



# INTRUCTION DETECTION SYSTEM



A PROJECT REPORT

P- 2162

*Submitted by*

<b>NAVEENKUMAR.S</b>	<b>71204205023</b>
<b>RAJAKUMAR.S</b>	<b>71204205032</b>
<b>DEEPAKPRASANTH.C</b>	<b>71204205301</b>

*In partial fulfillment for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY**

*in*

**INFORMATION TECHNOLOGY**

**KUMARAGURU COLLEGE OF TECHNOLOGY,  
COIMBATORE**

**ANNA UNIVERSITY: CHENNAI 600 025**

**APRIL 2008**



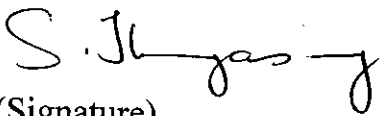
ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2008

ANNA UNIVERSITY: CHENNAI 600 025

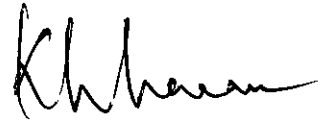
**BONAFIDE CERTIFICATE**

Certified that this project report "Intruction Detection System" is the bonafide work of "S.Naveenkumar(71204205023), S.Rajakumar (71204205032) and C.Deepakprasanth (71204205301)" who carried out the project work under my supervision.



(Signature)

Dr. S.Thangasamy



(Signature)

Mr. K.R Baskaran

**HEAD OF THE DEPARTMENT**

Dept of Information Technology,

Kumaraguru College of Technology,

Coimbatore- 641 006.

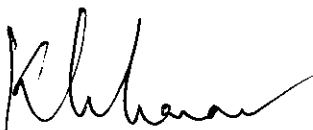
**SUPERVISOR**

Dept of Information Technology,

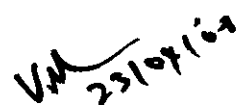
Kumaraguru College of Technology,

Coimbatore - 641 006.

Submitted for viva-voice examination held on..23.04.08..



INTERNAL EXAMINER



EXTERNAL EXAMINER

**ACKNOWLEDGEMENT**

---

## ACKNOWLEDGEMENT

We express our sincere thanks to our chairman **Padmabhushan Arutselvar Dr. N. Mahalingam B.Sc, F.I.E** and vice-chairman **Prof. K. Arumugam B.E., M.S., M.I.E.**, for all their support and ray of strengthening hope extended. We are immensely grateful to our principal **Dr. Joseph V. Thanikal M.E., Ph.D., PDF., CEPIT.**, for his invaluable support to the outcome of this project.

We thank to our project guide **Prof.K.R.Baskaran**, Asst Professor, Department of Information Technology for his support given to us through this project.

We thank the teaching and non-teaching staffs of our department for providing us the technical support for the duration of our project.

We express our humble gratitude and thanks to our beloved parents and family members who have supported and helped us to complete the project and our friends, for lending us valuable tips, support and co-operation throughout the project work.

**DECLARATION**

---

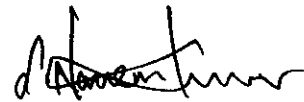
## DECLARATION

We hereby declare that the project entitled “**Intruction Detection System**” is done by us and to the best of our knowledge a similar work has not been submitted to the **Anna University** or any other Institution, for fulfillment of the requirement of the course study.

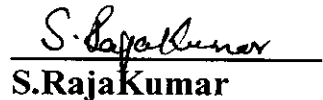
This report is submitted on the partial fulfillment of the requirement for all awards of the **Degree of Bachelor of Information Technology of Anna University, Chennai.**

Place: Coimbatore

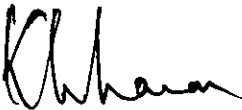
Date: 23.04.08



S.NaveenKumar



S.RajaKumar



Mr. K.R Baskaran



C.DeepakPrasanth

*ABSTRACT*

---

# ABSTRACT

Intrusion detection system is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity, availability or to bypass the security mechanisms of a computer or network.

Intrusion detection systems are also classified based on the types of systems they monitor. The two main systems monitored for intrusions are host-based systems and network based systems.

Host-based intrusion detection attempts to detect against attacks on a particular machine. This is typically done through analysis of a computers log files.

Host based IDS typically monitor system, event, and security logs on Windows NT and syslog in Unix environments. When any of these files change, the IDS compare the new log entry with attack signatures to see if there is a match. If so, the system responds with administrator alerts and other calls to action.

Host-based IDS have grown to include other technologies. One popular method for detecting intrusions checks key system files and executables via checksums at regular intervals for unexpected changes. The timeliness of the response is in direct relation to the frequency of the polling interval. Finally, some products listen to port activity and alert administrators when specific ports are accessed. This type of detection brings an elementary level of network-based intrusion detection into the host-based environment.



Host-based IDS monitor user and file access activity, including file accesses, changes to file permissions, attempts to install new executables and/or attempts to access privileged services.

The project modules include Password attack, Scanning attack, Sniffing attack and Spoofing attack. The project is developed using C and Shell scripting in Linux environment.

## **CONTENTS**

---

# TABLE OF CONTENTS

CHAP.NO.	TITLE	PAGE NO.
	<b>ACKNOWLEDGEMENT</b>	<b>iii</b>
	<b>ABSTRACT</b>	<b>v</b>
	<b>LIST OF FIGURES</b>	<b>ix</b>
<b>1.</b>	<b>INTRODUCTION</b>	
	1.1 GENERAL	1
	1.2 PROBLEM DEFINITION	9
<b>2.</b>	<b>LITERATURE REVIEW</b>	
	2.1 FEASIBILITY STUDY	
	2.1.1 EXISTING SYSTEM	10
	2.1.2 PROPOSED SYSTEM	11
	2.2 HARDWARE REQUIREMENTS	12
	2.3 SOFTWARE REQUIREMENTS	12
	2.4 SOFTWARE OVERVIEW	13
<b>3.</b>	<b>DETAILS OF THE METHODOLOGY EMPLOYED</b>	
	3.1 HOST BASED INTRUCTION DECTION	19
	3.2 ABOUT THE MODULE	
	3.2.1 SCANNING ATTACK	22
	3.2.2 PASSWORD ATTACK	23
	3.2.3 SNIFFING ATTACK	24
	3.2.4 SPOOFING ATTACK	25

<b>4.</b>	<b>SYSTEM DEVELOPMENT</b>	<b>27</b>
	4.1 STAGES OF DEVELOPMENT OF SYSTEM	28
	4.2 TESTING AND IMPLEMENTATION	
	4.2.1 SYSTEM TESTING	30
	4.2.2 TESTING STRATIGES	30
	4.2.3 SYSTEM IMPLEMENTATION	32
<b>5.</b>	<b>CONCLUSION</b>	<b>35</b>
<b>6.</b>	<b>FUTURE ENHANCEMENTS</b>	<b>36</b>
<b>7.</b>	<b>REFERENCES</b>	<b>37</b>
	<b>APPENDIX 1 - SOURCE CODE</b>	<b>38</b>
	<b>APPENDIX 2 – SCREEN – SHOTS</b>	<b>68</b>

## *LIST OF FIGURES*

---

## **LIST OF FIGURES**

<b>FIG NO.</b>	<b>FIGURE NAME</b>	<b>PAGE NO.</b>
3.1	SCANNING ATTACK	22
3.2	PASSWORD ATTACK	23
3.3	SNIFFING ATTACK	24
3.4	SPOOFING ATTACK	25
3.5	FEATURES OF HOST BASED IDS	26
4.1	SOFTWARE DEVELOPMENT	27

## ***INTRODUCTION***

---

# 1. INTRODUCTION

## 1.1 GENERAL:

An **Intrusion Detection System** (or **IDS**) generally detects unwanted manipulations to systems. There are a lot of different types of IDS, some of them are described here. The manipulations may take the form of attacks by skilled malicious hackers, or Script kiddies using automated tools.

An IDS is required to detect all types of malicious network traffic and computer usage that can't be detected by a conventional firewall. This includes network attacks against vulnerable services, data driven attacks on applications, host based attacks such as privilege escalation, unauthorized logins and access to sensitive files, and malware (viruses, trojan horses, and worms).

An IDS is composed of several components: **Sensors** which generate security events, a **Console** to monitor events and alerts and control the sensors, and a central **Engine** that records events logged by the sensors in a database and uses a system of rules to generate alerts from security events received. There are several ways to categorise an IDS depending on the type and location of the sensors and the methodology used by the engine to generate alerts. In many simple IDS implementations all three components are combined in a single device or appliance.

An *Intrusion Detection System* (IDS) is designed to monitor all inbound and outbound network activity and identify any suspicious patterns that may indicate a network or system attack from someone attempting to break into or compromise a system. IDS is considered to be a *passive-monitoring* system, since the main function of an IDS product is to warn you of suspicious activity taking place – not prevent them. An IDS essentially reviews your network traffic and data and will identify probes, attacks, exploits and other vulnerabilities. IDSs can respond to the suspicious event in one of



several ways, which includes displaying an alert, logging the event or even paging an administrator. In some cases the IDS may be prompted to reconfigure the network to reduce the effects of the suspicious intrusion.

An IDS specifically looks for suspicious activity and events that might be the result of a virus, worm or hacker. This is done by looking for known intrusion signatures or attack signatures that characterize different worms or viruses and by tracking general variances which differ from regular system activity. The IDS is able to provide notification of only *known* attacks.

The term IDS actually covers a large variety of products, for which all produce the end result of detecting intrusions. An IDS solution can come in the form of cheaper shareware or freely distributed open source programs, to a much more expensive and secure vendor software solution. Additionally, some IDSs consist of both software applications and hardware appliances and sensor devices which are installed at different points along your network.

### **Network-based vs. Host-based Systems:**

Intrusion detection systems are network or host based solutions. Network-based IDS systems (NIDS) are often standalone hardware appliances that include network intrusion detection capabilities. It will usually consist of hardware sensors located at various points along the network or software that is installed to system computers connected to your network, which analyzes data packets entering and leaving the network. Host-based IDS systems (HIDS) do not offer true real-time detection, but if configured correctly are close to true real-time.

Host-based IDS systems consist of software agents installed on individual computers within the system. HIDS analyze the traffic to and from the specific computer on which the intrusion detection software is installed on. HIDS systems often provide features you can't get with network-based IDS. For example, HIDS are able to monitor activities that only an administrator should be able to implement. It is also able to monitor changes to key system files and any attempt to overwrite these files. Attempts to install

Trojans or backdoors can also be monitored by a HIDS and stopped. These specific intrusion events are not always seen by a NIDS.

While it depends on the size of your network and the number of individual computers which require intrusion detection system, NIDS are usually a cheaper solution to implement and it requires less administration and training – but it is not as versatile as a HID. Both systems will require Internet access (bandwidth) to ensure they system is kept up-to-date with the latest virus and worm signatures.

In a network-based system, or NIDS, the sensors are located at choke points in the network to be monitored, often in the DMZ or at network borders. The sensor captures all network traffic flows and analyzes the content of individual packets for malicious traffic. In a host-based system, the sensor usually consists of a software agent which monitors all activity of the host on which it is installed. Hybrids of these two types of system also exist.

- A Network Intrusion Detection System is an independent platform which identifies intrusions by examining network traffic and monitors multiple hosts. Network Intrusion Detection Systems gain access to network traffic by connecting to a hub, network switch configured for port mirroring, or network tap. An example of a NIDS is Snort.
- A Host-based Intrusion Detection System consists of an agent on a host which identifies intrusions by analyzing system calls, application logs, file-system modifications (binaries, password files, capability/acl databases) and other host activities and state.
- A Hybrid Intrusion Detection System combines both approaches. Host agent data is combined with network information to form a comprehensive view of the network. An example of a Hybrid IDS is Prelude.

### Network intrusion detection system:

A **network intrusion detection system (NIDS)** is a system that tries to detect malicious activity such as denial of service attacks, port-scans or even attempts to crack into computers by monitoring network traffic.

The NIDS does this by reading all the incoming packets and trying to find suspicious patterns. If, for example, a large number of TCP connection requests to a very large number of different ports is observed, one could assume that there is someone committing a "portscan" at some of the computer(s) in the network. It also (mostly) tries to detect incoming shellcodes in the same manner that an ordinary intrusion detection systems does.

A NIDS is not limited to inspecting incoming network traffic only. Oftentimes valuable information about an ongoing intrusion can be learned from outgoing or local traffic as well. Some attacks might even be staged from the inside of the monitored network or network segment, and are therefore not regarded as incoming traffic at all.

Often, network intrusion detection systems work with other systems as well. They can for example update some firewalls' blacklist with the IP addresses of computers used by (suspected) crackers.

**Host-based intrusion-detection** is the art of detecting malicious activity *within* a single computer.

A host-based intrusion detection system (HIDS) uses host log information, system activity, and scanners such as virus scanners to determine whether a computer host is being used for illegitimate purposes. HIDS may be local to the protected host, remote (via syslogd, etc), or part of a distributed intrusion detection system.

A common technique is to make checksums of important system files that should not be altered under normal circumstances. Intruders are likely to replace system

components with so-called *root kits* that enable them to remain hidden in the system while performing further probing such as sniffing

### **Overview:**

A HIDS will monitor all or part of the dynamic behavior and of the state of a computer system. Much as a NIDS will dynamically inspect network packets, a HIDS might detect which program accesses what resources and assure that (say) a word-processor hasn't suddenly and inexplicably started modifying the system password-database. Similarly a HIDS might look at the state of a system, its stored information, whether in RAM, in the file-system, or elsewhere; and check that the contents of these appear as expected.

One can think of a HIDS as an agent that monitors whether anything/anyone - internal or external - has circumvented the security policy that the operating system tries to enforce.

### **Monitoring dynamic behavior:**

Many computer users have encountered tools that monitor dynamic system behavior in the form of anti-virus (AV) packages. While AV programs often also monitor system state, they do spend a lot of their time looking at who is doing what inside a computer - and whether a given program should or should not access one or another system resource. The lines become very blurred here, as many of the tools overlap in functionality.

### **Monitoring state:**

The principle of operation of a HIDS depends on the fact that successful intruders (crackers) will generally leave a trace of their activities. (In fact, such intruders often want to *own* the computer they have attacked, and will establish their "ownership" by installing software that will grant the intruders future access to carry out whatever

activity ( keyboard logging, identity theft, spamming, botnet activity, spyware-usage etc.) they envisage.)

