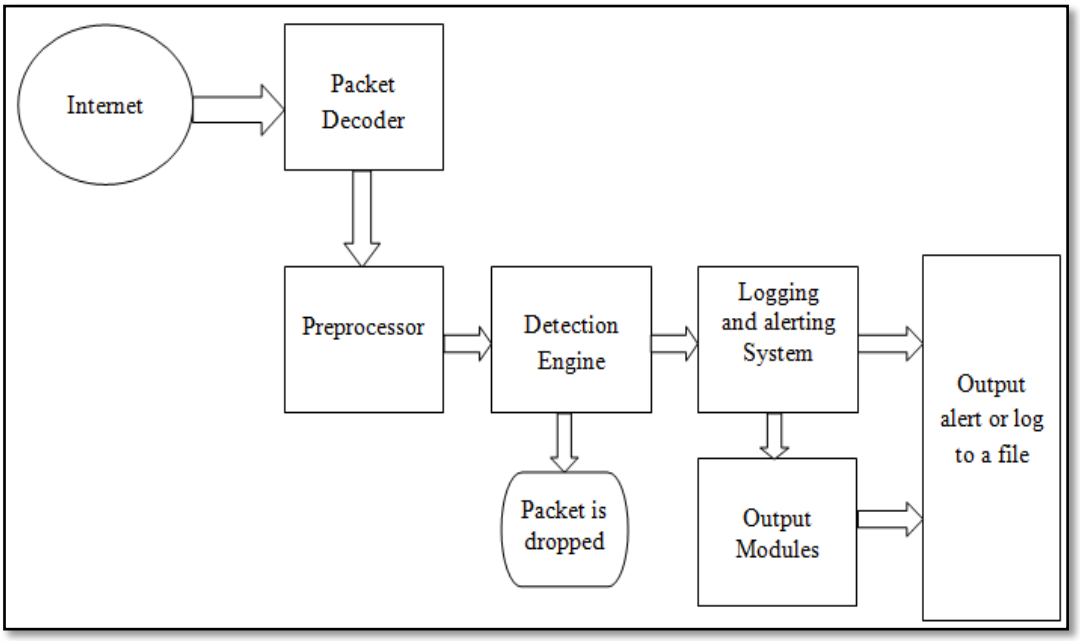
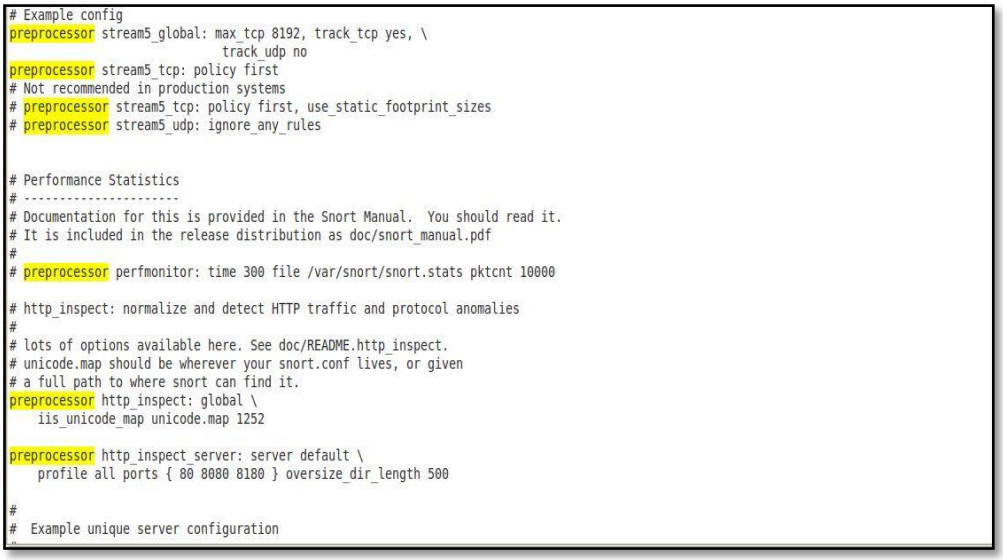


Figure 2.2 Arrangements of Components of Snort

Packet Decoder: Snort captures the data link layer frames via libpcap which is Unix based program in Linux operating system. For implementation in Windows it uses Winpcap. After capturing the traffic these files are sends to Snort decoder that creates the structures of all the protocols that are used to create traffic [10]. These structures are called packets that send further to the preprocessor component of snort.

Preprocessors: When Snort received a packet, it may not be processed by Snort's detection engine. So preprocessors are used for providing two functions. First they prepare packet for the detection engine for analyzing and second the anomaly based detection is perform through preprocessor. Preprocessor examine traffic for suspicious activities on the basis of created profiles. It detects attacks that are not detected by matching signatures. There are many preprocessor in Snort like Http_inspect which is used to detect URL obfuscation attack of known threats. It enables snort to read or write the hexadecimal values in the URL. Preprocessor stream4 provides two

functions, TCP stream reassembly and Stateful inspection of the traffic. Similar to detect port scanning and ARP spoofing attacks portscan and arpspoof preprocessors are used respectively [3]. All the preprocessors are added to the configuration file of Snort. The following Figure shows the preprocessor added to configuration file.



```
# Example config
preprocessor stream5_global: max_tcp 8192, track_tcp yes, \
                             track_udp no
preprocessor stream5_tcp: policy first
# Not recommended in production systems
# preprocessor stream5_tcp: policy first, use_static_footprint_sizes
# preprocessor stream5_udp: ignore_any_rules


# Performance Statistics
# ----------------------
# Documentation for this is provided in the Snort Manual.  You should read it.
# It is included in the release distribution as doc/snort_manual.pdf
#
# preprocessor perfmonitor: time 300 file /var/snort/snort.stats pktcnt 10000

# http_inspect: normalize and detect HTTP traffic and protocol anomalies
#
# lots of options available here. See doc/README.http_inspect.
# unicode.map should be wherever your snort.conf lives, or given
# a full path to where snort can find it.
preprocessor http_inspect: global \
    iis_unicode_map unicode.map 1252

preprocessor http_inspect_server: server default \
    profile all ports { 80 8080 8180 } oversize_dir_length 500

#
#  Example unique server configuration
```

Figure 2.3 Preprocessor written in snort configuration file

Detection Engine: After preprocessing the packet is pass to the heart of signature based IDPS that is detection engine. At Detection engine a pattern matching algorithm which is the core of detection engine is implemented to match the signatures. Most of the time is spent on the signature matching. Signatures come from the set of rules. Detection engine parse the rules line by line [10]. User can also pass its rules to detection engine. Snort's rules have header and option two parts. Header contains the information about protocol IP addresses and port numbers. The option section contains the signatures. Snort rule can be written as follow.

```
Alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg: " ICMP L3retriever
Ping"; icode:0; itype:8; content:"ABCDEFGHIJKLMNOPQRSTUVWABCDEFGH"
; depth:32; reference:arachnids,311; classtype: attempted-recon; sid:466; rev:4)
```

Figure 2.4 Snort rule example

In the rule the source is on the left side of the arrow and destination is on the right side. The "any" keyword specify that source can be any for any port number. '$' sign

put the users ip address in the rule at runtime. Option field has message to display and which content is to be matched and also specify the flow of the traffic.

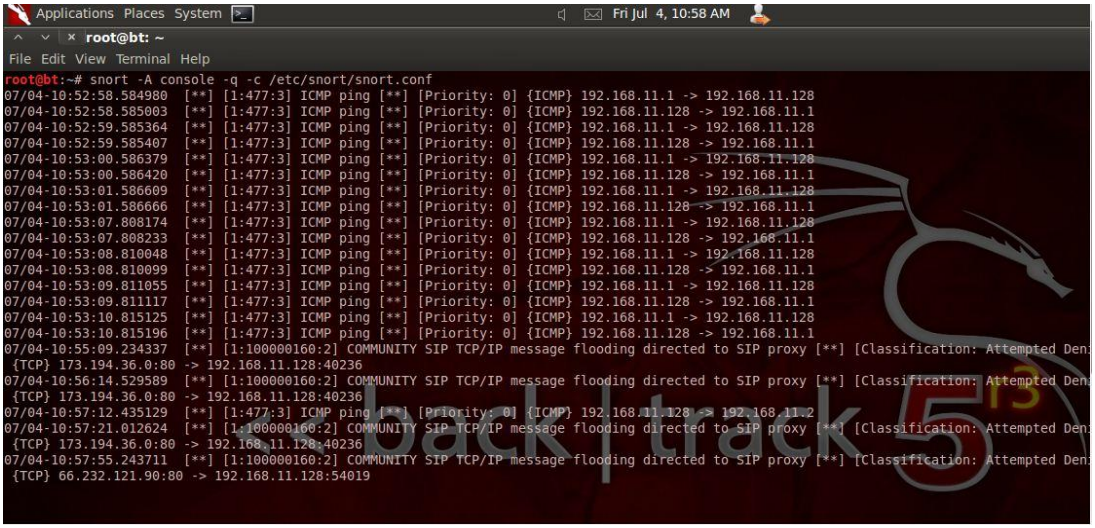
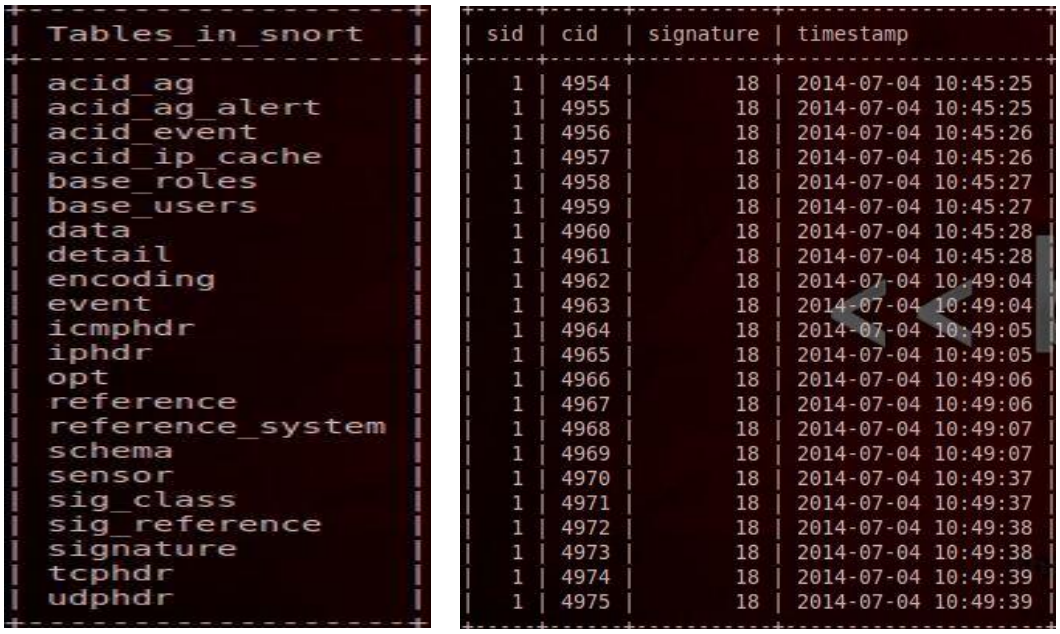