In theory, a computer user has the ability to detect any such modifications, and the HIDS attempts to do just that and reports its findings.

Ideally a HIDS works in conjunction with a NIDS, such that a HIDS finds anything that slips past the NIDS.

Ironically, most successful intruders, on entering a target machine, immediately apply best-practice security techniques to secure the system which they have infiltrated, leaving only their own backdoor open, so that other intruders can not take over *their* computers. (Crackers are a competitive bunch...) Again, one can detect (and learn from) such changes.

### **Technique:**

In general a HIDS uses a database (object-database) of system objects it should monitor - usually (but not necessarily) file-system objects. A HIDS could also check that appropriate regions of memory have not been modified, for example - the system-call table comes to mind for Linux, and various vtable structures in Microsoft Windows.

For each object in question a HIDS will usually remember its attributes (permissions, size, modifications dates) and perhaps create a checksum of some kind (an MD5 hash or similar) for the contents, if any. This information gets stored in a database for later comparison (checksum-database). Note that a matching MD5 hash does not provide a complete guarantee that an intruder or other unauthorised user has not tampered with the target file. Recent ( 2004) research has resulted in claims (still under debate) that the probability of such tampering may exceed what one might hope.

### **Operation:**

At installation time - and whenever any of the monitored objects change legitimately - a HIDS must initialise its checksum-database by scanning the relevant

objects. Persons in charge of computer security need to control this process tightly in order to prevent intruders making un-authorized changes to the database(s). Such initialisation thus generally takes a long time and involves cryptographically locking each monitored object and the checksum databases or worse. Because of this, manufacturers of HIDS usually construct the object-database in such a way that makes frequent updates to the checksum database unnecessary.

Computer systems generally have many dynamic (frequently changing) objects which intruders want to modify - and which a HIDS thus should monitor - but their dynamic nature makes them unsuitable for the checksum technique. To overcome this problem, HIDS employ various other detection techniques: monitoring changing file-attributes, log-files that decreased in size since last checked, and a raft of other means to detect unusual events.

Once a system administrator has constructed a suitable object-database - ideally with help and advice from the HIDS installation tools - and initialized the checksum-database, the HIDS has all it requires to scan the monitored objects regularly and to report on anything that may appear to have gone wrong. Reports can take the form of logs, e-mails or similar.

### **Protecting the HIDS:**

A HIDS will usually go to great lengths to prevent the object-database, checksum-database and its reports from any form of tampering. After all, if intruders succeed in modifying any of the objects the HIDS monitors, nothing can stop such intruders from modifying the HIDS itself - unless security administrators take appropriate precautions. Many worms and viruses will try to disable anti-virus tools, for example. Sadly, a lot of them succeed in doing so.

Apart from crypto-techniques, HIDS might allow administrators to store the databases on a CD-ROM or on other read-only memory devices (another factor militating for infrequent updates...) or storing them in some off-system memory. Similarly, a HIDS will often send its logs off-system immediately - in some instances via one-way

communications channels, such as a serial port which only has "Transmit" connected, for example.

One could argue that the trusted platform module comprises a type of HIDS. Although its scope differs in many ways from that of a HIDS, fundamentally it provides a means to identify whether anything/anyone has tampered with a portion of a computer. Architecturally this provides the ultimate (at least at this point in time) host-based intrusion detection, as depends on hardware external to the CPU itself, thus making it that much harder for an intruder to corrupt its object and checksum databases.

### **Intrusion-prevention system:**

An **intrusion prevention system** is any device which exercises access control to protect computers from exploitation. "Intrusion prevention" technology is considered by some to be an extension of intrusion detection (IDS) technology, but it is actually another form of access control, like an application layer firewall.

Intrusion prevention systems were invented independently by Jed Haile and Vern Paxson to resolve ambiguities in passive network monitoring by placing detection systems in-line. A considerable improvement upon firewall technologies, IPS make access control decisions based on application content, rather than IP address or ports as traditional firewalls had done. As IPS systems were originally a literal extension of intrusion detection systems, they continue to be related.

Intrusion prevention systems may also act at the host level to deny potentially malicious activity. There are advantages and disadvantages to host-based IPS compared with network-based IPS. In many cases, the technologies are thought to be complementary.

An Intrusion Prevention system must also be a very good Intrusion Detection system to enable a low rate of false positives. Some IPS systems can also prevent yet to be discovered attacks, such as those caused by a Buffer overflow.

## 1.2. PROBLEM DEFINITION:

When a user of an information system takes an action that the user was not legally allowed to take, it is called *intrusion*. The intruder may come from outside, or the intruder may be an insider who exceeds his limited authority to take action. Whether or not the action is detrimental, it is of concern because it might be detrimental to the health of the system or to the service provided by the system.

A system that tries to identify attempts to hack or break into a computer system or to misuse it. IDSs may monitor packets passing over the network, monitor system files, monitor log files, or set up deception systems that attempt to trap hackers.

Intrusion *detection* involves determining that some entity, an *intruder*, has attempted to gain, or worse, has gained unauthorized access to the system. Casual observation shows that none of the automated detection approaches seek to identify an intruder *before* that intruder initiates interaction with the system. Of course, system administrators routinely take actions to *prevent* intrusion. These can include requiring passwords to be submitted before a user can gain any access to the system, fixing known vulnerabilities that an intruder might try to exploit in order to gain unauthorized access, blocking some or all network access, as well as restriction of physical access. Intrusion detection systems are used in addition to such preventative measures. Some system errors may appear to the intrusion detection system to be intrusions. But, the detection of these errors increases the overall survivability of the system, so unintentional detection will be considered desirable and not precluded.

Intruders are classified into two groups. *External intruders* do not have any authorized access to the system they attack. *Internal intruders* have some authority, but seek to gain additional ability to take action without legitimate authorization. Internal intruders may act either within or outside their bounds of authorization.

Intrusion detection has traditionally been performed at the operating system (OS) level by comparing expected and observed system resource usage. OS intrusion detection systems can only detect intruders, internal or external, who perform specific system actions in a specific sequence or those intruders whose behavior pattern statistically varies from a norm.



**LITERATURE REVIEW**

---

## **2. LITERATURE REVIEW**

### **2.1 FEASIBILITY STUDY:**

System analysis will be performed to determine if it is flexible to design information based on policies and plans of the organization and on user requirements and to eliminate the weakness of the present system.

#### **2.1.1 Existing System:**

The majority of commercial IDS are network based. These IDSs detect attacks by capturing and analyzing network packets. Listening on a network segment or switch, one network based IDS can monitor the network traffic affecting multiple hosts that are connected to the network segment, thereby protecting those hosts. Network based IDSs often consists of a set of single purpose sensors or hosts placed at various points in a network. As the sensors are limited to running the IDS, they can be more easily secured against attack.

#### **Drawbacks of Existing System**

- Network based IDS's may have difficulty processing all packets in a large or busy network and, therefore may fail to recognize an attack launched during period of high traffic. The need to analyze packets quickly also forces vendors to both detect fewer attacks and also detect attacks with as little computing resource as possible which can reduce detection effectiveness.
- Network based IDS's cannot analyze encrypted information. This problem is increasing as more organization are virtual private networks.
- Most network based IDSs cannot tell whether or not in attack was successful, they can only discern that an attack was initiated. This means that after network-based IDS detects an attack, administrators must manually investigate each attacked host to determine whether it was indeed penetrated.

### 2.1.2 Proposed System:

Host based IDS operate on information collected from within an individual computer system. This vantage point allows host based IDS's to analyze activities with great reliability and precision, determining exactly which processes and users are involved in a particular attack on the operating system. Furthermore, unlike network based IDSs, host based IDSs can see the outcome of an attempted attack, as they can directly access and monitor the data files and system processes usually targeted by attacks.

#### Advantages of Proposed System

- Host based IDS are easier to manage, as information must be configured and managed for every host monitored.
- Host based IDSs, with their ability to monitor events local to a host, can detect attacks that cannot be seen by network based IDS.
- Host based IDSs can often operate in an environment in which network traffic is encrypted, when the host-based information sources are generated before data is encrypted and/or after the data is decrypted at the destination host.
- Host based IDSs are unaffected by switched network.



## 2.2 Hardware Requirements:

Processor	:	Pentium III
Processor speed	:	1.5 GHZ
Memory (RAM)	:	256MB
Hard disk	:	40GB
Floppy drive	:	3 1/2" 1.44MB drive
Monitor	:	15" color monitor
Keyboard	:	Samsung 107 keys
Mouse	:	Samsung scroll mouse

## 2.3 Software Requirements:

Operating System	-	Linux 9.0
Language	-	C & Shell Scripting
Kernel	-	2.4.20-8

## **2.4 SOFTWARE OVERVIEW:**

The project entitled “INTRUSION DETECTION SYSTEM” is developed with the platform LINUX and the scripting language used is C and shell scripting.

### **ABOUT LINUX:**

LINUX was born of a humble beginning. This brings us to university of Finland student Linus Torvalds, circa 1991. Torvalds began working on an operating system of his own. His first prototype, Version 0.01, was born in August of 1991. The first official version of LINUX, Version 0.02, was released on October 6<sup>th</sup>, 1991. Today, LINUX has evolved into a full-blown operating system that, in many ways, rivals commercial systems with the amount of interest it has garnered, LINUX has now become an actively developing and improving system. While many people do contribute to the development of LINUX, Kernel features has still controlled by Linus Torvalds.

### **LINUX FEATURES:**

There are many reasons to use LINUX.

### **FREE:**

LINUX is licensed under the free software foundations GNU General Public License (GPL). According to the terms of the License, anything using it must make available the source code used to build the software, and anything using source code from the software must be licensed under the GPL. This perpetuates the availability of the source code.

## **OPEN SOURCE:**

This implies that the source code used to create the application is made available at no charge to the General Public. Users configuration to the source code, which may be, in tern, merged into the distributed binary form of the application, are generally welcome. This is the way LINUX has come to thrive in.

## **NETWORK:**

A network is a group of computers and devices connected together in order to communicate and work with each other. LINUX was built with networking in mind and all distributions include the necessary programs and utilities to attached to and function on a network.

## **POWER:**

LINUX has the ability to transform your legacy hardware into a powerful, well turned machine. Take that old 486 or Pentium and turn it into a fully multitasking crash free system.

## **INTERACTIVE:**

LINUX allows user to interact with the system, entering commands that are executed immediately (rather than the mainframe method of queuing commands to be run in a batch).

## **MULTIUSER:**

Linux allows multiple people to access the same computer at the same time, differentiate between them, and understand which process belong to whom.

## **MULTITASKING:**

Linux is capable of carrying out more than one task at the same time.

## **MULTIPROCESSING SYSTEM:**

Linux also supports multiple CPUs on the same computer. System with multiple processors performs faster than single CPU system because the processors can combine forces to work on one task between multiple processors via a technique called multithreading.

## **LOG FILES IN LINUX:**

Linux keeps detailed records of events within the system. Many programs create these records, known as log files. The system administrator can refer to the log files to determine the status of the system, watch for intruders, or look for data about a particular program or event.

## **THE PURPOSE OF LINUX LOG FILES:**

On any Linux system, many events go on in the background as users log in and do their work. Daemons are special purpose background processors design to watch for network activity, run other programs, and monitor user actions. The status information collected by daemons is not displayed on the screen. Instead, it is written to log files, which we can then review. Among other things, log files allow a system administrator to:

- Check for potential security problems, such as repeated login failures or a program that is stopped and restarted without the knowledge of the system administrator.

- Review what was happening on the system in the movements before a major problem occurred.
- Manage the system load by computer statistics based on the log file information.

### **ABOUT C:**

The programming language C was developed in 1972 by Dennis Ritchie at AT & T Bell Laboratory, Murray Hill, New Jersey.

C proved to be an excellent programming language for writing system programs.

The UNIX operating system, C compiler and all UNIX applications software are written in C.

The C compiler combines the capability of an assembly language with the features of a high level language, and therefore it is well suited for writing both system software and business packages.

Programs written in C are efficient and fast. This is due to its variety of data type and powerful operators. C is highly portable. This means that C programs written for one computer can be run on another with little or no modifications.

C language is well suited for structured programming, thus requiring the user to think of a problem in terms of function modules or blocks. A proper collection of these modules would make the computer programs. This modular structure makes program debugging, testing and maintenance easier.

### **ABOUT SHELL SCRIPT:**

The shell is the command interpreter, or command-line environment, for Linux. A command interpreter is a program that accepts from the keyboard and uses that input to launch commands or otherwise control the computer system. There are hundreds of small utility programs designed to run from the command line.



A shell program is a sequence of one or more commands stored in a file. This file is executable, and can be called a shell procedure, the shell script. The shell program has some special built in commands that can be used in a shell script for iteration, conditional execution and defining local variables.

A particularly important feature of a Linux shell is that it gives users the ability to write scripts that the shell can execute. In general terms, a script is a text file that can be interpreted or executed by another program.

*DETAILS OF THE METHODOLOGY  
EMPLOYED*

---

### **3. DETAILS OF THE METHODOLOGY EMPLOYED**

When a user of an information system takes an action that the user was not legally allowed to take, it is called intrusion. The intruder may come from outside, or the intruder may be an insider who exceeds his limited authority to take action. Whether or not the action is detrimental, it is of concern because it might be detrimental to the health of the system or to the service provided by the system.

A system that tries to identify attempts to hack or break into a computer system or to misuse it. IDSs may monitor packets passing over the network, monitor system files, monitor log files, or set up deception systems that attempt to trap hackers.

Intrusion detection involves determining that some entity, an intruder, has attempted to gain, or worse, has gained unauthorized access to the system. Casual observation shows that none of the automated detection approaches seek to identify an intruder before that intruder initiates interaction with the system. Of course, system administrators routinely take actions to prevent intrusion. These can include requiring passwords to be submitted before a user can gain any access to the system, fixing known vulnerabilities that an intruder might try to exploit in order to gain unauthorized access, blocking some or all network access, as well as restriction of physical access. Intrusion detection systems are used in addition to such preventative measures. Some system errors may appear to the intrusion detection system to be intrusions. But, the detection of these errors increases the overall survivability of the system, so unintentional detection will be considered desirable and not precluded.

Intrusion detection systems are usually based on the premise that the operating system, as well as the intrusion detection software, continues to function for at least some period of time after an intrusion so that it can alert administrators and support subsequent remedial action.

Intruders are classified into two groups. External intruders do not have any authorized access to the system they attack. Internal intruders have some authority, but seek to gain additional ability to take action without legitimate authorization. Internal intruders may act either within or outside their bounds of authorization.

Intrusion detection has traditionally been performed at the operating system (OS) level by comparing expected and observed system resource usage. OS intrusion detection systems can only detect intruders, internal or external, who perform specific system actions in a specific sequence or those intruders whose behavior pattern statistically varies from a norm.

### **3.1 Host Based Intrusion Detection:**

Host-based intrusion detection started in the early 1980s before networks were as prevalent, complex and interconnected as they are today. In this simpler environment, it was common practice to review audit logs for suspicious activity. Intrusions were sufficiently rare that after the fact analysis proved adequate to prevent future attacks.

Today's host-based intrusion detection systems remain a powerful tool for understanding previous attacks and determining proper methods to defeat their future application. Host-based IDS still use audit logs, but they are much more automated, having evolved sophisticated and responsive detection techniques. Host based IDS typically monitor system, event, and security logs on Windows NT and syslog in Unix environments. When any of these files change, the IDS compare the new log entry with attack signatures to see if there is a match. If so, the system responds with administrator alerts and other calls to action.

Host-based IDS have grown to include other technologies. One popular method for detecting intrusions checks key system files and executables via checksums at regular intervals for unexpected changes. The timeliness of the response is in direct relation to the frequency of the polling interval. Finally, some products listen to port activity and alert administrators when specific ports are accessed. This type of detection brings an elementary level of network-based intrusion detection into the host-based environment.

## FEATURES OF HOST BASED IDS:

1. *Verifies success or failure of an attack* – Since host-based IDS use logs containing events that have actually occurred, they can measure whether an attack was successful or not with greater accuracy and fewer false positives can network-based systems. In this respect, host based IDS make an excellent complement to network-based intrusion detection, with the network component providing early warning and the host component providing verification of whether an attack was successful or not.
2. *Monitors specific system activities* – host-based IDS monitor user and file access activity, including file accesses, changes to file permissions, attempts to install new executables and/or attempts to access privileged services. For example, a host-based IDS can monitor all user logon and logoff activity, as well as what each user does while connected to the network. It is very difficult for a network-based system to provide this level of event detail.

Host-based technology can also monitor activities that are normally executed only by an administrator. Operating systems log any event where user accounts are added, deleted, or modified. The host-based IDS can detect an improper change as soon as it is executed. Host-based IDS can also audit policy changes that affect what systems track in their logs. Finally, host-based systems can monitor changes to key system files and executables. Attempts to overwrite vital system files, or to install Trojan horses or backdoors, can be detected and stopped. Network-based systems sometimes miss this kind of activity.

3. *Detects attacks that network-based systems miss* - Host-based systems can detect attacks that cannot be seen by network-based products. For example, attacks from the keyboard of a critical server do not cross the network, and so cannot be seen by a network-based intrusion detection system.
4. *Well-suited for encrypted and switched environments* – Since host-based systems reside on various hosts throughout an enterprise, they can overcome some of the deployment challenges faced by network-based intrusion detection in switched and encrypted environments. Switches allow large networks to be

managed as many smaller network segments. As a result, it can be difficult to identify the best locations for deploying network-based IDS to achieve sufficient network coverage. Traffic mirroring and administrative ports on switches can help, but these techniques are not always appropriate. Host-based intrusion detection provides greater visibility in a switched environment by residing on as many critical host as needed. Certain types of encryption also present challenges to network-based intrusion detection. Depending where the encryption resides within the protocol stack, it may leave a networkbased system blind to certain attacks. Host-based IDS do not have this limitation. By the time an operating system, and therefore the host-based system, sees incoming traffic, the data stream has already been de-encrypted.

5. ***Near-real-time detection and response*** - Although host-based intrusion detection does not offer true real-time response, it can come extremely close if implemented correctly. Unlike older systems, which use a process to check the status and content of log files at predefined intervals, many current host-based systems receive an interrupt from the operating system when there is a new log file entry. This new entry can be processed immediately, significantly reducing the time between attack recognition and response. There remains a delay between when the operating system records the event and the host-based system recognizes it, but in many cases an intruder can be detected and stopped before damage is done.
6. ***Requires no additional hardware*** - Host-based intrusion detection resides on existing network infrastructure, including file servers, Web servers, and other shared resources. This efficiency can make host-based systems very cost effective because they do not require another box on the network that requires addressing, maintenance, and management.
7. ***Lower cost of entry*** - While network-based intrusion detection systems can offer wide coverage for little effort, they are often expensive. Deploying a single intrusion detection system can cost more than \$10,000. Host-based intrusion detection systems, on the other hand, are often priced in the hundreds of dollars for a single agent and can be deployed by a customer with limited initial capital outlay.

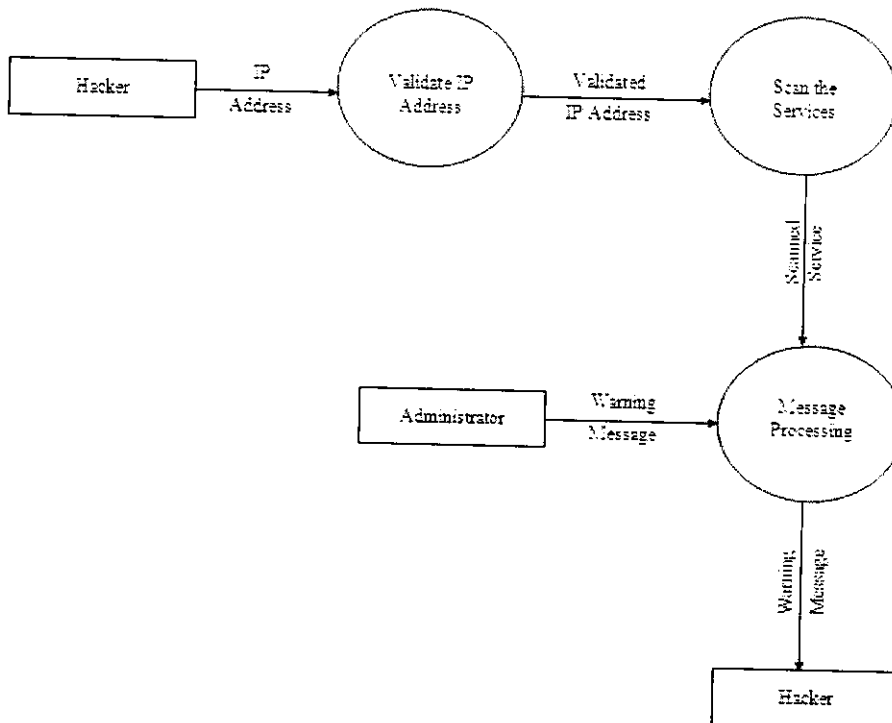
## 3.2 ABOUT THE MODULES:

### 3.2.1 Scanning attacks:

A Scanning attack occurs when an attacker probes a target network or system by sending different kinds of packets. Using the responses received from the target, the attacker can learn many of the system's characteristics and vulnerabilities. Some of them are,

- The topology of a target network.
- The active hosts on the network.
- The server software they are running.
- The software version numbers for all detected software.

### 3.1 Scanning Attack

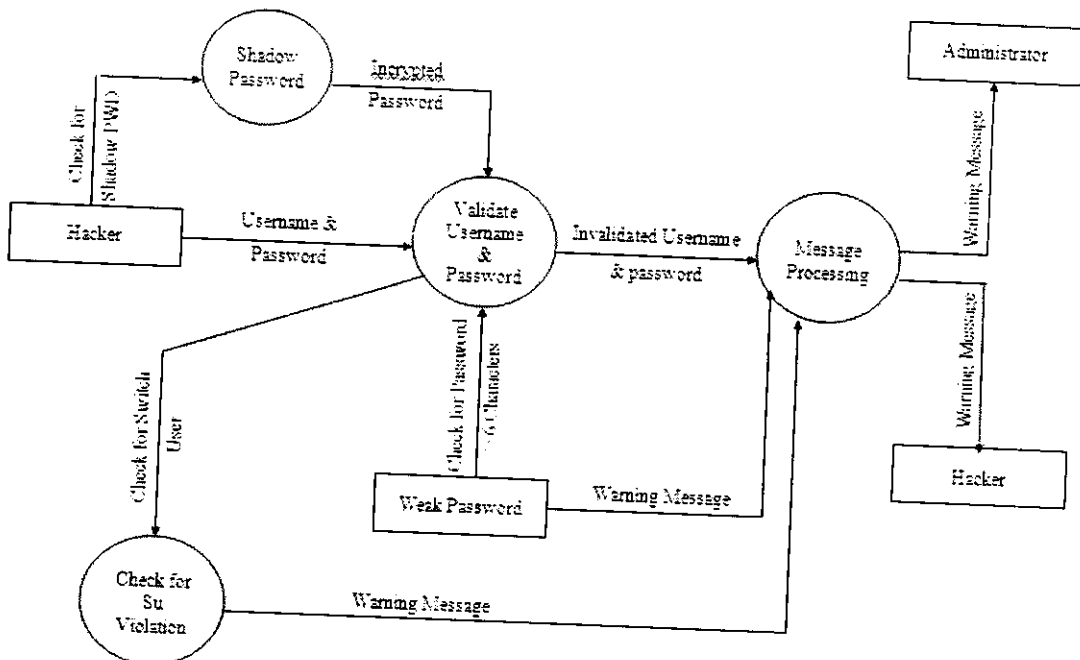


### 3.2.2 Password Attack:

Password attacks involves Authorized User attack and Public User attack. The authorized user attacks are those that start with a legitimate user account on the target system. Most authorized user attacks involve some sort of privilege escalation.

Public user attacks are those launched without any user account or privileged access to the target system. Public user attacks are launched remotely through a network connection using only the public access granted by the target. One typical attack strategy calls for an attacker to use a public user attack to gain initial access to a system. Then once on the system, the attacker uses authorized user attacks to take complete control of the target. It also involves the checking for weak password, shadow password and empty password.

### 3.2 Password Attack

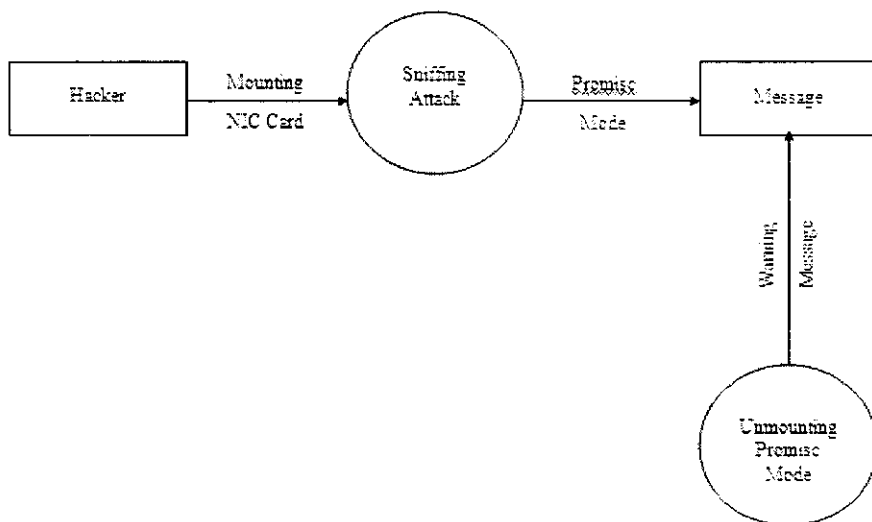




### 3.2.3 Sniffing Attack:

Sniffing means reading the contents of the Network Interface card in the promisc mode of the Linux environment. This attack is done in the promisc mode only. If any hackers tried to view the contents of the Network Interface Card the software will identify the hackers by giving a warning message to the hackers. Our software identifies the hacker and gives a warning message when the hacker tries to enter the promisc mode and all the hacker details will be appended into the recorded file present in the software.