Figure 2.5 Snort capturing ping

Logging and alerts: Snort syslog module keeps logs messages to the syslog utility. After detecting attack snort generates alarm to the security administrator about the attack. SNMP module is used to send alarms to the management server similar SMB module send pop-up alerts for Microsoft Windows system.



Tables_in_snort
acid_ag
acid_ag_alert
acid_event
acid_ip_cache
base_roles
base_users
data
detail
encoding
event
icmphdr
iphdr
opt
reference
reference_system
schema
sensor
sig_class
sig_reference
signature
tcphdr
udphdr

sid	cid	signature	timestamp
1	4954	18	2014-07-04 10:45:25
1	4955	18	2014-07-04 10:45:25
1	4956	18	2014-07-04 10:45:26
1	4957	18	2014-07-04 10:45:26
1	4958	18	2014-07-04 10:45:27
1	4959	18	2014-07-04 10:45:27
1	4960	18	2014-07-04 10:45:28
1	4961	18	2014-07-04 10:45:28
1	4962	18	2014-07-04 10:49:04
1	4963	18	2014-07-04 10:49:04
1	4964	18	2014-07-04 10:49:05
1	4965	18	2014-07-04 10:49:05
1	4966	18	2014-07-04 10:49:06
1	4967	18	2014-07-04 10:49:06
1	4968	18	2014-07-04 10:49:07
1	4969	18	2014-07-04 10:49:07
1	4970	18	2014-07-04 10:49:37
1	4971	18	2014-07-04 10:49:37
1	4972	18	2014-07-04 10:49:38
1	4973	18	2014-07-04 10:49:38
1	4974	18	2014-07-04 10:49:39
1	4975	18	2014-07-04 10:49:39

Figure 2.5 Shows Snort databases that contain many tables and shows event table information

Output Modules: Snort keeps log of the detected attack in the MySql database. Their date, time, severity, ip address and many more details are stored in database. For generating alarm Snort use these stored values. Snort use ACID that implemented using PHP on apache server. This output webpage shows all the details to the user any time. Snort can also save output to the XML or CSV files.

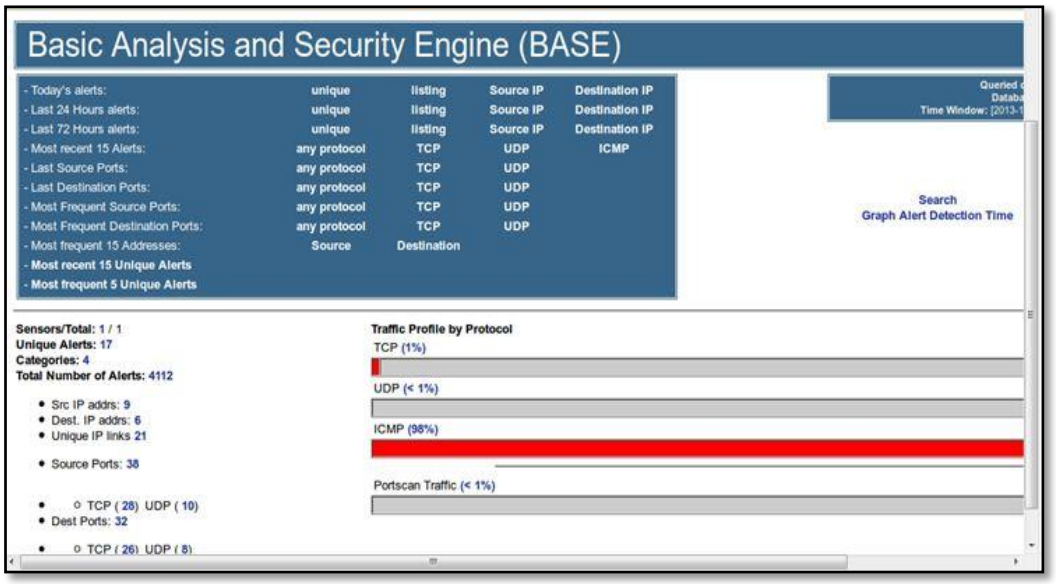


Figure 2.6 Snort attack output on BASE

Snort now acts as Intrusion Prevention system when deployed at network boundary. It can take action based on intrusion detection. As traffic is passed through IPS sensor therefore the analysis of the traffic must be fast enough as there will be less delay for the traffic to reach its destination. Intrusion Prevention Systems protect valuable assets in real time environment by rejecting or dropping the packets [29]. In 2000 there were 854 rules in Snort which is the IDPS tool out of which 786 rules use pattern matching for intrusion detection [27]. As cyber crimes are increasing day by day, the signatures of the attack are also increases. In 2003 rules becomes 2000 in number but in 2006 number of rules reach to 6000 while in 2007 there is a database of rules which contain 7856 rule in Snort [11]. It shows that attacks are increasing very rapidly so detection of these attacks before they actual performed on the information system is an important part of security. Snort needs some improvements on ability to preprocess traffic to detect anomaly based attacks, a hardware to reduce the load and better pattern matching algorithm for detection.

2.2 Pattern Matching

The Pattern Matching in Intrusion detection system is a problem to find the first occurrence of patterns or keywords for the attacks which are of any length in a stream of data packets that cause attack. For detecting the attack pattern should be exact which is known as Exact Pattern Matching. In Computer Science Pattern Matching algorithms solve the problem to find the keyword in the given input text [9]. Text T is a sequence of characters. An alphabet $\sum$ is the set of possible characters for a string. For example {a, b, c} is the set of alphabets that are used in a string then "abbcbac" is a string over the set. Pattern P is a string of length m then prefix of P is substring of type P [0…i] and suffix of P is a substring of type P [i……m-1]. So the problem consists of finding a substring of T equal to P. For example

Text: Buffer overflow attack is performed.
Pattern: attack

Then find the occurrence of pattern 'attack' in the text 'Buffer overflow attack is performed' is known as Exact Pattern Matching. Various algorithms used different techniques to find pattern in the text string.

The **Brute-Force** algorithm is also known as Naive search algorithm need no pre-processing and use constant space to keep information of current position. This algorithm searches the pattern through character by character matching. The brute force algorithm first compares every character in T with the first character in P then with the second and so on until the last character in P is reached [9]. This algorithm shifts the pattern to the right by exactly one position after a mismatch. The following example illustrates how naive algorithm searches the pattern.

Text is AAAAAAAAHAAAAAAAAAA
Pattern is AAAAH

Step1. AAAAAAAAAHAAAAAAAAAA
AAAAH 5 comparisons made
Step2. AAAAAAAAAHAAAAAAAAAA
AAAAH 5 comparisons made
Step3. AAAAAAAAAHAAAAAAAAAA
AAAAH 5 comparisons made

Step4. AAAAAAAAHAAAAAAAAAAA

AAAAH 5 comparisons made

Step5. AAAA**AAAAH**AAAAAAAAAAA

AAAAH 5 comparisons made

In the example there is mismatch at the last character of the pattern so there are five comparisons on every step. It is the worst case example of brute force algorithm. In best case that is pattern found at first substring then the running time is equal to length of pattern.

Michael O. Rabin and Richard M. Karp came up with the idea of hashing of the string in 1987 [12]. This pattern matching algorithm calculates a hash value for the pattern, and compares it with the hash value of same length subsequence of text string. If calculated hash values are not equal, the algorithm will calculate the hash value for next same length subsequence, therefore there is only single comparison per text subsequence. But if the hash values are equal, the algorithm will compare the pattern with that subsequence by following naive search algorithm. This algorithm is known as **RabinKarp** algorithm.