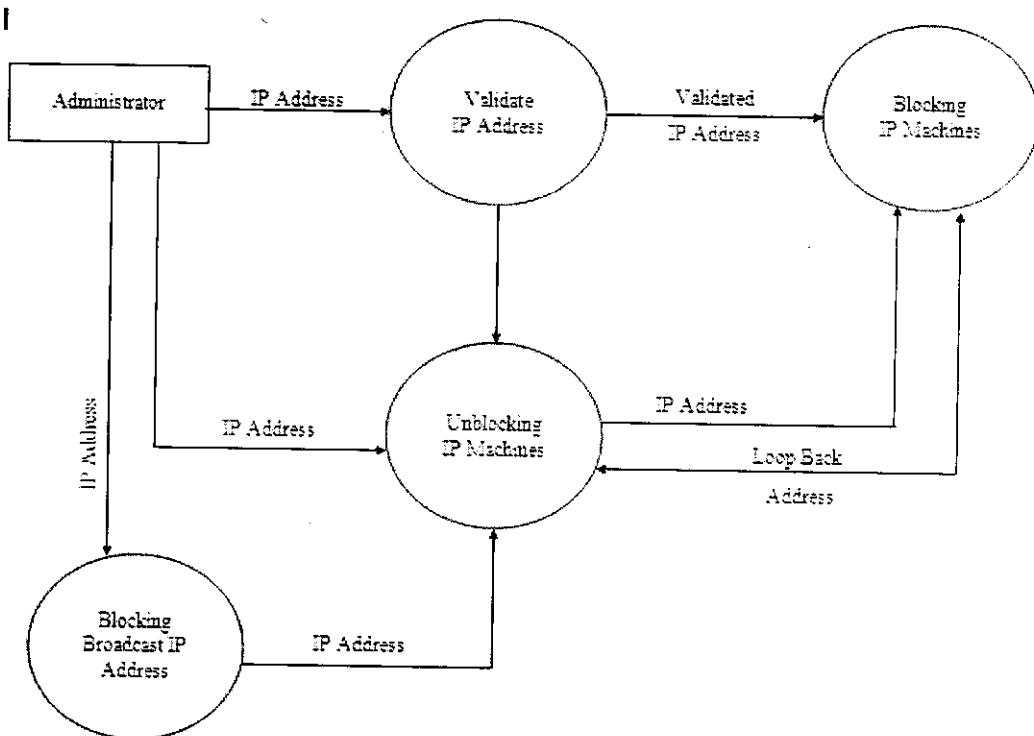
### 3.3 Sniffing Attack



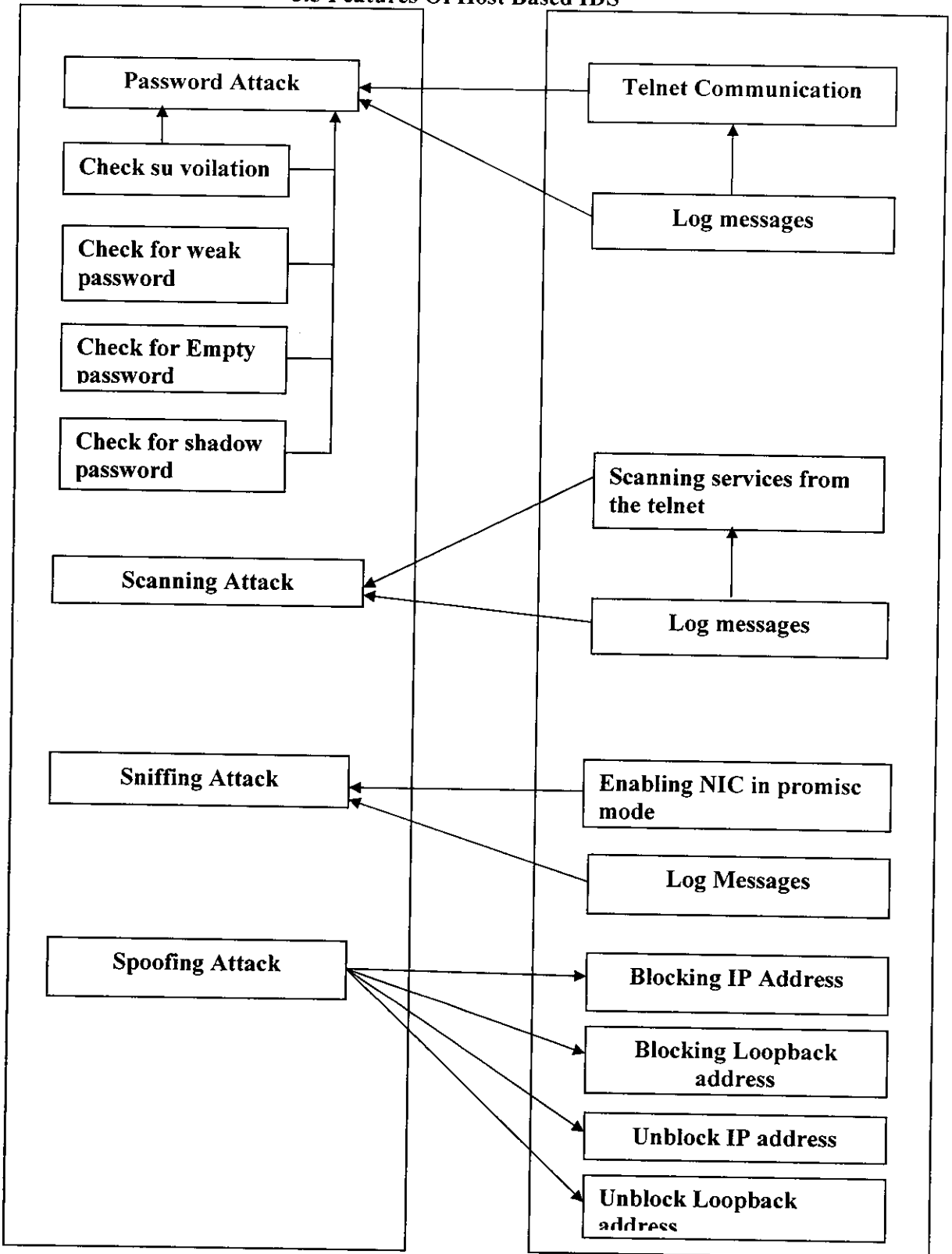
### 3.2.4 Spoofing Attack:

Spoofing means the Internet packet with the local address. When the hackers tried to hack the Internet packet that has the local address the software will identify the hackers by giving a warning message to the hackers and also the hacker details will be appended into the recorded file in the software.

### 3.4 Spoofing Attack



### 3.5 Features Of Host Based IDS



**SYSTEM DEVELOPMENT**

---

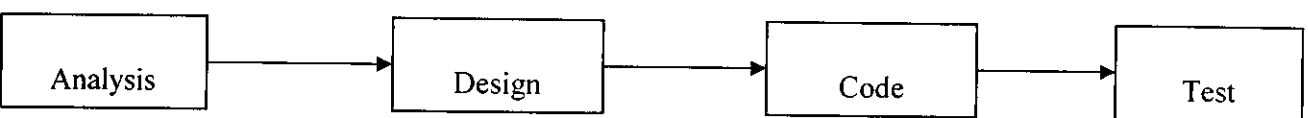
## 4. SYSTEM DEVELOPMENT

The development phase focuses on how the engineer attempts to develop the system. It also deals with how data are to be structured, how the function is to be implemented within a software architecture, how procedural details are to be implemented, how interfaces are characterized, how the design will be translated into a programming, and how testing will be performed. The methods applied during the development phase will vary but three specific technical tasks should always occur.

- ❖ The software design
- ❖ Code generation
- ❖ Software testing

The classical life cycle model is the linear sequential model! is a systematic and sequential approach to software development. It progress through analysis, design, coding, testing and support.

### 4.1 Software Development



#### **SYSTEM SCOPE:**

The system group has changed with the responsibility to develop a new system to meet the requirements and design and development of a new information system. The source of these study facts is variety of users at all level through out organization.

## **4.1 STAGES OF DEVELOPMENT OF SYSTEM:**

- ✓ Feasibility assessment
- ✓ Requirement analysis
- ✓ External design
- ✓ Architectural design
- ✓ Detailed design
- ✓ Coding
- ✓ Debugging
- ✓ Maintenance

### **FEASIBILITY ASSESSMENT:**

In this stage problem was defined. Criteria for choosing solutions were developed, proposed, possible solutions, estimated costs and benefits of the system and recommended the course of action to be taken.

### **REQUIREMENT ANALYSIS:**

During requirement analysis high-level requirements like the capabilities of the system must provide in order to solve a problem. Function requirements, performance requirements for the hardware specified during the initial planning were elaborated and made more specific in order to characterize features and the proposed system will incorporate.

### **EXTERNAL DESIGN:**

External design of any software development involves conceiving, planning out and specifying the externally observable characteristic of the software product. These characteristics include user displays and report formats external data source and data sinks and the functional characteristics.

### **INTERNAL DESIGN ARCHITECTURAL AND DETAILED DESIGN:**

Internal design involved conceiving, planning out and specifying the internal structure and processing details in order to record the design decisions and to be able to indicate why certain alternations were chosen in preference to others. This phase also includes elaboration of the test plans and provides blue prints of implementation, testing and maintenance activities. The product of internal design is architectural structure specification.

The work products of internal design are architectural structure specification, the details of algorithm and data structure and test plan. In architectural design the conceptual view is refined.

### **DETAILED DESIGN:**

Detailed design involved specifying the algorithmic details concern data representations interconnections among data structures and packaging of the software product. This phase emphasizes more on semantic issues and less synthetic details.

### **CODING:**

This phase involves actual programming, ie, transacting detailed design into source code using appropriate programming language.

### **DEBUGGING:**

This stage was related with removing errors from programs and making them completely error free.

### **MAINTENANCE:**

During this stage the systems were loaded and went put to use. They also get modified accordingly to the requirements of the user. These modifications included making enhancements to system and removing problems

## **4.2 TESTING AND IMPLEMENTATION**

### **4.2.1 SYSTEM TESTING:**

#### **TESTING PHASE:**

The philosophy behind testing is to find errors. The common view of testing is that it is performed to whom that there are no errors in a program. However it is virtually impossible to true that no program will be free and clear of errors. Therefore the most useful approach and practical approach is aid the understanding that testing is the process of executing a program explicit indention of finding errors that is making the program fail.

### **4.2.2 TESTING STRATEGIES:**

#### **CODE TESTING:**

This examines the logic of the program. To follow this test cases are developed such that every path of the program is tested.

#### **SPECIFICATION TESTING:**

Specification testing examines the specification starting what the program should do and how it should perform under various conditions. Then test cases are developed for each conditions and combination of conditions and to be submitted for processing.

#### **UNIT TESTING:**

Unit testing focuses verification effort on the smallest unit of software design the module. Using the detail design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of the tests and the errors detected as a result is limited by the constrained scope established for unit testing. The unit test is always white box oriented and the step can be conducted in parallel for multiple modules. According to unit testing the router system is error free.



### **INTEGRATION TESTING:**

Integration testing is a systematic technique for constructing the program structure while the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested modules and build a program structure that has been dictated by design. There is often a tendency to attempt non-incremental integration. That is to construct the program using a “big bang” approach. All modules are combined in advance. The entire program is tested as a whole and shows usually results. Sets of errors are encountered. Correction is difficult because the isolation of causes is complicated by the vast expanse of the entire program once these errors are corrected new ones appear and the process continues and have endless loop. According to the integration testing in the intrusion detection system all the modules are integrated and tested for its accuracy.

### **VALIDATION TESTING:**

At the culmination of integration testing, software is completely assembled as a package, interfacing errors have been uncovered and corrected and a final serious of software tests validation testing may begin.

Validation can be defined in many ways. But a simple definition is that validation succeeds when the software functions in a manner that can be reasonably expected by the customer, reasonable expectations are defined in the software requirements specification a document that describes all user viable attributes of the software. The specification contains a section called Validation testing approach. The validation testing is also known as beta testing. The intrusion detection system was tested using above specified test.

### **SYSTEM TESTING:**

The purpose of system testing is to identify and correct errors in the candidate system. System testing is the stage of implementation which aims at ensuring that the system accurately and efficiently before the actual operation comments.

No program or system design is perfect communication between the user and the designer is not always complete or clear and time is usually start. The result is errors and more errors. The number and nature of errors in a design depends on several factors.

- Communication between user and the designer.
- The programmer's ability to generate a code that reflects exactly the system specifications.
- The time frame for the design.

Testing is vital to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct the goal will be successfully achieved. Inadequate testing or non-testing its leads to the error. This creates two problems.

- The time lag between the cause and the appearance of the problem.
- The effect of the system on file is records with in the system.

A small system error can conceivably explode into a much larger problem. Effective testing early in the process translates directly into long term cost savings from a reduced number of errors.

Another reason for system testing is its utility as a user oriented vehicle before implementation. The best program is worthless if it does not meet user needs. The system should be tested properly to see whether it needs user requirements.

#### **4.2.3 SYSTEM IMPLEMENTATION:**

System implementation is the stage of the project when the theoretical design is tuned into a working system. If the implementation system stage is not carefully controlled and planned, It can cause chaos. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the users a confidence that the system will work and be effective.

- The implementation stage in a system project in its own right involves Careful-planning investigation of the current system and the constraints and the implementation.
- Training of staff in the newly developed system.

### **DOCUMENTATION:**

Before implementing the system, two important documents should be prepared.

- User manual
- System manual

### **USER MANUAL:**

If explain in detail, to the user about the guidelines and procedures to use various functions such as

### **HOW TO ENTER THE DDS APPLICATION:**

- This system is for the user who works on the Linux operating system.
- User wants to enter DDS application; the user should have the root privileges.

### **HOW TO RETURN FROM APPLICATION:**

In application, after entering the different jobs give stop command to finish giving the number of jobs. After finishing the application control and java keys are pressed successfully.

### **SYSTEM MANUAL:**

It explains all the aspects on design, which is used to mainly for the further maintenance of the system.

## **USER TRAINING AND DEMONSTRATION:**

After the successful completion of acceptance testing, the system is ready to use.

In order to put the system into use, the following activities should be taken care of.

- Preparation of user and system documentation
- User training kit
- Conducting user training with demonstration and hands on.

General testing is given to the user. The main aim of the training would be to furnish the user with a working knowledge of the newly developed system. The users are trained to newly developed system. The user manuals are circulated to the users.

**CONCLUSION**

---

## 5. CONCLUSION

Unauthenticated access denial tool helps to detect and prevent hacking and to achieve availability, high and better performance to the traditional data center model. Formalized service levels within corporate and between businesses will become standard practice. Availability, performance and reliability are chief among the metrics measured. Continuous availability on a technical level tends to reduce business-planning costs. These issues lead to cost avoidance.

The tool prevents the hacking throughout the network and maintains the traffic efficiently among network servers. This vendor-neutral guide to the concepts and terminology of hacking offers practical guidance to planning and implementing the technology in most environments. This benefit includes extreme availability, redundancy/resiliency/replication, and security throughout the network and prevents the hacking.

FUTURE ENHANCEMENT

## **6. FUTURE ENHANCEMENT**

- Network- and host-based intrusion detection integrated into a single system
- Shared management console with a consistent interface for product configuration, policy management and single-event display for notifications from both host and network components
- Integrated event database
- Integrated reporting
- Event correlation capabilities
- Integrated on-line help for incident response
- Unified and consistent installation procedures
- Wireless networks are forecasted to expand rapidly



## REFERENCES

## 7. REFERENCES

### BOOKS:

1. David Nash, "Linux in easy step", Dream Tech Publications.
2. Michael K. Johnson and Erik W. Troan, "Linux Application Development", Red Hat Publications.
3. Yashwan Kanitkar, "Shell Programming".
4. T.F. Lunt., "Detecting Intruders in Computer Systems".
5. D. Denning, "An Intrusion Detection Model".