Hash value of "AAAAA" is 37.
Hash value of "AAAAH" is 100.

Step1. AAAAAAAAAHAAAAAAAAAA

AAAAH 37≠100 1 comparison made

Step2. AAAAAAAAAHAAAAAAAAAA

AAAAH 37≠100 1 comparison made

Step3. AAAAAAAAAHAAAAAAAAAA

AAAAH 37≠100 1 comparison made

Step4. AAAAAAAAHAAAAAAAAAA

AAAAH 37≠100 1 comparison made

Step5. AAAA**AAAAH**AAAAAAAAAA

AAAAH 100=100 6 comparisons made

In the above example text and pattern are same as in Brute-force search example but in this the total comparisons made are 10 unlike brute force where comparisons were 25 in number. So in this worst case RabinKarp is faster than Brute-force.

D. Knuth, James H. Morris and V. Pratt discovered an algorithm in 1977 [13]. They found first linear time pattern matching algorithm by analyzing the brute force search. This algorithm maintains information gathered during scan of the text that naive approach wasted. For this it calculate the prefix function. Prefix function find all prefix values of the pattern at the maximum that match the suffix of itself. It creates a table that stores the position where from where pattern must be shift. It helps to avoid backtracking on the text 'T'. This algorithm is named as **Knuth-Morris-Pratt** (KMP) algorithm. Following example shows how this algorithm matches the pattern.

Text is ABACAABACCABACABAAD

Pattern is ABACAB

Table 2.1 Shift table of KMP

Position	0	1	2	3	4	5
Pattern	A	B	A	C	A	B
Shift position	-1	-1	0	-1	0	1

Step1. ABACAABACCABACABACD
ABACAB 6 comparisons made

Step2. ABACAABACCABACABACD
ABACAB 1comparison made

Step3. ABACAABACCABACABACD
ABACAB 5 comparisons made

Step4. ABACAABACCABACABACD
ABACAB 1 comparison made

Step5. ABACAABACC**ABACAB**ACD
ABACAB 6 comparisons made

In the following example pattern 'abacab' is searched in string 'abacaabaccabacabac'. It creates a table to calculate prefix that matches with suffix. In this {'a', 'ab'} are the prefix values that match with suffix. Therefore mismatch at position five shifts the pattern forward with value four.

Robert S. Boyer and J Strother Moore discovered an algorithm that searches for the index of the first occurrence of pattern in the text in 1977[14]. **BoyerMoore** compares the pattern with text starting from right to left of the substring. This algorithm uses two heuristics for creating valid shift tables. Two techniques are explained follow.

Bad character heuristic: According to this if there is any alphabet for that no match occurs somewhere else in the pattern then the pattern can be aligned to this text alphabet. Another on mismatching with the alphabet that does not present in the pattern at all will shift the pattern with valid shift of equal to the size of pattern.

Good suffix heuristics: This approach work like KMP algorithm Pre-processing function. If suffix is matched then the pattern can be shifted until the next occurrence of suffix in the pattern is aligned to the text alphabets exactly same as suffix.

Maximum value between these two table values is selected for the shift. Table 2.2 shows the shifts calculated by using above techniques.

Text is ABACAABACCABACABAAD
Pattern is ABACAB

Table 2.2 Shift table of Boyer-Moore

Position	0	1	2	3	4	5
Pattern	A	B	A	C	A	B
Bad character	A	B	C	D		
Shift	1	4	2	6		
Good Suffix	A	B	A	C	A	B
Shift	9	8	7	6	7	1

Step1. ABACAABACCABACABACD
ABACAB 1 comparison made
Step2. ABACAABACCABACABACD
ABACAB 3 comparisons made
Step3. ABACAABACCABACABACD
ABACAB 1 comparison made

Step4. ABACAABACCABACABACD

ABACAB 4 comparisons made

Step5. ABACAABACC**ABACAB**ACD

ABACAB 6 comparisons made

In the example Boyer-Moore algorithm creates two tables one on the basis of bad character shift and another on the basis of prefix values. Maximum between two values is selected for the shift. As in the Step2 there is a mismatch on the fourth position of pattern, as there is bad character shift is 1 where as good suffix shift is 6 so therefore mismatch character is shifted to the tenth position.

R. A. Baeza-Yatesand and G. H. Gonnet [15] introduced a bit parallel shift technique in which state is represented by a number and on every step arithmetic and logical operations are performed provided that number should be large enough to represent all the states. In this technique time complexity for the patterns less than word size of computer used is linear. This algorithm use $O(\sum)$ extra memory space and $O(m + |\sum|)$ preprocessing time but L. Colussi [16] proposed algorithm which is obtained by applying three improved steps on naive search algorithm. The first step improved the algorithm as KMP algorithm whose worst case is 2n and average case take $(1+\alpha)n$. Next steps improved as Boyer-Moore as 1.5n character matching in the worst and sub-linear on random text [17]. It shows that it is faster than Boyer-Moore which takes 2n comparisons.

A. Alfred V. and J. Margaret C. proposed a pattern matching algorithm to match the set of patterns with the text [18]. It is the extension of the KMP algorithm. This algorithm combines all elements of a set into a syntax tree. After this tree is converted into non-deterministic automaton (NDFA) and then into a deterministic automaton (DFA). Every state has a pointer to the each alphabet character and list of the patterns that are matched. In preprocessing it computes failure values like in KMP. Failure pointers point to prefix of that node. Matching time complexity is O(nlogn). B. Commentz Walter came with the idea of combining the BoyerMoore with the Aho-Corasick algorithm [19]. In this algorithm DFA is created by matching from rightmost character of the text using right to left approach. For matching the pattern with the text it uses the matching window size equal to the length of pattern. Start comparing pattern from right side on the substring of the text. On mismatch it shifts the pattern

with pre computed valid shift table value. S. Wu and U. Manber proposed an algorithm which use only BoyerMoore algorithm [20]. For precomputation it uses only bad-character shift. In this algorithm text is process by matching blocks of text equal to k rather than one by one character. It uses the hashing to index the patterns. Three tables are created in this Shift table, hash table and the prefix table. This algorithm is good for medium number of patterns but not performed well on the large number of patterns. But Y.Jingbo, J. Zheng, and S. Ding. present an algorithms by analyzing BM algorithm and relative algorthims [28]. The proposed algorithm uses the combination of the last character and the next character of the substring and use the singleness to increase the shift value. This algorithm improves the Wu Manber efficiently.

M. Elhad defined that existing algorithms LCS have drawbacks of high computational time so proposed refinements on LCS. There are two refinements on LCS. First refinement uses flip property which speed up to 50% on its average case [21]. As complexity of lcs is O(mn) refinement reduce time complexity half of O(mn). Another refinement is done on worst case which uses split property which speed it also upto 50%. L. Arockiam [22] depict that the problem in the LCS algorithm is that it compare both matched and unmatched positions so new modifications was proposed which focus on matched positions. It stores the left positions that are matched stored in a different array to process. Therefore it reduces the time complexity.

Zhengqiang, proposed a modification in IDS string matching algorithm [23]. Modification divides the different modes into group and implement bitmap match for less than three bytes. So Modification increase the speed of algorithm for pattern matching and pattern is up to three bytes. Aldwairi M. et. al [24] proposed the exclusion-inclusion filter is implemented for fast searching which is modified bloom filter. This filter is used to reject clean traffic and pass malicious traffic to the Intrusion Detection System. In this way it increase the speed of IDS 3.4 times on average and normal traffic speed is 5.5 times but for worst case traffic it increase the speed 1.6 times.

P. Brodanac, L. Budin and D. Jakobovic proposed a pattern matching algorithm by parallelized RabinKarp [25]. In this technique text string is divided into k equal parts

and equally k threads are implemented for searching the pattern. As RabinKarp use hash matching technique then no need to calculate hash of pattern again for every thread. This algorithm performed well for large text strings and large files. This parallelization method made the RabinKarp three times faster than the serial method. But for the short string it takes more time because of creation of processes. B. Fuyao, G. Zhao, and L. Qingwei [30] uses hashing technique. In this hash value of strings called fingerprint compared rather than the characters directly. This approach considers the bitwise operation and work on size of the alphabet and the pattern length. In this way spurious hits are reduced and run time is linear.

H. Le and V. K. Prasanna proposed an algorithm for Network Intrusion detection system that uses tree search data structure for resulting set of post processed patterns [26]. This algorithm is known as “leaf-attaching” that preprocess dictionary without increasing the number of patterns. This algorithm achieved efficiency 0.56 for Roget’s dictionary and 1.32 for Snort dictionary. They implement it on 45 nm ASIC and a state-of-the-art FPGA and achieved 24 and 3.2 Gbps respectively. It process input of prefix tree and produce output a set of disjoint patterns. Dictionary update involves simply rewriting the content of the memory, which can be done quickly without reconfiguring the chip.