APPENDIX 1-SOURCE CODE

## 8. APPENDIX

### 8.1 APPENDIX1-SOURCE CODE

#### 8.1.1 C – PROGRAM

#### CODING FOR BLOCK ACCOUNT:

```
#include<stdio.h>
main()
{
    char userName[255];
    char *cmdString, timeString[10];
    char reply[] = "y";
    int time;
    int len;

    while(strcmp(reply, "y") == 0) {
        printf("Enter user Name: ");
        scanf("%s", userName);

        printf("For how many minutes the account should be locked:
");
        scanf("%d", &time);

        len = sizeof("sh BlockAccount.sh ") + sizeof(userName) + 1;
        cmdString = (char *)malloc(len);

        strcpy(cmdString, "echo \"User '");
        strcat(cmdString, userName);
        strcat(cmdString, "' has been blocked successfully!!!\");
        system(cmdString);
        strcpy(cmdString, "");

        strcpy(cmdString, "./BlockAccount.sh ");
        strcat(cmdString, userName);
        sprintf(timeString, " %d", time*60);
        strcat(cmdString, timeString);
        strcat(cmdString, " &> /dev/null &");
        system(cmdString);

        printf("Continue Blocking another user (y/n): ");
        scanf("%s", reply);
    } // end while
} // end main
```

## CODING FOR GETSERVICE DETAILS:

```
#include<netdb.h>

printChar(char c, int count) {
    int i;
    for(i=0; i<count; i++)
        printf("%c", c);
    printf("\n");
}

main() {
    struct servent *sv;

    while((sv = getservent()) != NULL) {
        printChar('-', 79);
        printf("Official Name: %s\n", sv->s_name);
        while(*sv->s_aliases)
            printf("%s ", *sv->s_aliases++);

        printf("Port Number: %d\n", sv->s_port);
        printf("Protocol To be used: %s\n", sv->s_proto);
        printChar('-', 79);
    }
}
```

## CODING FOR IFSTATUS:

```
#include<sys/param.h>
#include<ctype.h>
#include<stdio.h>
#ifdef MAXHOSTNAMELEN
#define MAXHOSTNAMELEN 64
#endif
#include<sys/ioctl.h>
#include<net/if.h>
#include<sys/socket.h>
char *hostName = NULL;
char *programName = NULL;
int verbose = 0;
main(int argc, char **argv)
{
    char *p;
    char hostNameBuf[MAXHOSTNAMELEN+1];
    programName=*argv;
    hostName=hostNameBuf;
    while(--argc)
    {
        if(++argv != '-')
            usage();
        switch(++argv)
        {
            case 'v':
                verbose++;
        }
    }
}
```

```

        break;
    default:
        usage();
        break;
    }
}
if(gethostname(hostNameBuf, sizeof(hostNameBuf))<0)
    printf("\n Hostname not found");
for(p=hostName; *p!='\0'; p++)
{
    if(islower(*p))
        *p=toupper(*p);
}
checkInterfaces();
exit(0);
}

checkInterfaces()
{
    int n, s;
    char cbuf[1024];
    struct ifconf ifc;
    struct ifreq ifr, *ifrp;
    if((s=socket(AF_INET, SOCK_DGRAM, 0))<0)
        printf("\nSocket Error");
    ifc.ifc_buf=cbuf;
    ifc.ifc_len=sizeof(cbuf);
    if(ioctl(s, SIOCGIFCONF, (char *)&ifc)<0)
        printf("\nioctl error");
    close(s);
    ifrp=ifc.ifc_req;
    for(n=ifc.ifc_len/sizeof(struct ifreq); n>0; n--, ifrp++)
    {
        if((s=socket(AF_INET, SOCK_DGRAM, 0))<0)
            printf("\nSocket Error");
        strcpy(ifr.ifr_name, ifrp->ifr_name);
        if(ioctl(s, SIOCGIFFLAGS, (char *)&ifr)<0)
            printf("\nioctl error");
        if(verbose)
        {
            printf("Interfaces
%s: falgs=0x%x\n", ifr.ifr_name, ifr.ifr_flags);
        }
        if(ifr.ifr_flags & IFF_PROMISC)
        {
            printf("WARNING: %s INTERFACES %s IS IN
PROMISCUOUS MODE.\n", hostName, ifr.ifr_name);
        }
        if(ifr.ifr_flags & IFF_DEBUG)
        {
            printf("WARNING: %s INTERFACES %s IS IN GEBUG
MODE.\n", hostName, ifr.ifr_name);
        }
    }
    close(s);
}
}

```

```

usage()
{
    fprintf(stderr, "Usage: %s [-v]\n", programName);
    exit(1);
}

```

## CODING FOR MAIN:

```

#include<stdio.h>
#include<string.h>
#include "put.h"
main()
{
    int ch1,ch2,ch3,ch4,ch5,i,ch21,ch31,choice,ch41,ch42;
    char c1,c2,c3,c4;
    system("clear");
    system("sh loading.sh");
    system("sh setting.sh");
loop:
    system("clear");

    putchar('*');
    putLine(1);
    printf("\n\n\t\t\t INTRUSION DETECTION SYSTEM\n ");
    printf("\n\t1.Password Attack\n\t2.Scanning Attack\n\t");
    printf("3.Sniffing Attack\n\t4.Spoofing
Attack\n\t5.Exit\n");

    putchar('*');
    putLine(1);
    printf("\n\n Enter Your Choice : ");
    scanf("%d",&ch1);
    switch(ch1)
    {
        case 1:
            system("clear");
                for(i=0;i<=78;i++)
                {
                    printf("=");
                }
                printf("\n\t\t\tOptions for password
attack\n\n");
                printf("\t1.Check SU Violation\n\t2.Check Login
Failures\n\t");
                printf("3.Check BLANK Password\n\t4.Check
SHADOW Password\n\t");
                printf("5.Check Weak Password\n\t6.View Recorded
Attack");
                printf("\n\t7.GRUB password\n\t8.Blocked
Users\n\t9.Goto The Main Menu\n");
                for(i=0;i<=78;i++)
                {
                    printf("=");
                }
                printf("\n\n Please Enter Your Choice : ");
                scanf("%d",&ch2);

```





```

        case 2:
            printf("\nService Started
            system("sh
            Successfully\n");
            checkLogin.sh &");
            printf("Press CTRL+C To
            Abort");
            break;
        }
        break;
    case 3:
        printf("\n\t Checking For BLANK
        Password field");
        system("sh passw.sh");
        break;
    case 4:
        printf("\n\t Checking For BLANK
        Password fields");
        system("sh passw.sh");
        printf("\n\t Checking For Shadowed
        Password fields");
        system("sh isShadowed.sh");
        printf("\n\t Checking Shadowed File
        Permission.....");
        checkPerm.sh");
        system("sh
        break;
    case 5:
        printf("Checking For Weak Password");
        system("sh weak.sh");
        system("sh weak1.sh");
        break;
    case 6:
        printf("\n\n\t\t\t Recorded Details");
        PasswordAttack.log");
        system("sh catfile.sh
        break;
    case 7:
        printf("\t\t\t GRUB PASSWORD\n\n\n");
        system("sh grub.sh");
        break;
    case 8:
        system("clear");

        putchar('*');
        putLine(1);
        printf("\t\n\n\t\t BLOKED ACCOUNTS
        DETAILS\n");
        printf("\t\t 1.BLOKED ACCOUNTS
        Name\n\t\t 2.UNBlocking
        ACCOUNTS\n");
        putchar('*');
        putLine(1);
        printf("\t\t Enter The option : ");
        scanf("%d",&ch41);
        switch(ch41)
        {
            case 1:

```

```

        printf("Blocked Users
Details");
        printf("Press CTRL+C To
Abort");
        system("cat /etc/passwd");
        break;
        case 2:
        printf("\nUnlocking Users\n");
        system("sh unlock.sh ");
        printf("Press CTRL+C To
Abort");
        break;
    }
    break;
    case 9:
        goto loop;
}
break;
case 2:
system("clear");
for(i=0;i<=78;i++)
{
    printf("=");
}
printf("\n\t\t\tOptions for Scanning
attack\n\n");
printf("\t1.Check It Now\n\t2.Run It As
Daemon\n\t");
printf("3.View The Service Details\n\t4.View
Recorded Attack");
printf("\n\t5.Goto The Main
Menu\n");
for(i=0;i<=78;i++)
{
    printf("=");
}
//system("gcc -o scanDect scanDect.o");
system("killall scanDect &> r");
system("adduser scanlogd &> r");
system("rm -f r");
// system("./scanDect");
for(i=0;i<=78;i++)
{
    printf("~");
}
printf("\n\n Please Enter Your Choice : ");
scanf("%d",&ch3);
switch(ch3)
{
case 1:
    printf("Process Started.....");
    system("sh checkScan.sh");
    printf("Press CTRL+C To Abort");
    break;
case 2:
    printf("\nService Started
Successfully\n");

```



```

attack\n\n\t");
printf("\n\t\t\tOptions for Spoofing
Address\n\n\t");
printf("1.Block IP Address\n\t2.Block Loop Back
Particular");
printf("3.Broadcast Traffic Address\n\t4.Block
Main Menu");
printf("service\n\t 5.UnBlocking\n\t 6.Goto
putChar('#');
putLine(1);
printf("\n\n Please Enter Your Choice : ");
scanf("%d",&ch5);
switch(ch5)
{
    case 1:
        system("sh BlockparIP.sh");
        break;
    case 2:
        system("sh LoopBTR.sh");
        break;
    case 3:
        system("sh BroadCast.sh");
        break;
    case 4:
        system("sh BlockParser.sh");
        break;
    case 5:
        system("sh UnSpoof.sh");
        break;
    case 6:
        goto loop;
        break;
}
break;

case 5:
    exit(0);
    break;
printf("\n\n\n\n\t");
default:printf("Please Enter The Vaild Option");
}
printf("\n\n\n\n");
for(i=0;i<78;i++)
{
    printf("-");
}
printf("\n\n\tIf u want to continue press 1 else press
0 : ");
scanf("%d",&choice);
if(choice)
{
    system("clear");
    goto loop;
}
else
    exit;
}

```

## CODING FOR PUT:

```
#include<stdio.h>

void putChar(char c) {
    int i=0;
    printf("\n");
    for (i=0;i<=79;i++)
        printf("%c",c);
}

void putLine(int l) {
    int i=0;
    for(i=0;i<l;i++)
        printf("\n");
}
```

## CODING FOR SCANDECT:

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
#include <syslog.h>
#include <sys/times.h>
#include <sys/types.h>
#define __BSD_SOURCE
#define __FAVOR_BSD
#include <netinet/in_system.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#define _SCANLOGD_NETINET
#include <arpa/inet.h>

#include "params.h"
#include "in.h"
#include "in_linux.h"

#ifdef USE_RLOG
#include "rlog.h"
#endif

#define HF_DADDR_CHANGING          0x01
#define HF_SPORT_CHANGING         0x02
#define HF_TOS_CHANGING           0x04
#define HF_TTL_CHANGING           0x08

struct host {
    struct host *next;
```

```

    clock_t timestamp;
    time_t start;
    struct in_addr saddr, daddr;
    unsigned short sport;
    int count;
    int weight;
    unsigned short ports[SCAN_MAX_COUNT - 1];
    unsigned char tos;
    unsigned char ttl;
    unsigned char flags_or;
    unsigned char flags_and;
    unsigned char flags;
};

static struct {
    struct host list[LIST_SIZE];
    struct host *hash[HASH_SIZE];
    int index;
} state;

static int hashfunc(struct in_addr addr)
{
    unsigned int value;
    int hash;

    value = addr.s_addr;
    hash = 0;
    do {
        hash ^= value;
    } while ((value >>= HASH_LOG));

    return hash & (HASH_SIZE - 1);
}

static void do_log(struct host *info)
{
    int limit;
    char s_saddr[32];
    char s_daddr[64 + 8 * SCAN_MAX_COUNT];
    char s_flags[16];
    char s_tos[16];
    char s_ttl[16];
    char s_time[32];
    int index, size;
    unsigned char mask;

    limit = info->count;
prepare:
    snprintf(s_saddr, sizeof(s_saddr),
             (info->flags & HF_SPORT_CHANGING) ? "%s" : "%s:%u",
             inet_ntoa(info->saddr),
             (unsigned int)ntohs(info->sport));

    snprintf(s_daddr, sizeof(s_daddr), "%s%s ports ",
             inet_ntoa(info->daddr),

```

```

        (info->flags & HF_DADDR_CHANGING) ? " and others," :
        "");

        for (index = 0; index < limit; index++) {
            size = strlen(s_daddr);
#ifdef LOG_MAX_LENGTH
            if (size >= LOG_MAX_LENGTH) {
                limit = index;
                break;
            }
#endif
            snprintf(s_daddr + size, sizeof(s_daddr) - size,
                "%u, ", (unsigned int)ntohs(info-
>ports[index]));
        }

        for (index = 0; index < 8; index++) {
            mask = 1 << index;
            if ((info->flags_or & mask) == (info->flags_and &
mask)) {
                s_flags[index] = "fsrpauxy"[index];
                if (info->flags_or & mask)
                    s_flags[index] = toupper(s_flags[index]);
            } else
                s_flags[index] = '?';
        }
        s_flags[index] = 0;

        snprintf(s_tos, sizeof(s_tos),
            (info->flags & HF_TOS_CHANGING) ? "" : ", TOS %02x",
            (unsigned int)info->tos);

        snprintf(s_ttl, sizeof(s_ttl),
            (info->flags & HF_TTL_CHANGING) ? "" : ", TTL %u",
            (unsigned int)info->ttd);

        strftime(s_time, sizeof(s_time), "%X", localtime(&info-
>start));

#ifdef LOG_MAX_LENGTH
        if (strlen(s_saddr) + strlen(s_daddr) +
            strlen(s_tos) + strlen(s_ttl) + strlen(s_time) +
            (4 + 5 + 8 + 2) > LOG_MAX_LENGTH) {
            if (--limit > 0) goto prepare;
        }
#endif

        syslog(SYSLOG_LEVEL,
            "%s to %s..., %s%s %s",
            s_saddr, s_daddr, s_flags, s_tos, s_ttl, s_time);

#ifdef USE_RLOG
        rlog(
            "%s to %s..., %s%s %s",
            s_saddr, s_daddr, s_flags, s_tos, s_ttl, s_time);
#endif
    }
}

```