Chapter 3

Problem Formulation

3.1 Problem Definition

The IDPS has gain popularity because it is used in organization as a component of security model. It is a great defense for the system and the network against attacks. But due to increase in speed and size of the network traffic, vulnerabilities are also increases. Because attacks are increased, that results into increase in the number of signatures of the attacks. So to match large database of signatures with the content of network captured traffic before attack occurred to the system is the problem of the signature based IDPS. If attack goes to the system before detection then taking preventive measures for the attack will become difficult. Therefore attack detection on time is necessary for the Intrusion detection and prevention systems. To match the signatures with the file, pattern matching algorithms are used that are the core of IDS. An IDPS spent most of its running time on pattern matching. Therefore to develop or select the fast pattern matching algorithm is the exact problem of the existing signature based intrusion detection systems. Pattern matching is the problem to match the pattern string with the substring of text equal to the length of pattern size. For intrusion detection systems pattern matching is the problem to match the large number of different size patterns with the text. Therefore run time complexity of algorithm increase with the factor of number of patterns.

3.2 Objectives

1. To study and explore various pattern matching algorithms.
2. To divine and develop each selected algorithms.
3. To analyze the performance of those algorithms based on number of patterns and size of the file.
4. To demonstrates working of these pattern matching algorithms.

Chapter 4

Design and Implementation

4.1 Algorithm used

Various Pattern Matching Algorithms are implemented. All the algorithms have a different technique to search for patterns in the given file. Their pseudo code is as follow:

4.1.1 Brute-Force Search

NAIVE-SEARCH (T, P)

Input: T [1…….n] is text string; P [1…….m] is a pattern string
Output: Starting index of the substring of T matching P or -1

```
1.  for i ← 0 to n-m do
2.              j ← 0
3.      while j<m && P[j]==T[ i+j] do
4.              j ← j+1
5.              If j==m return i        //match occur
6.  return -1                   // match unsuccessful
```

This algorithm is very easy to understand. It checks all the positions from 0 to n-m that pattern start from there if yes then it matches second, third and so on letters of character with the substring of text. In the above pseudocode there is no pre-processing for the text or pattern. The worst case run time complexity is O(m(n-m+1)) because outer loop running in n-m+1 times and inner loop running m times. In the best case where pattern starts at first position in the text has only O(m) comparisons.

4.1.2 Rabin-Karp Algorithm

RABIN-KARP (T, P)

Input: T [1…….n] is a text string; P [1…….m] is a pattern string
Output: starting index of the substring of T matching P or -1

```
1.  h ←d^(m-1) mod q
2.  p ← 0
3.  t0 ← 0
     // Pre-processing
4.  for i ←1 to m do
5.       p ← (d p + P[i]) mod q
6.       t0 ← (d t0+T[i]) mod q
    // Searching
7.  for s ← 0 to n-m
8.       If p == ts                    //hash value match occur
9.            If P[1…m]==T[s+1…….s+m]
10.                return s    // match occur
11.       If s < n-m
12.       ts+1 ← (d(ts-T[s+1]h) + T[s+m+1]) mod q
13.  return -1                    //match unsuccessful
```

Rabin-Karp first computes the hash value of the pattern using mathematical calculations. Sometimes hash value of one string match with the hash value of another string that hit is known as spurious hit. Spurious hit is removed by checking string character by character matching. Preprocessing time complexity is computed as O(m). Its worst case matching time is O((n-m+1)m) when spurious hit occur on every step. So total run time complexity is O((n-m+1)m)+O(m). In average case matching complexity is O((n-m+1) + cm) = O(n+m) where c is constant represent number of spurious hit. So total average case complexity is O(m)+O(n+m) = O(n+m).

4.1.3 Knuth-Morris-Pratt Algorithm

KMP-Matcher (T, P)

Input: T [1……n] is an array of text; P [1…….m] is an array of pattern

Output: Starting index of the substring of T matching P or -1

```
1.  f ← KMP-Failure-Function(P)    //build failure function
2.  i ← 0
3.  j ← 0
4.  while i < n && j<m do
5.       if P[j] == T[i]then
6.            i ← i + 1
7.            j ← j + 1
8.       else if j == 0
```

```
9.              i ← i + 1
10.     else   j ← f(j-1)      // After matching prefix in P
11.  if j==m return i-m              // Match occur
12.  return-1                  // Match unsuccessful
```

Knuth-Morris-Pratt algorithm use failure function which computes the value of valid shift of the characters of pattern by matching against pattern itself. This information helps to move pattern with more valid shifts rather than moving one position to right.

KMP-Failure-Function (P)

Input: P [1……m] an array of pattern

Output: Return a table of failure function for P

```
1.  table [0] ← -1
2.  for j ← 1 to m-1 do
3.       i ← table [j-1]
4.       while P[j] != P[i+1] && i>=0 do
5.              i← table (i)
6.       if P[j] == P[i]         // Matched i + 1 characters
7.              table[j] ← i + 1
8.       else  table[j] ← -1
9.   return table
```

The KMP algorithm take preprocessing time complexity is O(m) where as matching time is O(n) therefore its total time complexity is O(n) + O(m) = O(n+m).

4.1.4 Boyer-Moore Algorithm

BMMatcher (T, P)

Input: T [1……n] is an array of text; P [1…….m] is an array of pattern

Output: Starting index of the substring of T matching P or -1

```
1.  i ← m - 1
2.  j ← m – 1
3.  charTable[] ← BAD_CHARACTER_SHIFT_BM(P);
4.  offsetTable[]  ← GOOD_SUFFIX_SHIFT_BM(P)
5.  while i < n-1 do
6.       if P[j] == T[i] then
```

```
7.              if j == 0 then
8.                      return i                        //Match occur
9.              else i ← i - 1
10.                     j ← j - 1
11.     else    i ← i + max(j - Last(T[i]), Match(j))   // shift P relative to T
12.  return -1
```

BoyerMoore algorithm uses two heuristics. First it shift the pattern according to bad character shift in which if the character of pattern mismatch with the character of the text that is not present at all in the pattern shift the pattern equal to its length. Second is good suffix heuristic in which it checks the prefix of pattern that matches with the suffix of pattern itself. According to this it calculates the valid shifts of the pattern. Both the functions return a table. Pseudo code of both the algorithm is given below.

BAD_CHARACTER_SHIFT_BM (P)

Input: P [1…….m] is a pattern string

Output: Return a table of all alphabets with valid shifts

```
1.  alphabet_size ← 256
2.  table ← new array[alphabet_size].
3.  for i ← 0 to alphabet_size-1 do
4.      table[i] ← m
5.  for i ← 0 to m-1 do
6.      table[P[i]] ← m-1-i
7.  return table
```

GOOD _SUFFIX_SHIFT_BM (P)

Input: P [1…….m] is a pattern string

Output: Return a table of patterns with valid shifts

```
1.  table ← int array[m]
2.  lastPrefixPosition ← m
      // check if i+1 a prefix of pattern
3.  for i ← m-1 to 0
4.  k ← i+1
5.     while( k < m)
6.     j ← 0
```

```
7.          if (pattern[i] == pattern[j]) then
8.              lastPrefixPosition ← k
9.      i ← i+1
10.     j ← j+1
11.  table[m - 1 - i] = lastPrefixPosition - i + m - 1;
12.  i ← i-1
       // returns the maximum length of the substring ends at i and is a suffix
13.  for i ← 0 to m
14.  len ← 0;
15.  j ← m-1
16.       while(i>=0)
17.            if(pattern[i]==pattern[j]) then
18.            len ← len+1
19.       i ← i-1
20.       j ← j-1
21.   table[len] ← m-1-i+len
22.   return table
```

BoyerMoore use extra memory space to store the value of the bad character and good suffix length tables of the pattern. It creates a table of size equal to number of alphabets used in the text. Therefore its preprocessing time complexity is $O(m+\Sigma)$. With large alphabets use in text it can shift the pattern with m valid shifts. Therefore its best case run time of matching is O(n/m). But worst case complexity is still O(nm). But in practice it will take best case running time.

4.1.5 Proposed Algorithm

In the proposed work, parallelization of the pattern matching algorithms is done using multithreading concept. Threads are the lightweight processes. Therefore use of these helps to better utilize CPU and Memory. Thread runs concurrently therefore speed of the searching pattern will increase. This approach combines the technique of Brute force and RabinKarp to search for the patterns. First it makes the sets of the patterns and each set contains the patterns that start from the same character. Then threads equal to the number of sets are created. Each thread starts searching the first character of the pattern in the text. After finding the first character it uses RabinKarp approach to match the hash values of the text and the substring. With this one comparison is decrease in eliminating spurious hit step. Algorithm is defined as follow

PARALLELIZED_ALGO (T, P{})

Input: P{} is a set of patterns , T [1…….n] is a text string

Output: Return the position where pattern match start

```
1.  ArrayList patterns ← <HashSet<String>>();
2.  HashSet<String>[] patterns  ← HashSet<String>[256];
3.   for  i ← 0 to  patterns.length
4.            patterns[i].add(strings start with (char)i)
5.   thread ← Thread array[256]
6.  for i ←0 to 256
7.        thread[i] ←new Runnable()
8.           while(patterns[i].hasNext()
9.              String pat ← patterns[i].getElement();
10.             pathash←pat.hashValue
11.              m←pat.length
12.              for j← 0 to n-m
13.                   if( T.substring(j,j+1)==(char)i) do
14.                       Text ← T.substring(j,j+m)
15.                           if( Text.hashValue==pathash) do
16.                                for k ← 1 to m
17.                                    if (P[k]==T[ j+k+1]) do
18.                                      k ← k+1
19.                                          if k==m return j          //match occur
20.               j ← j+1
21.   i←i+1
22.  return -1
```

This above algorithm have preprocessing of creating sets of the patterns according to alphabet character and also take time for generating the hashes of the patterns and the substring start with first character of pattern. Therefore preprocessing run time complexity is O(m+|∑|). In searching phase it takes complexity equal to O(n+m) / 256 and also reduces the time used for calculating the hash value in every step.