```

static void safe_log(struct host *info)
{
    static clock_t last = 0;
    static int count = 0;
    clock_t now;

    now = info->timestamp;
    if (now - last > LOG_DELAY_THRESHOLD || now < last) count =
0;
    if (++count <= LOG_COUNT_THRESHOLD + 1) last = now;

    if (count <= LOG_COUNT_THRESHOLD) {
        do_log(info);
    } else if (count == LOG_COUNT_THRESHOLD + 1) {
        syslog(SYSLOG_LEVEL, "More possible port scans
follow");
#ifdef USE_RLOG
        rlog("More possible port scans follow");
#endif
    }
}

static void process_packet(struct header *packet, int size)
{
    struct ip *ip;
    struct tcphdr *tcp;
    struct in_addr addr;
    unsigned short port;
    unsigned char flags;
    struct tms buf;
    clock_t now;
    struct host *current, *last, **head;
    int hash, index, count;

    ip = &packet->ip;
    tcp = (struct tcphdr *)((char *)packet + ((int)ip->ip_hl <<
2));

    if (ip->ip_p != IPPROTO_TCP || (ip->ip_off &
htons(IP_OFFMASK)) ||
        (char *)tcp + sizeof(struct tcphdr) > (char *)packet +
size)
        return;

    addr = ip->ip_src;
    port = tcp->th_dport;
    flags = tcp->th_flags;

    if (!addr.s_addr) return;

    now = times(&buf);

    count = 0;
    last = NULL;
    if ((current = *(head = &state.hash[hash =
hashfunc(addr)])))

```



```

do {
    if (current->saddr.s_addr == addr.s_addr) break;
    count++;
    if (current->next) last = current;
} while ((current = current->next));

if (current)
if (now - current->timestamp <= SCAN_DELAY_THRESHOLD &&
now >= current->timestamp) {
    for (index = 0; index < current->count; index++)
        if (current->ports[index] == port) {
            current->flags_or |= flags;
            current->flags_and &= flags;
            return;
        }

    if (flags & (TH_ACK | TH_RST)) return;

    current->timestamp = now;

    if (current->weight >= SCAN_WEIGHT_THRESHOLD) return;

    current->flags_or |= flags;
    current->flags_and &= flags;

    if (current->daddr.s_addr != ip->ip_dst.s_addr)
        current->flags |= HF_DADDR_CHANGING;
    if (current->sport != tcp->th_sport)
        current->flags |= HF_SPORT_CHANGING;
    if (current->tos != ip->ip_tos)
        current->flags |= HF_TOS_CHANGING;
    if (current->ttl != ip->ip_ttl)
        current->flags |= HF_TTL_CHANGING;

    current->weight += (ntohs(port) < 1024) ?
        PORT_WEIGHT_PRIV : PORT_WEIGHT_HIGH;

    if (current->weight >= SCAN_WEIGHT_THRESHOLD) {
        safe_log(current);
        return;
    }

    if (current->count < SCAN_MAX_COUNT)
        current->ports[current->count++] = port;

    return;
}

if (current) {
    current->saddr.s_addr = 0;

    if (last)
        last->next = last->next->next;
    else if (*head)
        *head = (*head)->next;
    last = NULL;
}

```

```

    if (flags & TH_ACK) return;

    if (count >= HASH_MAX && last) last->next = NULL;

    if (state.list[state.index].saddr.s_addr)
        head =
&state.hash[hashfunc(state.list[state.index].saddr)];
    else
        head = &last;
    last = NULL;
    if ((current = *head))
    do {
        if (current == &state.list[state.index]) break;
        last = current;
    } while ((current = current->next));

    if (current) {
        if (last)
            last->next = last->next->next;
        else if (*head)
            *head = (*head)->next;
    }

    current = &state.list[state.index++];
    if (state.index >= LIST_SIZE) state.index = 0;

    head = &state.hash[hash];
    current->next = *head;
    *head = current;

    current->timestamp = now;
    current->start = time(NULL);
    current->saddr = addr;
    current->daddr = ip->ip_dst;
    current->sport = tcp->th_sport;
    current->count = 1;
    current->weight = (ntohs(port) < 1024) ?
        PORT_WEIGHT_PRIV : PORT_WEIGHT_HIGH;
    current->ports[0] = port;
    current->tos = ip->ip_tos;
    current->ttl = ip->ip_ttl;
    current->flags_or = current->flags_and = flags;
    current->flags = 0;
}

void pexit(char *name)
{
    perror(name);
    exit(1);
}

#ifdef SCANLOGD_USER
static void drop_root(void)
{
    struct passwd *pw;
    gid_t groups[2];

```

```

    errno = 0;
    if (!(pw = getpwnam(SCANLOGD_USER))) {
        fprintf(stderr,
            "getpwnam(\"" SCANLOGD_USER "\"): %s\n",
            errno ? strerror(errno) : "No such user");
        exit(1);
    }

    groups[0] = groups[1] = pw->pw_gid;
    if (setgroups(1, groups)) pexit("setgroups");
    if (setgid(pw->pw_gid)) pexit("setgid");
    if (setuid(pw->pw_uid)) pexit("setuid");
}
#else
static void cleanup(int signum)
{
    exit(0);
}
#endif

int main(void)
{
#ifdef SCANLOGD_USER
    signal(SIGTERM, cleanup);
    signal(SIGINT, cleanup);
    signal(SIGHUP, cleanup);
#endif

    if (in_init()) return 1;

    chdir("/");
    setsid();

#ifdef USE_RLOG
    errno = 0;
    if (openrlog(RLOG_ID)) {
        fprintf(stderr,
            "openrlog(\"" RLOG_ID "\"): %s\n",
            errno ? strerror(errno) : "Failed");
        return 1;
    }
#endif

#ifdef SCANLOGD_USER
    drop_root();
#endif

    switch (fork()) {
    case -1:
        pexit("fork");

    case 0:
        break;

    default:
        _exit(0);
    }
}

```

```

    }

    setsid();

    memset(&state, 0, sizeof(state));

    openlog(SYSLOG_IDENT, 0, SYSLOG_FACILITY);

    in_run(process_packet);

    return 1;
}

```

### CODING FOR SCANNER:

```

#include<stdio.h>
#include<unistd.h>

#define MAXSERVICES 100

void printChar(char c, int count) {
    int i;
    for(i=0; i<count; i++)
        printf("%c", c);

    printf("\n");
}

int main() {
    FILE *fp, *fp1, *fp2;
    char line[81], serviceName[MAXSERVICES][30];
    char portData[MAXSERVICES][20], runStatus[MAXSERVICES][20];
    char serviceNumbers[81], allowOrRestrict[20];
    char hostToBeScanned[50], cmdStr[500];

    int totalLines = 0, count =0, servNumber;
    int number;

    //system("clear");
    printf("\n\n\n\n");
    printChar('#', 79);
    printf("Port Scanner Started.....\n\n");
    printf("Enter the machine Name/IP to scanned: ");
    scanf("%s", hostToBeScanned);
    fp2 = fopen("ip", "w");
    fprintf(fp2, hostToBeScanned);
    fclose(fp2);
    sprintf(cmdStr, "nmap %s > nmapdata", hostToBeScanned);
    system(cmdStr);
    fp = fopen("nmapdata", "r");
    fp1 = fopen("serviceDetails", "w");

    printf("\nScanned Results are.....\n");
    while(fgets(line, 80, fp)!=NULL) {

```

```

    totalLines += 1;
    if(totalLines <= 5)
        continue;
    if(strcmp(line, "\n") == 0)
        break;
    count += 1;
    sscanf(line, "%s %s", portData[count], runStatus[count],
    serviceName[count]);
    fprintf(stdout, "\t\t%d %s\n", count, serviceName[count]);
}

printf("\nThe above services were running on 'es' ..\n",
hostToBeScanned);
printChar('#', 79);
unlink("serviceDetails");
unlink("nmapdata");
system ("sh log.sh");
} // end main

```

### CODING FOR SERVICE MANAGER:

```

#include<stdio.h>

main(int argc, char *argv[]) {
    char *cmdString;
    char action[7];
    int len;
    if(strcmp(argv[1], "start") == 0) {
        strcpy(action, " start");
    } else if(strcmp(argv[1], "stop") == 0) {
        strcpy(action, " stop");
    } else if(strcmp(argv[1], "status") == 0) {
        strcpy(action, " status");
    } else {
        printf("Usage: <%s> <start | stop | status> <service1>
<service2> ... <serviceN>\n", argv[0]);
        exit(-1);
    }

    while(argc-- > 2) {
        // LOGIC for starting multiple services
        len = strlen("service ") + strlen(argv[argc]) +
strlen(action);
        cmdString = (char *)malloc(len + 1);
        strcpy(cmdString, ""); // clear previous content if any.
        strcat(cmdString, "service ");
        strcat(cmdString, argv[argc]);
        strcat(cmdString, action);
        printf("command is: '%s'\n", cmdString);
        system(cmdString);
    } // end while
} // end main

```

## 8.1.2 JAVA – PROGRAM

### CODING FOR GETSERVICE DETAILS:

```
public class GetServiceDetails {
    int printChar(char c, int count) {
        int i = 0;
        for (i = 0; i < count; i++) {
            System.out.print(c);
        }
        System.out.println("");
        return 0;
    }

    public static void main(String[] args) {
        Servent * sv;

        while ((sv = getservent()) != 0) {
            printChar('-', 79);
            System.out.println("Official Name: " + sv.s_name);
            while ( * sv.s_aliases) {
                System.out.print(( * sv.s_aliases++) + " ");
            }
            System.out.println("Port Number: " + sv.s_port);
            System.out.println("Protocol To be used: " +
sv.s_proto);
            printChar('-', 79);
        }
    }
}
```

### CODING FOR IFSTATUS:

```
import java.io.IOException;
public class Ifstatus {
    public final static int MAXHOSTNAMELEN = 64;
    String hostName = null;
    Pint programName = new Pint();
    programName = 0;
    int verbose = 0;

    public static void main(String[] args) {
        String[] argv = new String[args.length + 1];
        argv[0] = "Ifstatus";
        for (int i = 1; i < args.length; i++) {
            argv[i] = args[i - 1];
        }
        Ifstatus instance = new Ifstatus();
        System.exit(instance.main(argv.length, argv));
    }

    int main(int argc, String[] argv) {
        String p = null;
```

```

String hostNameBuf = new String();
programName = argv.charAt(0);
hostName = hostNameBuf;
while (--argc != 0) {
    if ( * *++(int) argv != '-') {
        usage();
    }
    switch ( *++argv.charAt(0)) {
        case 'v':
            verbose++;
            break;
        default :
            usage();
            break;
    }
}
if (hostNameBuf =
INetAddress.getLocalHost().getHostName() < 0) {
    System.out.print("\n Hostname not found");
}
for (p = hostName;p.charAt(0) != '\0';p++) {
    if (Character.isLowerCase(p.charAt(0))) {
        p = CStringUtils.setCharAt(p, 0,
Character.toUpperCase(p.charAt(0)));
    }
}
checkInterfaces();
return 0;
}

int checkInterfaces() {
    int n = 0;
    int s = 0;
    String cbuf = new String();
    Ifconf ifc;
    Ifreq ifr;
    * ifrp;
    s = socket(AF_INET, SOCK_DGRAM, 0);
    if (s < 0) {
        System.out.print("\nSocket Error");
    }
    ifc.ifc_buf = cbuf;
    ifc.ifc_len = sizeof ();
    try {
        ioctl(s, SIOCGIFCONF, (char) & ifc);
    }
    catch (Exception e) {
        System.out.print("\niioctl error");
    }
    try {
        close(s);
        ifrp = ifc.ifc_req;
        for (n = ifc.ifc_len / sizeof ();n > 0;n--, ifrp++)

            s = socket(AF_INET, SOCK_DGRAM, 0);
            if (s < 0) {
                System.out.print("\nSocket Error");
            }
    }
}

```

```

    }
    ifr.ifr_name = ifrp.ifr_name;
    try {
        ioctl(s, SIOCGIFFLAGS, (char) & ifr);
    }
    catch (IOException e) {
        System.out.println("\nioctl error");
    }
    if (verbose != 0) {
        System.out.println("Interfaces " +
ifr.ifr_name + ":falgs=0x" + ifr.ifr_flags);
    }
    if ((ifr.ifr_flags & IFF_PROMISC) != 0) {
        System.out.println("WARNING:" + hostName + "
INTERFACES " + ifr.ifr_name + " IS IN PROMISCUOUS MODE.");
    }
    if ((ifr.ifr_flags & IFF_DEBUG) != 0) {
        System.out.println("WARNING:" + hostName + "
INTERFACES " + ifr.ifr_name + " IS IN GEBUG MODE.");
    }
    }
    return close(s);
}
catch (IOException e) {
    System.err.println("IO Exception:" + e);
}
}

int usage() {
    System.err.println("Usage:" + programName + " [-v]");
    return System.exit(1);
}
}

```

### CODING FOR MAIN:

```

import java.io.IOException;

/* main */

public class Main {
    public static void main(String[] args) {
        Pint ch1 = new Pint();
        Pint ch2 = new Pint();
        int ch3 = 0;
        int ch4 = 0;
        int ch5 = 0;
        int i = 0;
        int ch21 = 0;
        int ch31 = 0;
        int choice = 0;
        int ch41 = 0;
        int ch42 = 0;
        char c1 = '\u0000';
        char c2 = '\u0000';
    }
}

```



```

char c3 = '\u0000';
char c4 = '\u0000';
Runtime.getRuntime().exec("clear");
Runtime.getRuntime().exec("sh loading.sh");
Runtime.getRuntime().exec("sh setting.sh");
Runtime.getRuntime().exec("clear");

putChar('*');
putLine(1);
System.out.print("\n\n\t\t\t INTRUSION DETECTION
SYSTEM\n ");
System.out.print("\n\t1.Password Attack\n\t2.Scanning
Attack\n\t");
System.out.println("3.Sniffing Attack\n\t4.Spoofing
Attack\n\t5.Exit");

putChar('*');
putLine(1);
System.out.print("\n\n Enter Your Choice : ");
try {
    Scanner scanner22 = new
Scanner(CIUtils.stdin.readLine());
    ch1 = scanner22.readString();
    switch (ch1.value) {
        case 1:
            Runtime.getRuntime().exec("clear");
            for (i = 0;i <= 78;i++) {
                System.out.print("=");
            }
            System.out.println("\n\t\t\tOptions for
password attack\n");
            System.out.print("\t1.Check SU
Violation\n\t2.Check Login Failures\n\t");
            System.out.print("3.Check BLANK
Password\n\t4.Check SHADOW Password\n\t");
            System.out.print("5.Check Weak
Password\n\t6.View Recorded Attack");
            System.out.println("\n\t7.GRUB
password\n\t8.Blocked Users\n\t9.Goto The Main Menu");
            for (i = 0;i <= 78;i++) {
                System.out.print("=");
            }
            System.out.print("\n\n Please Enter Your
Choice : ");
            Scanner scanner23 = new
Scanner(CIUtils.stdin.readLine());
            ch2 = scanner23.readString();
        }
    }
catch (IOException e) {
    System.err.println("IO Exception:" + e);
}
}
switch (ch2) {
case 1:
    Runtime.getRuntime().exec("clear");
    putChar('*');
    putLine(1);

```



```

        Successfully");
        System.out.println("\nService Started
        checkLogin.sh &");
        Runtime.getRuntime().exec("sh
        Abort");
        System.out.print("Press CTRL+C to
        break;
    }
    break;
    case 3:
        System.out.print("\n\t Checking For BLANK
        Password field");
        Runtime.getRuntime().exec("sh passw.sh");
        break;
    case 4:
        System.out.print("\n\t Checking For BLANK
        Password fields");
        Runtime.getRuntime().exec("sh passw.sh");
        System.out.print("\n\tChecking For Shadowed
        Password fields");
        Runtime.getRuntime().exec("sh isShadowed.sh");
        System.out.print("\n\tChecking Shadowed File
        Permission.....");
        Runtime.getRuntime().exec("sh checkPerm.sh");
        break;
    case 5:
        System.out.print("Checking For Weak Password");
        Runtime.getRuntime().exec("sh weak.sh");
        Runtime.getRuntime().exec("sh weak1.sh");
        break;
    case 6:
        System.out.print("\n\n\t\t\t Recorded Details");
        Runtime.getRuntime().exec("sh catfile.sh
        PasswordAttack.log");
        break;
    case 7:
        System.out.println("\t\t\t GRUB PASSWORD\n\n");
        Runtime.getRuntime().exec("sh grub.sh");
        break;

    case 8:
        Runtime.getRuntime().exec("clear");
        putChar('*');
        putLine(1);
        System.out.println("\t\n\n\t\t BLOCKED ACCOUNTS
        DETAILS");
        System.out.println("\t\t 1.BLOCKED ACCOUNTS
        Name\n\t\t 2.UNBlocking ACCOUNTS");
        putChar('*');
        putLine(1);
        System.out.print("\t\t Enter The option : ");
        Scanner scanner32 = new
        Scanner(CIUtils.stdin.readLine());
    }
    catch (IOException e) {
        System.err.println("IO Exception:" + e);
    }
}

```

```

        ch41 = scanner32.readString();
    }

    switch (ch41) {
        case 1:
            System.out.print("Blocked Users Details");
            System.out.print("Press CTRL+C To Abort");
            Runtime.getRuntime().exec("cat /etc/user1");
            break;
        case 2:
            System.out.println("\nUnlocking Users");
            Runtime.getRuntime().exec("sh unlock.sh ");
            System.out.print("Press CTRL+C To Abort");
            break;
    }

    break;
case 9:
    goto loop;
}
break;
case 2:
Runtime.getRuntime().exec("clear");
for (i = null;i <= 78;i++) {
    System.out.print("=");
}
System.out.println("\n\t\t\tOptions for Scanning attack\n");
System.out.print("\t1.Check It Now\n\t2.Run It As Daemon\n\t");
System.out.print("3.View The Service Details\n\t4.View Recorded
Attack");
System.out.println("\n\t5.Goto The Main Menu");
for (i = null;i <= 78;i++) {
    System.out.print("=");
}
Runtime.getRuntime().exec("adduser scanlogd &> r");
Runtime.getRuntime().exec("rm -f r");
// system("./scanDect");
for (i = null;i <= 78;i++) {
    System.out.print("~");
}
System.out.print("\n\n Please Enter Your Choice : ");
try {
    Scanner scanner33 = new Scanner(CIUtils.stdin.readLine());
}
catch (IOException e) {
    System.err.println("IO Exception:" + e);
}
ch3 = scanner33.readString();
switch (ch3) {
    case 1:
        System.out.print("Process Started.....");
        Runtime.getRuntime().exec("sh checkScan.sh");
        System.out.print("Press CTRL+C To Abort");
        break;
    case 2:

```

```

        System.out.println("\nService Started Successfully");
        System.out.print("CTRL+C to abort");
        Runtime.getRuntime().exec("sh checkScan.sh &");
        break;
    }
}
case 3:
    System.out.println("\n\t\t\tView The Service Details\n");
    System.out.print("\t\t\t\t\t Probing Service Details\n:");
    Runtime.getRuntime().exec("gcc GetServiceDetails.java -o
servicedetails");
    Runtime.getRuntime().exec("./servicedetails| less");
    break;
case 4:
    System.out.print("VIEW THE RECORDED ATTACK ");
    Runtime.getRuntime().exec("sh catfile.sh
ScanningAttack.log");
    break;
}
break;
case 3:
    Runtime.getRuntime().exec("clear");
    putChar('#');
    putLine(1);
    System.out.print("\t1.Check It Now\n\t2.Run It As Daemon\n\t");
    System.out.println("3.View The Recorded Attack\n\t4.Goto The
Main Menu");
    putChar('#');
    putLine(1);
    System.out.print("\n\n Please Enter Your Choice : ");
    try {
        Scanner scanner34 = new Scanner(CIOUtils.stdin.readLine());
        ch4 = scanner34.readString();
        switch (ch4) {
            case 1:
                System.out.print("\n\n\t\t\t\t\t SNIFFING ATTACK");
                Runtime.getRuntime().exec("sh Sniff.sh");
                break;
            case 2:
                System.out.print("\n\n\t\t\t SNIFFING ATTACK RUN IT AS
DAEMON");
                break;
            case 3:
                System.out.print("\n\t\t\t\t\t RECORDED ATTACK");
                Runtime.getRuntime().exec("sh catfile.sh
SniffingAttack.log");
                break;
            case 4:
                // goto loop;
                break;
        }
    }
    break;
case 4:
    Runtime.getRuntime().exec("clear");
    Runtime.getRuntime().exec("sh ip.sh");

```

```

        putChar('#');
        putLine(1);
        System.out.print("\n\t\t\tOptions for Spoofing
attack\n\n\t");
        System.out.print("1.Block IP Address\n\t2.Block Loop Back
Address\n\t");
        System.out.print("3.Broadcast Traffic Address\n\t4.Block
Particular");
        System.out.print("service\n\t 5.UnBlocking\n\t 6.Goto Main
Menu");
        putChar('#');
        putLine(1);
        System.out.print("\n\n Please Enter Your Choice : ");
        Scanner scanner35 = new Scanner(CIOUtils.stdin.readLine());
    }
    catch (IOException e) {
        System.err.println("IO Exception:" + e);
    }
    ch5 = scanner35.readString();
    switch (ch5) {
        case 1:
            Runtime.getRuntime().exec("sh BlockparIP.sh");
            break;
        case 2:
            Runtime.getRuntime().exec("sh LoopBTR.sh");
            break;
        case 3:
            Runtime.getRuntime().exec("sh BroadCast.sh");
            break;
        case 4:
            Runtime.getRuntime().exec("sh BlockParser.sh");
            break;
        case 5:
            Runtime.getRuntime().exec("sh UnSpoof.sh");
            break;
        case 6:
            // goto loop;
            break;
    }
    break;

    case 5:
        System.exit(0);
        break;
    default :
        System.out.print("\n\n\n\n\t");
        // goto loop;
    }System.out.println("\n\n\n");
    for (i = null;i < 78;i++) {
        System.out.print("-");
    }System.out.print("\n\n\tIf u want to continue press 1 else
press 0 : ");
    try {
        Scanner scanner36 = new Scanner(CIOUtils.stdin.readLine());
    }catch (IOException e) {
        System.err.println("IO Exception:" + e);
    }

```

```

}choice = scanner36.readString();
if (choice) {
Runtime.getRuntime().exec("clear");
}else {
exit;
}}
}

```

## CODING FOR SCANNER:

```

import java.io.*;
public class Scanner {

    public final static int MAXSERVICES = 100;

    void printChar(char c, int count) {
        int i = 0;
        for (i = 0; i < count; i++) {
            System.out.print(c);
        }
        System.out.println("");
    }

    public static void main(String[] args) {
        DataObjectInputStream fp = null;
        DataObjectOutputStream fp1 = null;
        DataObjectOutputStream fp2 = null;
        String line = new String();
        String[] serviceName = new char[MAXSERVICES][30];
        String[] portData = new char[MAXSERVICES][20];
        String[] runStatus = new char[MAXSERVICES][20];
        String serviceNumbers = new String();
        String allowOrRestrict = new String();
        String hostToBeScanned = new String();
        String cmdStr = new String();

        int totalLines = 0;
        int count = 0;
        int servNumber = 0;
        int number = 0;
        System.out.println("\n\n\n");
        printChar('#', 79);
        System.out.println("Port Scanner Started.....\n");
        System.out.print("Enter the machine Name/IP to scanned:
");
        try {
            hostToBeScanned = CIOUtils.stdin.readLine();
            try {
                fp2 = new DataObjectOutputStream(new
FileOutputStream("ip"));
            }
            catch (FileNotFoundException e) {
                System.err.println("FileNotFoundException: " +
e);
            }
        }
    }
}

```

```

        } // end main;
        fp2.print(hostToBeScanned);
        fp2.close();
    }
    catch (IOException e) {
        System.err.println("IO Exception:" + e);
    }
    cmdStr = "nmap " + hostToBeScanned + " > nmapdata";
    Runtime.getRuntime().exec(cmdStr);
    try {
        fp = new DataObjectInputStream(new
FileInputStream("nmapdata"));
    }
    catch (FileNotFoundException e) {
        System.err.println("FileNotFoundException:" + e);
    }
    try {
        fpl = new DataObjectOutputStream(new
FileOutputStream("serviceDetails"));
    }
    catch (FileNotFoundException e) {
        System.err.println("FileNotFoundException:" + e);
    }
    System.out.println("\nScanned Results
are.....");

    while (line = fp.readLineAndNewline() != 0) {
        totalLines += 1;
        if (totalLines <= 5) {
            continue;
        }
        if (line.equals("\n")) {
            break;
        }
        count += 1;
        Scanner scanner182 = new Scanner(line);
        portData[count] = scanner182.readString();
        runStatus[count] = scanner182.readString();
        serviceName[count] = scanner182.readString();
        System.out.println("\t\t" + count + " " +
(serviceName[count]));
    }
    System.out.println("\nThe above services were running on
'" + hostToBeScanned + "' ..");
    printChar('#', 79);
    unlink("serviceDetails");
    unlink("nmapdata");
    Runtime.getRuntime().exec("sh log.sh");
}
}

```



## CODING FOR SERVICE MANAGER:

```
public class ServiceManager {
    public static void main(String[] args) {
        String[] argv = new String[args.length + 1];
        argv[0] = "ServiceManager";
        for (int i = 1; i < args.length; i++) {
            argv[i] = args[i - 1];
        }
        ServiceManager instance = new ServiceManager();
        System.exit(instance.main(argv.length, argv));
    }

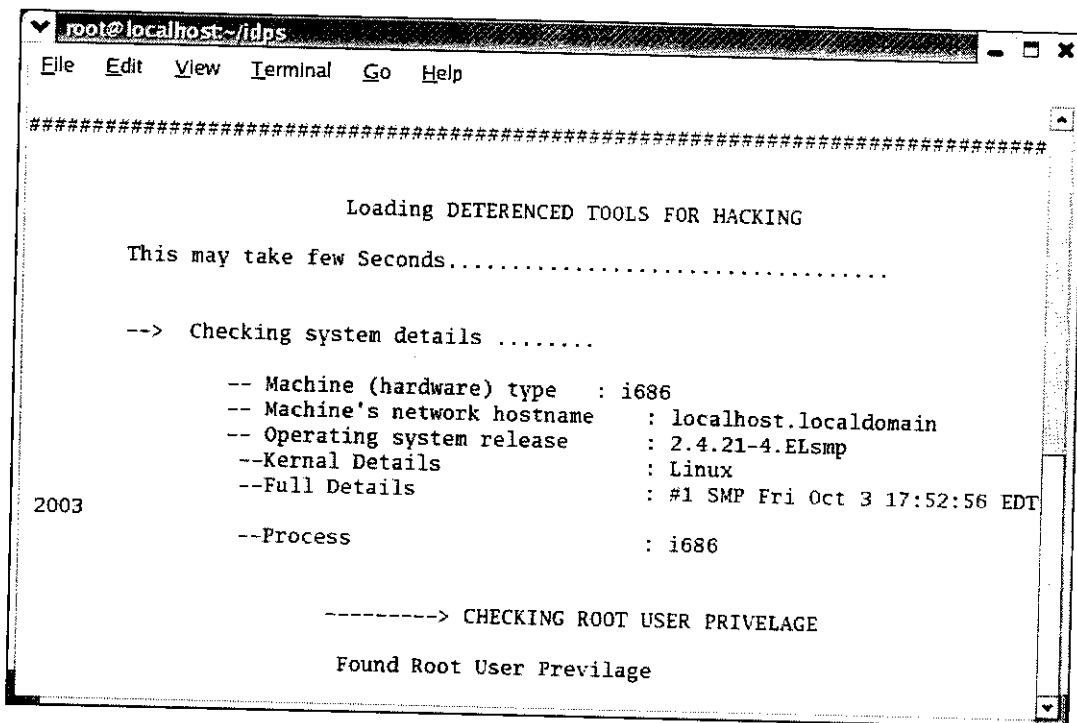
    int main(int argc, String[] argv) {
        String cmdString = null;
        String action = new String();
        int len = 0;
        if (argv[1].equals("start")) {
            action = " start";
        }
        else if (argv[1].equals("stop")) {
            action = " stop";
        }
        else if (argv[1].equals("status")) {
            action = " status";
        }
        } // end while
        else {
            System.out.println("Usage: <" + (argv[0]) + ">.
<start | stop | status> <service1> <service2> ... <serviceN>");
            System.exit(- 1);
        } // end main