4.2 Technologies Used

An IDS environment is created in Netbeans IDE using Java language. Patterns are stored in MySql database that are matched with the text files. Detection engine which is working on the scenario of converting a pcap file into text file and detecting the attack by matching packet stream with the signature defined in the database file using various pattern matching algorithms and compared their run time. Wireshark is used for capturing the packets and parsing of the packets.

4.2.1 Java: Detection engine of IDS is created in the Java language. Java is product by SUN systems. Java provides various in built libraries of classes that make it easy to create projects. Java provides various GUI components that make easy to create a front end in the Java. Netbeans IDE 7.1.1 is used which is java development toolkit is that provides the drag and drop features of swing and awt components. Java has various features:

- Java is platform independent language means it work on the strategy Write Once Run Anytime (WORA).
- Java is pure object oriented language.
- Java is robust because it performs strictly typed and performing run time checks.
- Java provides support for multithreading concept that helps to parallelize the implementation and better utilization of CPU.
- Java converts class code to the byte codes and these byte codes are interpreted by JVM. JVM is installed on every type of operating systems.
- It provides programming to handle run time exceptions.

Used Packages of Java

- Javax.swing: It is package that provides Container and components for creating a GUI like Frames, Windows and components like button, text-field etc.
- Org.jfee.chart: It provides the classes that help to create a bar graph within the project by taking the values of x-axis and y-axis.
- Java.sql: This library of classes is used to connect the java front end with back end MySql. Driver Manager class is used to manage the services of JDBC

drivers. This package also helps to manipulate sql queries. It send the sql queries to the MySql through execute or update query functions

- Org.jnetpcap: jNetPcap library is used to read the pcap files that are captured on the interface. It provides two features either to read pcap files offline or the online mode.
- Java.io: This package helps to read and write the files. User can read and write files through buffered input/output streams.
- Java.lang: This package contains runnable interface which help to create threads. Thread class implement this interface. Threads help to implement parallel processing.

4.2.2 Wireshark

Wireshark is a packet sniffing tool. It captures the packet on the selected Ethernet interface. It receives a copy of inflow or outflow traffic of the system. The captured link layer encapsulates the high layer protocols HTTP, TCP, DNS, UDP or IP. Wireshark has various features like it can also analyze the packet and displays the contents of the selected protocol in the traffic. Suppose a user wants to analyze HTTP header so it parse the traffic and display only http traffic. Wireshark provides the filter to redefine the traffic according to user requirements. Various plug-ins can be used for new protocols. Editcap feature provides the facility to edit the pcap file programmatically.

4.2.3 MySql

MySql is an open source relational database that written in C and C++ languages. It is a fast database that implements client/server architecture. It is a thread-based memory allocation system. MySql is compatible with sql query based language to manipulate the database. It provides various data type like varchar, int, char, binary, date and time. It has features to create join and views.

4.3 Implementation of Pattern Matching Algorithms

In this implementation packets are captured through Wireshark by selecting the Ethernet interface. It filters the traffic according to ip address or header type then save capture traffic as pcap files. Then these files are passed to the detection engine where various pattern matching algorithms are implemented and patterns come from Mysql database.

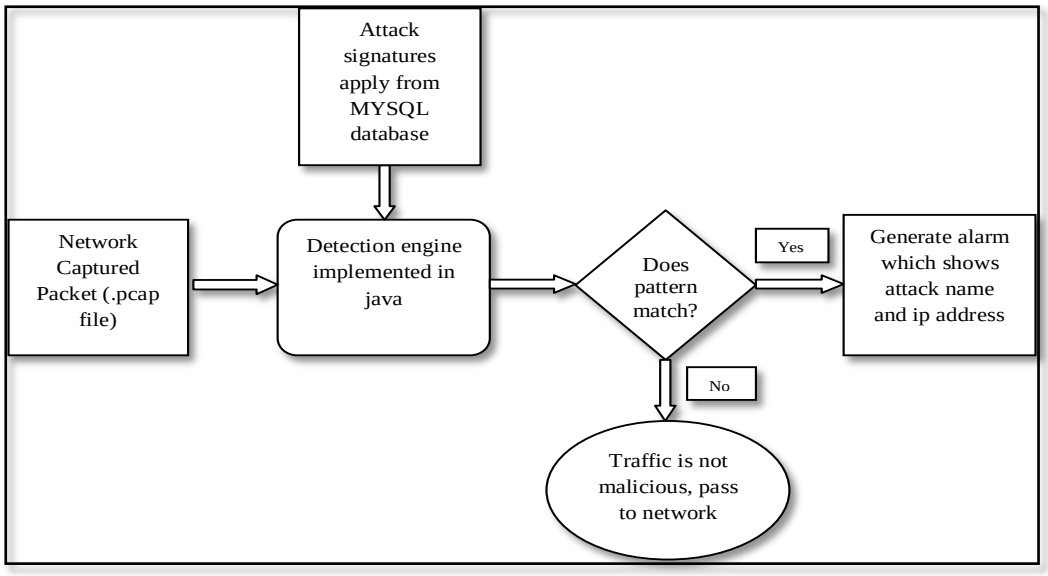


Figure 4.1 Flow diagrams of IDS

If any pattern match with the content of packet then it generate alarms that show source address and attack name and store logs on the database with timestamp. Attack detection interface which is implemented in Java is used to select the file for match the content of packet with the signatures. Various functionalities of system are as follow:

Select attack file:

Select the pcap file from system by clicking on browse button. Browse button will open the directories to select the attack file as shown in figure. Select the file by double clicking on it.

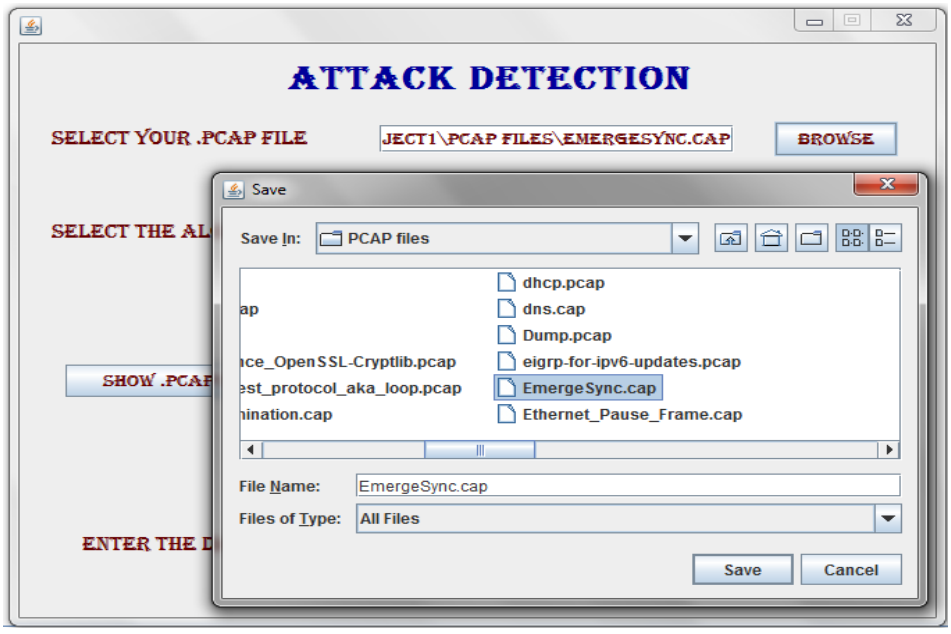


Figure 4.2 Browse the pcap file

Select Algorithm:

After browsing file, user can select a particular algorithm among Brute-force, RabinKarp, KMP, BoyerMoore and Parallelization of algorithms from ComboBox. As shown in Figure 4.3 RabinKarp algorithm is selected for searching the pattern. After clicking on start algorithm, firstly the code convert the given network captured file into text file and then a sql query run on backend which provide the patterns of attacks to the interface. Then all these patterns matched with the given content.

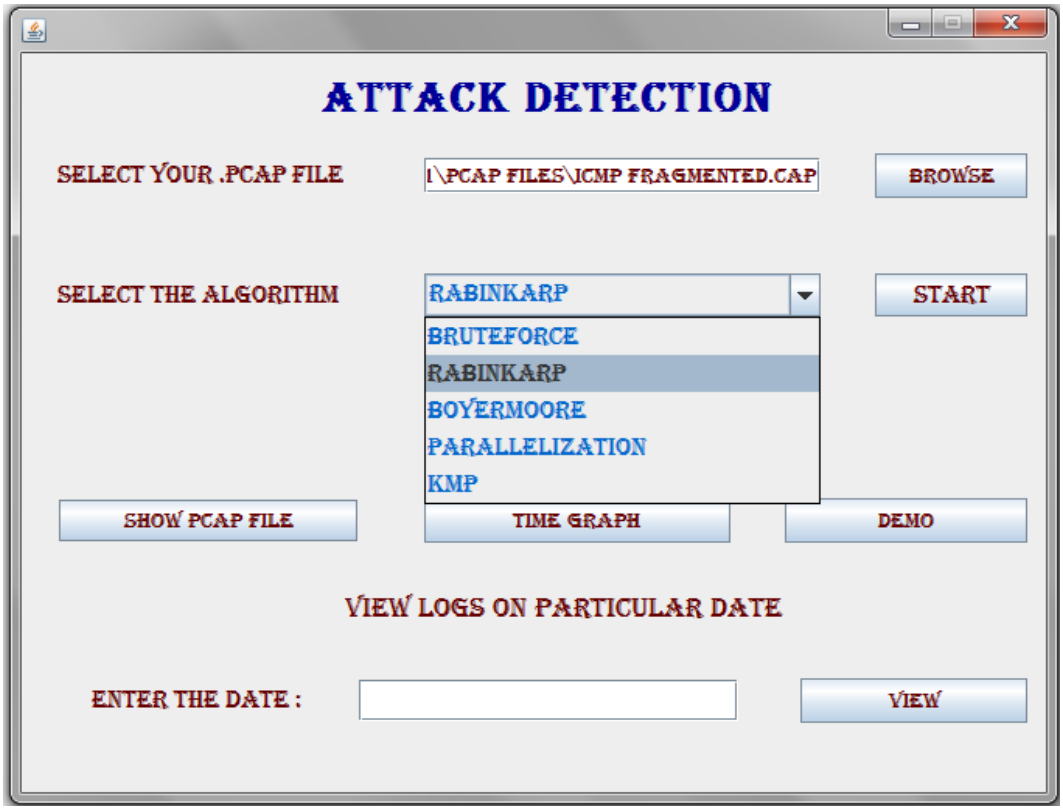


Figure 4.3 Main interface to select algorithm

After the complete search, if pattern is matched with content of file the program calculates three things: source IP address, running time by algorithm in milliseconds and attack name which is shown in Attack Details interface. Figure 4.4 shows the attack details search by RabinKarp algorithm with elapsed time. In similar manner all other algorithms also shows the same details but different elapsed time.

Figure 4.4 Attack details by RabinKarp algorithm

Similar in Figure 4.5 there are attack details by Parallelized algorithm. It clearly shows that parallelized technique decrease the running time of RabinKarp with factor 256 (alphabet size) and also reduce the time used for hash calculation.

Figure 4.5 Attack details by Parallelized algorithm

View packet capture content:

The user who wants to view the content of the network captured traffic can display it by clicking the show pcap file button, the program convert the given .pcap file into text file and show the whole converted file in the new frame as shown in Figure 4.6.

Figure 4.6 Show content of pcap file

View details:

User can view the stored attack details through view logs on particular date panel. By entering the desired date on the text field user can view the details on that particular date. Figure 4.7 shows the logs stored in database on date '2014/06/07'.

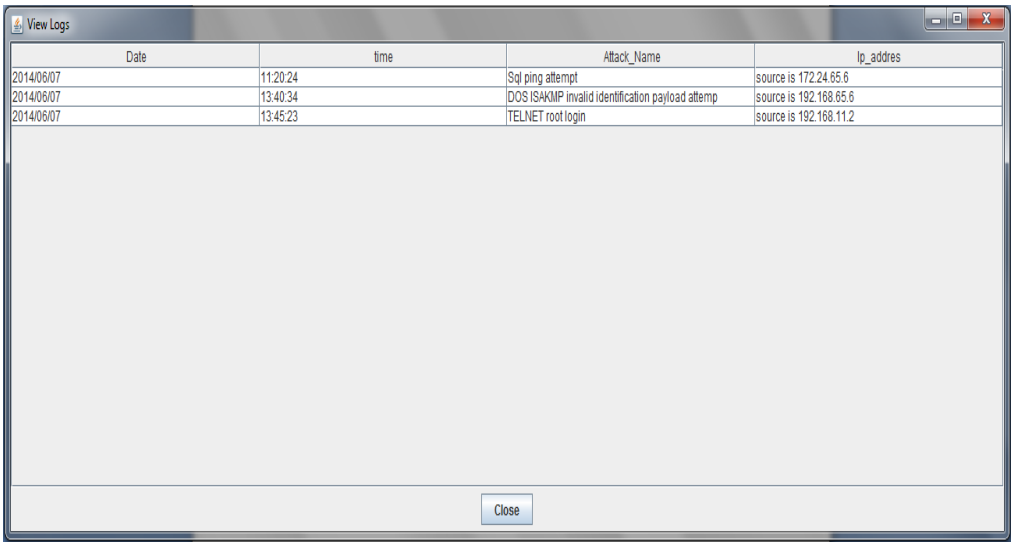


Figure 4.7 Details of attacks on particular date

Comparative analysis of selected existing algorithms:

Time graph button compare the results of all the algorithms. By implementing the progressive bar, running time percentage is easily visualize to users.

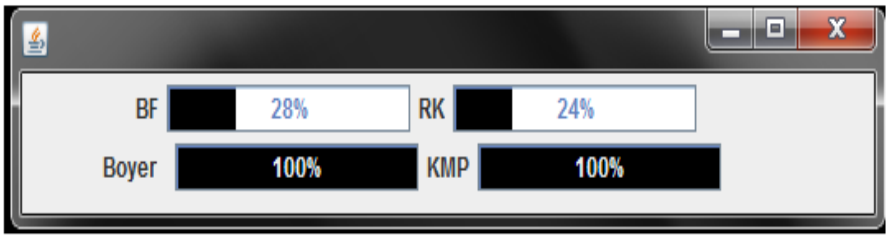


Figure 4.8 Progress bar implementation

After completing the search 100% by all the algorithms, elapsed time is calculated. Based on these elapsed time implement a comparison graph in java as shown in figure 4.8. This graph compares running time of all the algorithms calculated in milliseconds for analyzing. After analyzing the graph it has been observed that Boyer Moore is faster than all other algorithms.

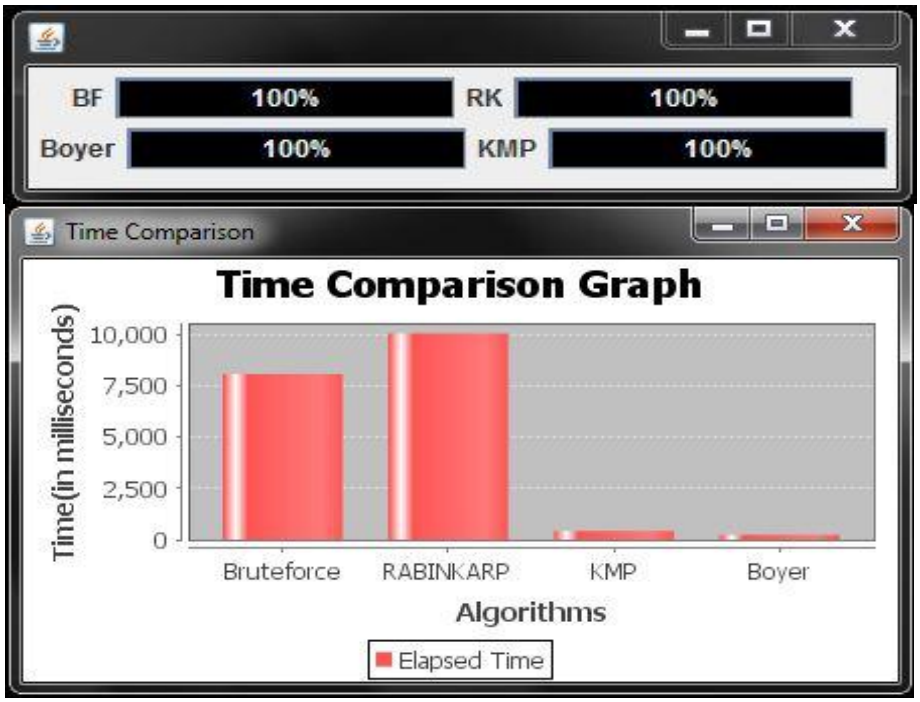


Figure 4.9 Time Comparison Graph of elapsed time

Above is the analysis on the network captured file. Demo button open the new frame which implement all the pattern matching algorithms using multithreading and demonstrates the working of algorithms.