        while (argc-- > 2) {
            // LOGIC for starting multiple services
            len = "service ".length() + argv[argc].length() +
action.length();
            cmdString = new String();
            cmdString = "";
            cmdString += "service " + argv[argc] + action;
            System.out.println("command is: '" + cmdString +
            "'");
            Runtime.getRuntime().exec(cmdString);
        }
    }
}
```

APPENDIX 2-SCREEN SHOTS

## 8.2 APPENDIX 2 – SCREEN-SHOT

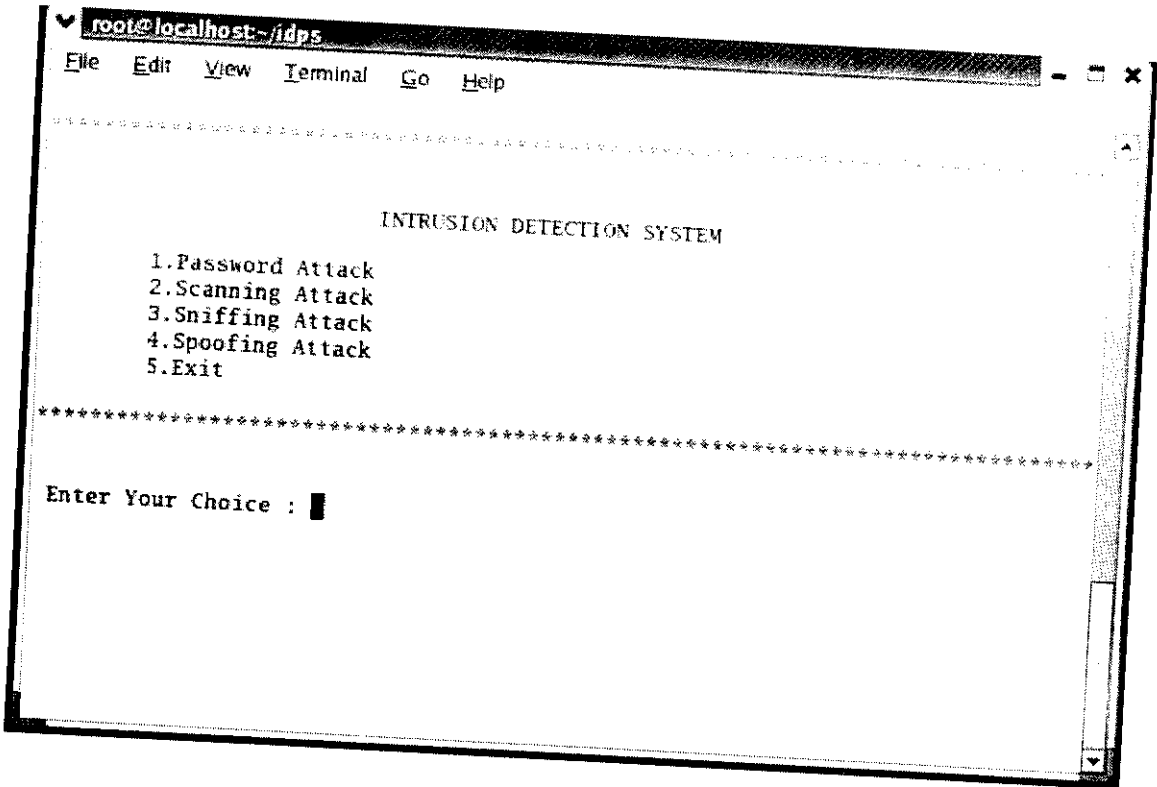
### Welcome screen 'Displaying system details'



The screenshot shows a terminal window titled 'root@localhost:~/idps'. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', 'Go', and 'Help'. The terminal content is as follows:

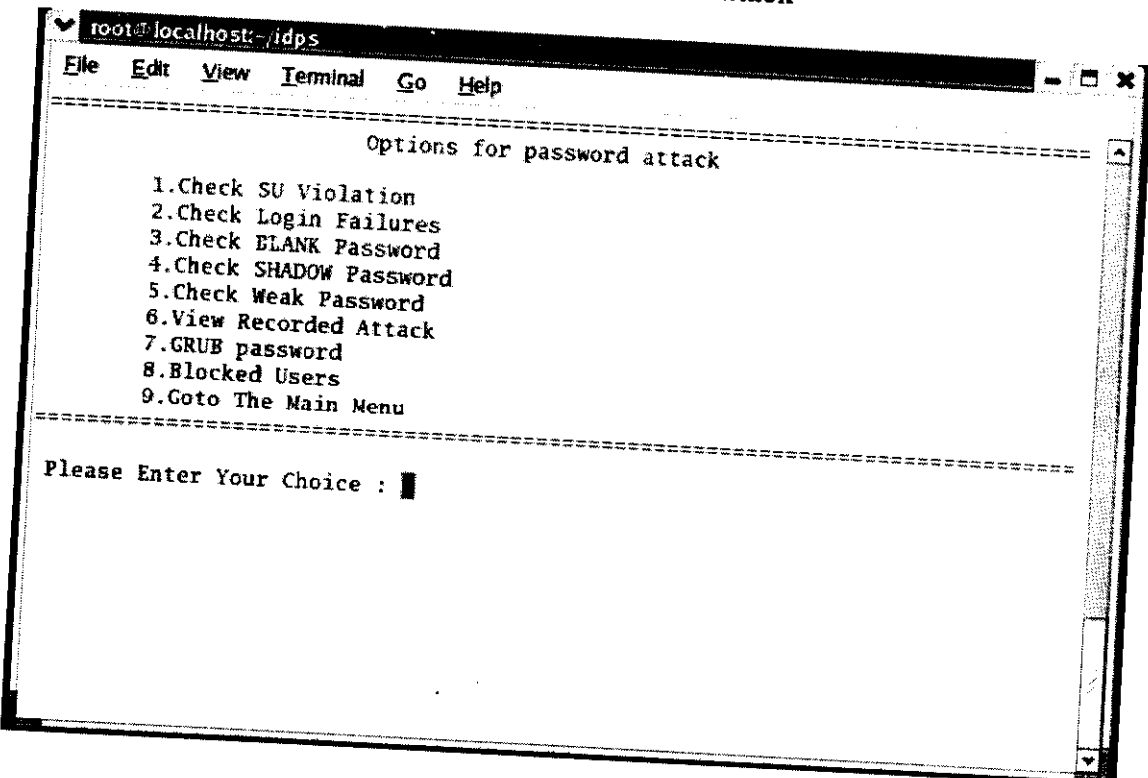
```
#####  
  
Loading DETERENCED TOOLS FOR HACKING  
  
This may take few Seconds.....  
  
--> Checking system details .....  
-- Machine (hardware) type : i686  
-- Machine's network hostname : localhost.localdomain  
-- Operating system release : 2.4.21-4.ELsmp  
--Kernal Details : Linux  
--Full Details : #1 SMP Fri Oct 3 17:52:56 EDT  
2003  
--Process : i686  
  
-----> CHECKING ROOT USER PRIVELAGE  
  
Found Root User Previlage
```

## Main Window



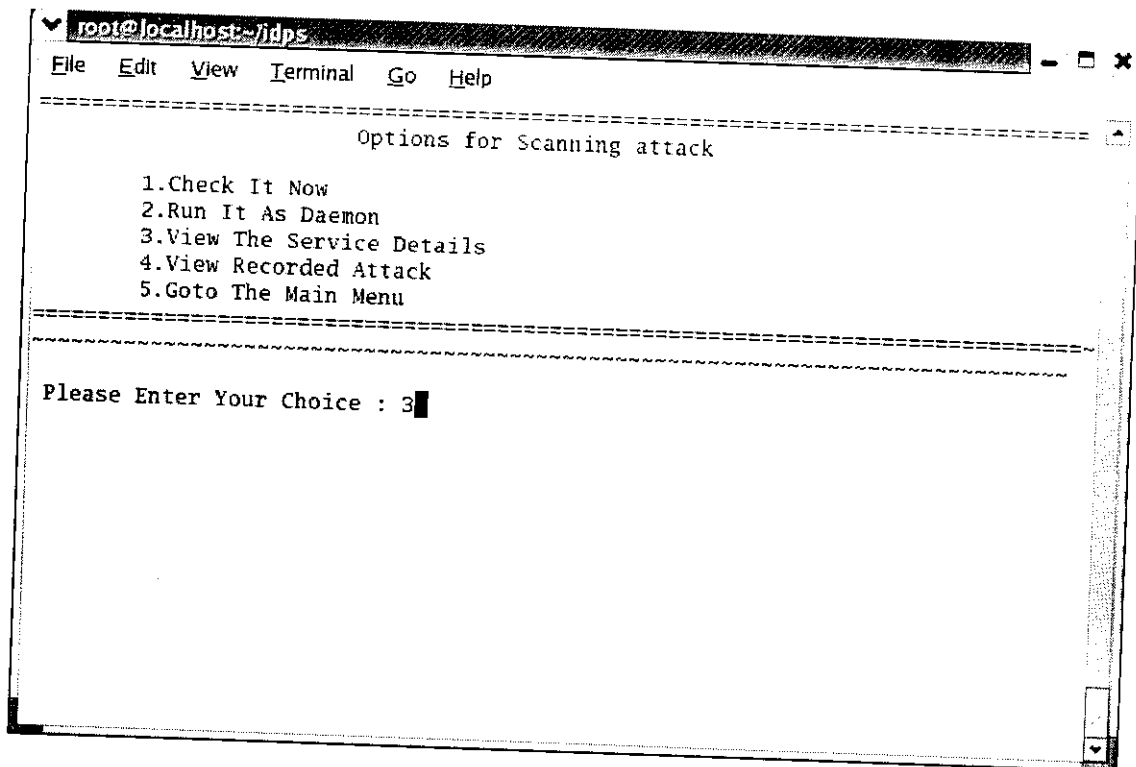
```
root@localhost:~/idps
File Edit View Terminal Go Help
*****
                INTRUSION DETECTION SYSTEM
1.Password Attack
2.Scanning Attack
3.Sniffing Attack
4.Spoofing Attack
5.Exit
*****
Enter Your Choice : █
```

## Options For Password Attack



```
root@localhost:~/idps
File Edit View Terminal Go Help
-----
                Options for password attack
1.Check SU Violation
2.Check Login Failures
3.Check BLANK Password
4.Check SHADOW Password
5.Check Weak Password
6.View Recorded Attack
7.GRUB password
8.Blocked Users
9.Goto The Main Menu
-----
Please Enter Your Choice : █
```

## Options For Scanning Attack

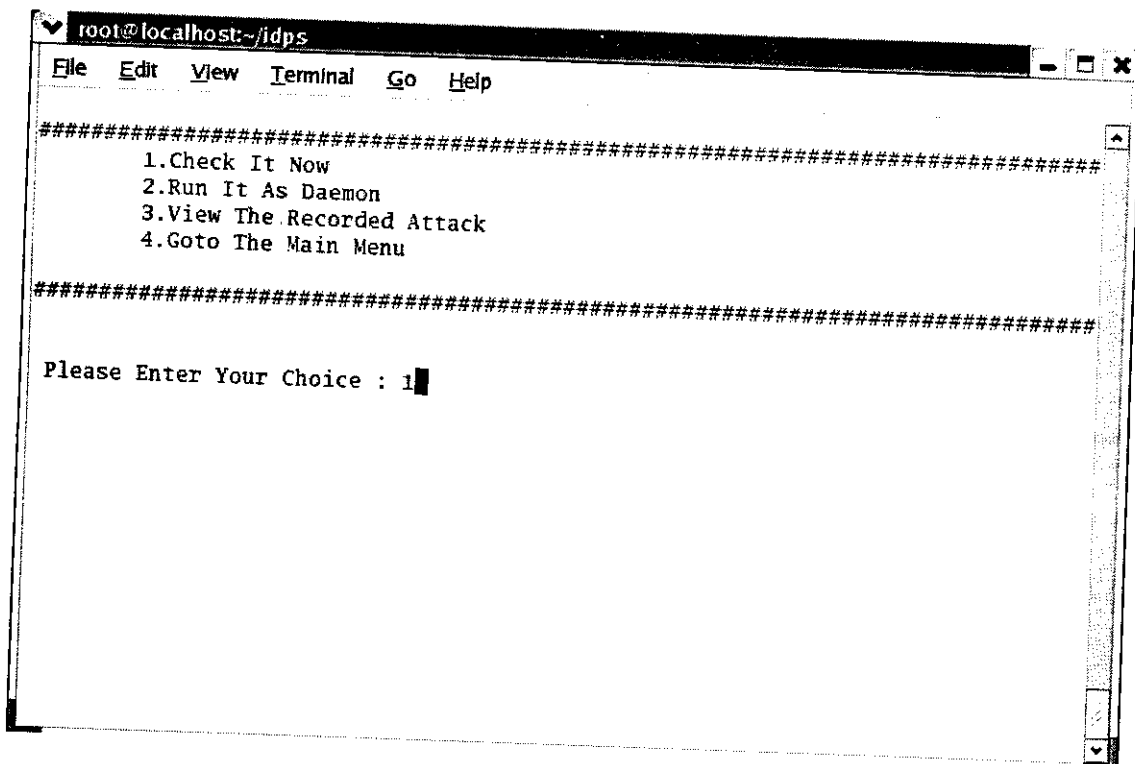


```
root@localhost:~/jdps
File Edit View Terminal Go Help
-----
Options for Scanning attack

1.Check It Now
2.Run It As Daemon
3.View The Service Details
4.View Recorded Attack
5.Goto The Main Menu

-----
Please Enter Your Choice : 3
```

## Options For Sniffing Attack



```
root@localhost:~/jdps
File Edit View Terminal Go Help
#####
1.Check It Now
2.Run It As Daemon
3.View The Recorded Attack
4.Goto The Main Menu
#####
Please Enter Your Choice : 1
```

## Options For Spoofing Attack

```
root@localhost:~/jdps
File Edit View Terminal Go Help
#####
Options for Spoofing attack

1.Block IP Address
2.Block Loop Back Address
3.Broadcast Traffic Address
4.Block Particularservice
5.UnBlocking
6.Goto Main Menu
#####

Please Enter Your Choice : 1
```