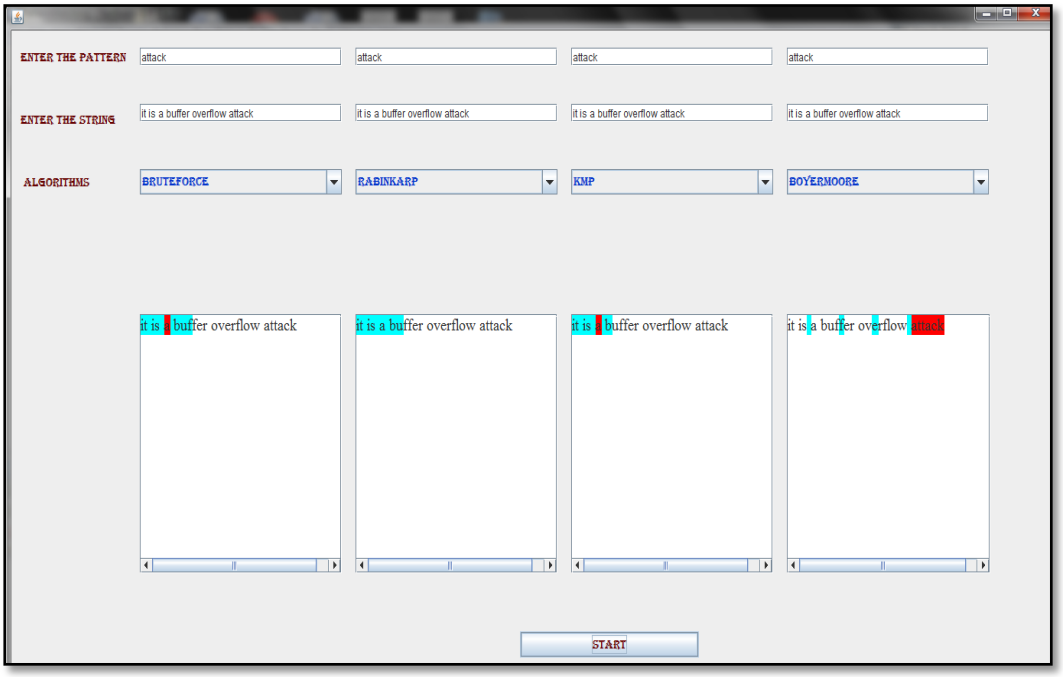


Figure 4.10 demonstrates the working of algorithms

Figure 4.10 clearly shows that BoyerMoore algorithms complete its search before all other algorithms Cyan highlighter shows the mismatch and red highlighter highlight the match characters and the pattern. After completing the search user can view all the algorithms matching positions as shown in Figure 4.11.

Figure 4.11 demonstration of algorithms after searching the pattern

Chapter 5

Experimental Results

Running time of Brute force, RabinKarp, KMP, BoyerMoore and proposed parallelization algorithms are recorded by varying the size of attack file and by varying the number of patterns in a table.

Size of the attack file	No. of patterns	Time in sec				
		Brute-force	Rabin-Karp	KMP	Boyer-Moore	Parallelization
408,036 bytes	40	120	180	10	4	1
	100	296	360	19	11	2
	200	422	586	55	39	3
	400	689	720	108	92	4
	800	879	1003	252	186	5
	1000	1020	1245	339	223	6
1,609,728 bytes	40	240	390	19	9	2
	100	420	570	29	16	3
	200	596	774	67	52	4
	400	758	920	125	102	5
	800	960	1180	227	189	6
	1000	1210	1420	420	257	7
6,828,032 bytes	40	65	89	7	3	0.4
	100	158	186	10	6	1
	200	280	367	18	12	1.6
	400	389	508	39	26	2
	800	620	724	72	49	3
	1000	750	967	98	71	5

19,269,599 bytes	40	789	1030	73	48	5
	100	1290	1486	99	81	7
	200	1430	1703	127	102	8
	400	1890	2156	197	145	10
	800	2278	2589	345	232	11
	1000	2456	2834	410	308	13

Table 5.1 Run time of algorithms against variation in file size and number of patterns

A bar graph is created which compare all the algorithms on the basis of their running time vs. Different size pcap files. Figure 5.1 shows that BoyerMoore algorithm is faster than all other existing algorithms. But parallelization of combined approach of brute force and RabinKarp is faster than BoyerMoore and it decreases the run time of RabinKarp with factor equal to alphabet size.

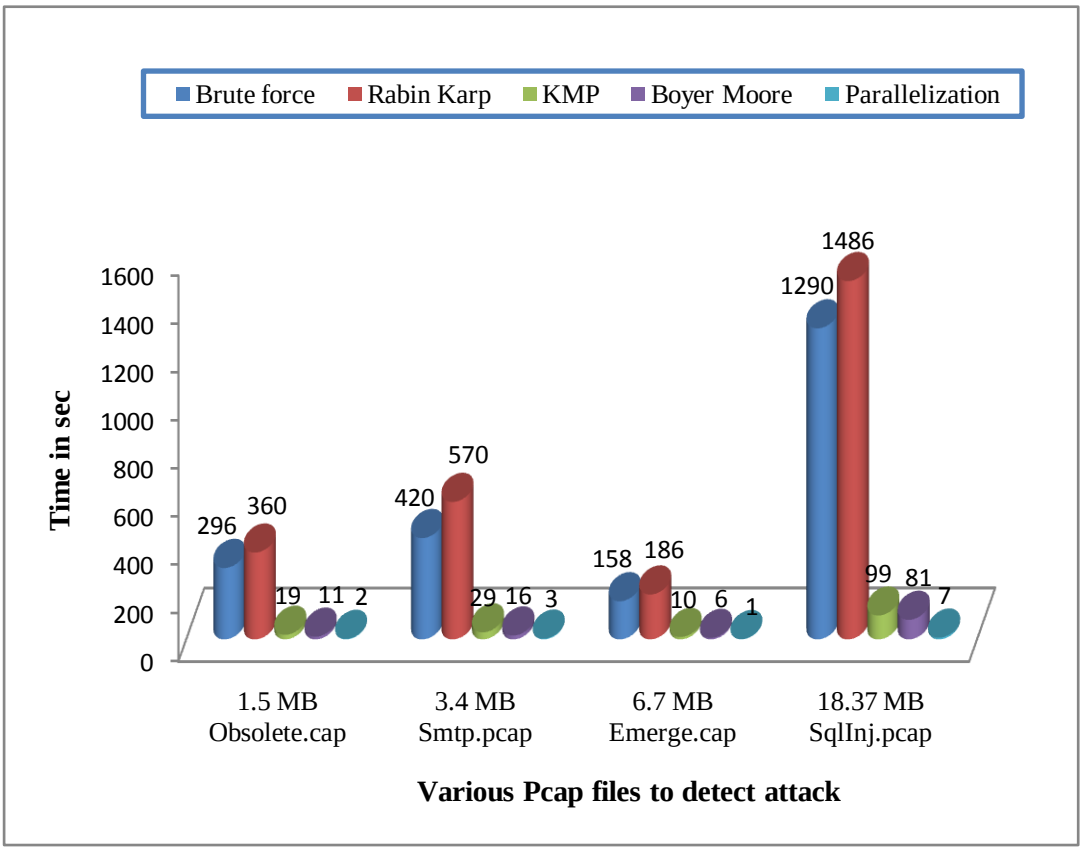


Figure 5.1 Time comparison graph of pattern matching algorithms against different files of different size

A line graph is created by varying the number of patterns of particular network captured traffic file against running time of all the algorithms taken for searching the number of patterns.

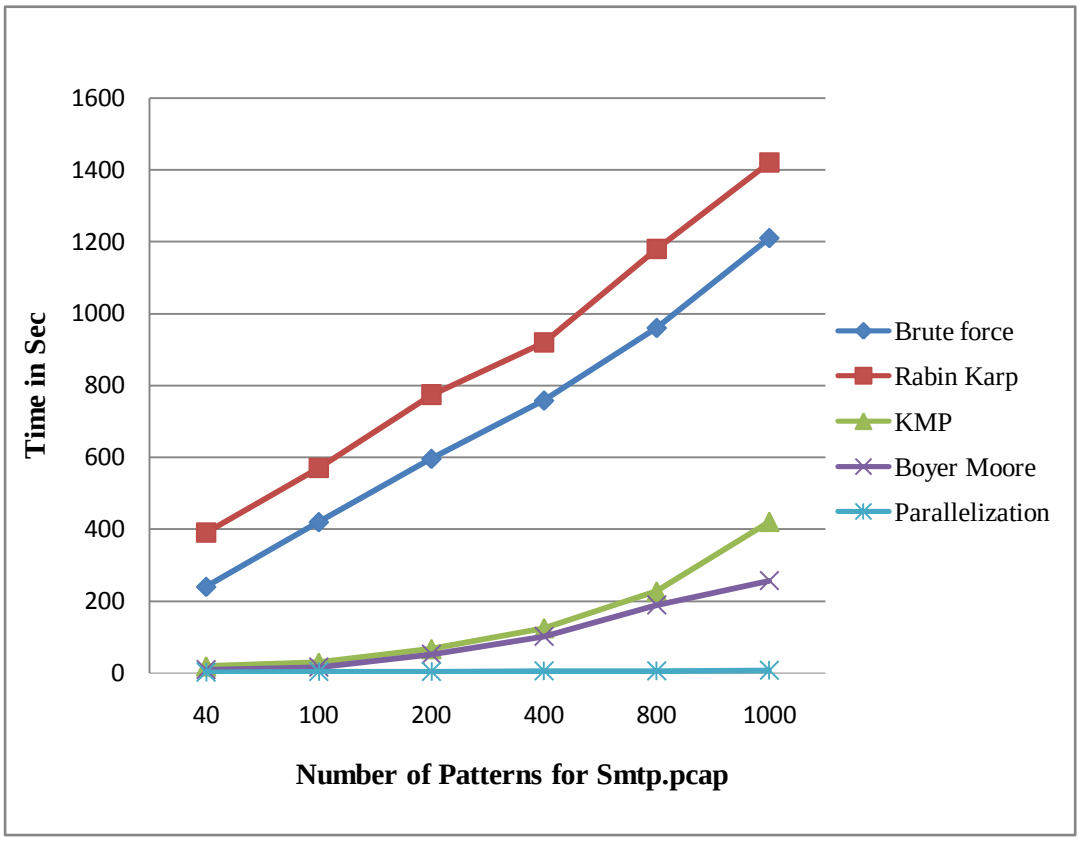


Figure 5.2 Time comparison graph of pattern matching algorithms against number of patterns

Figure 5.2 shows that RabinKarp algorithm performs worst in case of pattern matching for intrusion detection systems. BoyerMoore is shows best performance among existing algorithms. But the running time of parallelization technique is better than all selected existing algorithms. Graph shows that run time of parallelization is decreasing as number of patterns are increasing.

Following table 5.2 is created that compares the working of algorithms according to their searching techniques, searching ideas and approach used. It also compares their preprocessing and searching phase time complexities. This table summarizes all the pattern matching algorithms used in this thesis.

Parameters	**Algorithms**				
	Brute-Force	***Rabin-Karp***	***KMP***	***Boyer-Moore***	***Parallelized***
Pre-processing	No	O (km)	O(km)	O(k(m+|∑|))	O(km+|∑|)
Search Type	Left to Right	Left to Right	Left to Right	Right to Left	Left to Right
Running Time	O(k(nm))	O (k(n+m))	O(k(n+m))	O(k(n/m))	$\frac{O(k(n+m))}{\|\sum\|}$
Search Ideas	Search by matching all characters	Compares hash values of the text and the pattern	Constructs an automaton from the pattern	Bad-character and good-suffix heuristics to find valid shift	Compares the first character and hash values of the text and pattern
Approach	linear search	hashing based	heuristics based	heuristics based	linear and hashing based

Table 5.2 Comparison of complexities of algorithms

In the table, k is the constant that represent number of patterns to be searched. m is the size of pattern and n is the size of file. Big O is a notation that describes the upper bound of the algorithms. Table 5.2 shows that algorithms use three approaches linear search, hashing based and heuristics based.

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

Intrusion detection and prevention systems are security measures whose performance depends on fast pattern matching. From this work it is concluded that RabinKarp approach is efficient but due to mathematical calculations that are involve in calculating hash values performs worst. Therefore running time of RabinKarp is more than Brute-force search. Knuth-Morris-Pratt uses prefix heuristics to shift the pattern but in worst case when there is no prefix it performs like Brute-force search technique. Therefore it is not good algorithm for intrusion detection systems. Boyer-Moore uses prefix heuristics with the bad character shift. In case when there is no prefix in pattern then it uses the bad character shift. But in case if pattern length is small then Boyer-Moore performs less efficiently. Therefore new algorithm is proposed and implemented. Proposed parallelization of Combined approach of RabinKarp and Brute-force performs best among all the pattern matching algorithms. From comparison table it is proved that parallelization is 256 times better than RabinKarp algorithm. It means parallelization reduce the running time with factor $|\sum|$ which is number of alphabets exist. This algorithm takes more memory for implementing the parallel processes and time is spent on creating thread is more. Rather than spending time on preprocessing it takes very less time for searching. Pattern searching parallelization algorithm is selected to implement for the intrusion detection and prevention systems.

6.2 Future Scope

The project implementation compares the various pattern matching algorithms and prove that practically proposed pattern searching parallelization performs best. Till now all the algorithms used in IDPS are only single threaded but in future, pattern matching parallelization method can be implemented in Snort NIDPS using GPU (Graphics Processing Unit) programming so that it can detect attack before it occur to the system.

References

[1] S. Patil, P. Rane, Dr. B. B. Meshram, "IDS vs. IPS." Proceedings of International Journal of Computer Networks and Wireless Communications, Vol. 2, 2012.

[2] K. Scarfone, P. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)" NIST Publication, 2007.

[3] R. Rehman, "Intrusion detection with SNORT: advanced IDS techniques using SNORT, APACHE, MYSQL, PHP, and ACID", Pearson Education Inc., Prentice Hall 75-129, 2003.

[4] Zero Day Protection [Online]. Available: https://www.watchguard.com/products/zeroday.asp.

[5] Data Leakage - Threats and Mitigation [Online]. Available: http://www.sans.org/reading-room/whitepapers/awareness/data-leakage--threats-mitigation-1931.

[6] J. Han, M. Beheshti, K. Kowalski and J. Tomelden. "Component-based Software architecture design for Network Intrusion detection and prevention system", IEEE 6th international conference on Information Technology, pp. 248-253, 2009.

[7] Malicious Threats and Vulnerabilities in Instant Messaging [Online]. Available: http://www.symantec.com/avcenter/reference/malicious.threat.instant.messaging.pdf.

[8] N. Ye, S. Masum, E. Qiang and S. Vilbert. "Multivariate statistical analysis of Audit trails for Host-Based Intrusion Detection", IEEE Transactions on Computers, Vol. 24, No. 10, 2006.

[9] T.H. Corman, C.E. Leiserson, R.L. Rivest and C. Stein, "Introduction to Algorithms", The MIT Press, 2009. Cambridge, Massachusetts London, England.

[10] Snort [Online]. Available: http://snort.org.

[11] H. Changwei, J. Xiong, and Z. Peng. "Applied research on Snort intrusion detection model in the campus network." Proceedings of IEEE Symposium on In Robotics and Applications (ISRA), pp. 596-599, 2012.

[12] M. R. Karp and M. O. Rabin. "Efficient randomized pattern-matching algorithms." Proceedings of IBM Journal of Research and Development, Vol.

31(2), pp. 249-260, 1987.

[13] D. E. Knuth, J. H. Morris, and V. R. Pratt. "Fast pattern matching in strings." Proceedings of SIAM journal on computing, Vol. 6(2), pp. 323-350, 1977.

[14] R. S. Boyer and J. S. Moore. “A fast string searching algorithm.” Proceedings of Communications of the ACM, Vol. 20, pp. 762–772, October 1977.

[15] R. A. Baeza-Yatesand and G. H. Gonnet. "A new approach to text searching." Proceedings of Communications of the ACM Vol. 35(10), pp. 74-82, 1992.

[16] L. Colussi, "Correctness and efficiency of pattern matching algorithms." Information and Computation Vol. 95 No.2, pp. 225-251, Elsevier 1991.

[17] L. Colussi, G. Zvi, and G. Raffaele. "On the exact complexity of string matching." Proccedings of 31st IEEE Annual Symposium, pp. 135-144, 1990.

[18] Aho, Alfred V., and Margaret J. Corasick. "Efficient string matching: an aid to bibliographic search." Communications of the ACM 18.6 (1975): 333-340.

[19] B. Commentz-Walter, Beate. “A string matching algorithm fast on the average”. Proceedings of the 6th Colloquium, on Automata, Languages and Programming, Springer-Verlag, pp. 118-132, 1979.

[20] S. Wu, U. Manber, and Eugene Myers. "A Fast Algorithm For Multi-Pattern Searching". Technical Report TR 94-17, University of Arizona at Tuscon, 1994.

[21] E. Mohamed and A. Al-Tobi, "Refinements of Longest Common Subsequence Algorithm." Proceedings of IEEE International Conference on Computer Systems and Applications (AICCSA), pp. 1-5,2010.

[22] J. Rubi, R. Devika, and L. Arockiam. "Positional_LCS: A position based algorithm to find Longest Common Subsequence (LCS) in Sequence Database (SDB)." Proceedings of IEEE Conference on Computational Intelligence & Computing Research (ICCIC), pp. 1-4, Coimbatore, 2012.

[23] Q. Zheng. "An improved multiple patterns matching algorithm for intrusion detection.” Proceedings of IEEE Conference on Intelligent Computing and Intelligent Systems (ICIS), Vol. 2, pp. 124-127, Xiamen, 2010.

[24] J. Aldwairi, M. Monther, and D. Alansari. "Exscind: Fast pattern matching for intrusion detection using exclusion and inclusion filters." Proceedings of 7th IEEE International Conference on Next Generation Web Services Practices (NWeSP),2011.

[25] P. Brodanac, L. Budin, and D. Jakobovi, “Parallelized Rabin-Karp Method for

Exact String Matching" Proceedings of 33rd IEEE International Conference on Information Technology Interfaces, pp. 27-30, 2011.

[26] H. Le and V. K. Prasanna. "A memory-efficient and modular approach for large-scale string pattern matching." Proceedings of IEEE Conference on Computers, Vol. 62(5), pp. 844-857, 2013.

[27] C. Jason, S. Staniford, and J. McAlerney, Coit. "Towards faster string matching for intrusion detection or exceeding the speed of snort." Proceedings of IEEE International Conference on DARPA Information Survivability, Vol. 1, pp. 367-373, 2001.

[28] Y.Jingbo, J. Zheng, and S. Ding. "An Improved Pattern Matching Algorithm." Proceedings of 3rd IEEE International Conference on Intelligent Information Technology and Security Informatics (IITSI), pp.599-603, 2010.

[29] B. Jakub, P. Buciak, and P. Sapiecha. "Building dependable intrusion prevention systems." Proceedings of IEEE International Conference on Dependability of Computer Systems (DepCos), pp. 135-142, 2006.

[30] B. Fuyao, G. Zhao, and L. Qingwei. "A string matching algorithm based on efficient hash function." Proceedings of IEEE Conference on Information Engineering and Computer Science, pp. 1-4, 2009.

List of Publication

Vibha Gupta, Maninder Singh and Vinod K. Bhalla, " Pattern Matching Algorithms for Intrusion detection and prevention systems" 3rd International Conference on Advances in Computing, Communications and informatics, IEEE Delhi, India September 2014. (Communicated)