A LINUX BASED FIREWALL SYSTEM

BY -

VIKAS AGARWAL-031214
VIBHANSHU SINGH-031418
UTKARSH CHOUDHARY-031213

JAYPEE UNIVERSITY OF INFORMATION & TECHNOLOGY

विद्या तत्व ज्योतिसमः

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

MAY – 2007

Submitted in partial fulfillment of the Degree of Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY-WAKNAGHAT

CERTIFICATE

This is to certify that the work entitled, " A LINUX BASED FIREWALL SYSTEM" submitted by UTKARSH CHOUDHARY(C.S.E.) , VIKAS AGARWAL(C.S.E.) and VIBHANSHU SINGH(I.T.) in partial fulfillment for the award of degree of Bachelor of Technology of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

[illegible] 27.05.07

Mr. Ajay Kumar Singh
Project Supervisor

ACKNOWLEDGEMENT

Many people have contributed to this project in a variety of ways over the past few months. We are sincerely thankful to Mr. Ajay Kumar Singh under whose supervision the whole project was carried out for her regular reviews and suggestions. We also acknowledge the many helpful comments received from other teachers of the different departments and visualization courses and seminars. We are deeply indebted to all those who provided reviews and suggestions for improving the materials and topics covered in our package, and we extend our apologies to anyone we may have failed to mention.

Thank You

Utkarsh

UTKARSH CHOUDHARY (031213)

Vikas

VIKAS AGARWAL (031214)

Vibhanshu

VIBHANSHU SINGH (031418)

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

LAN	–	Local Area Network
WAN	–	Wide Area Network
TCP	–	Transfer Control Protocol
UDP	–	User Datagram Protocol
IP	–	Internet Protocol
VPN	–	Virtual Private Network
HTTP	–	Hyper Text Transfer Protocol
FTP	–	File Transfer Protocol

ABSTRACT

This report focuses on the representation of a simple firewall to monitor and control incoming networking requests on a LINUX system from any remote host.

The currently available techniques for the implementation of a firewall on LINUX systems are iptables , ipchains etc. This one is a simple packet filter firewall which has been implemented using the inetd daemon and tcp wrapper concepts. The advantage of using the inetd daemon is that it runs in the background just like other daemons on a LINUX based system without consuming much memory thus hiding the complex internal details from the user.

The inetd daemon simultaneously monitors as many number of ports as defined by user and invokes separate processes for TCP or UDP protocol whenever there is a request on any of the monitored ports.

As a result, we have been able to implement and demonstrate a basic level firewall for LINUX systems , and report on the complexities involved in building and deploying such a system.

CHAPTER 1

NETWORK SECURITY BASICS

Introduction

- Networks are telecommunication highways over which information travels.
- Networks and their associated information technology resources are exposed to potential points of attack (eg . spoofing , traffic flow analysis, trap doors, trojan horses, viruses, worms, etc).
- Centralized network management authority does not exist so layered security measures are needed to protect data as it traverses the network.

These layered security measures include:

- Firewalls
- Routers
- Intrusion Detection systems
- Other components (VPN, encryption, etc)

Types of Networks

- ***Local Area Networks(LAN)-***

  A discrete network that is designed to operate in a specific limited area like a floor of building.
- ***Wide Area Network(WAN)-***

  A network of sub networks that interconnects LANs over wide geographic areas; usually within a single organization.
- ***Intranet-***

  A TCP/IP based logical network within an organization's internal network.
- ***Extranet-***

  A TCP/IP based network that is accessed by users outside the organization but that is not publicly accessible.

- ***Internet-***

  A global, public TCP/IP network.
- ***Virtual Private Network (VPN)-***

  A network where packets that are internal to a private network pass across a public network; traffic is encrypted, integrity protected, and encapsulated into new packets that are sent across the internet.

Connecting Networks Together

- ***Bridges:*** Operate at the Link layer to forward data to all other connected networks if the destination computer is not on the local network.
- ***Routers:*** Operate at the network layer and direct (or route) packets to the appropriate "next hop" based on their routing tables and the destination computer's IP address.
- ***Switches:*** Operate at the Link layer (or network layer) to deliver data to the specific port where the destination MAC address is located.
- ***Firewalls:*** Devices that sit between networks to contrl and restrict the network traffic that is allowed to flow between those networks. Firewall enforce network security policy.
- ***Modems or Dial-in-Line:*** A device program that allows a computer to transmit data over telephone lines. These connections can be just as dangerous as if u had a T1 or a T3 line.

Network Security Risks

- Denial of service-attacks on the availability of networks or computer systems
- Network packets that violates protocol compliance or that are malformed can cause some systems to crash
- Some network attacks flood a network with more packets than the network can handle.
- Other network attacks create half-open connections to utilize none are left.
- Information theft - Attacks on confidential information (eg. customer private information, credit card information ,etc)

- Network services can be abused by malicious users to logon to hosts and other devices on the network.
- Confidential information may be easily accessible through network services due to misconfigurations, poor access controls, etc.
- Information messages are intercepted while packets are being sent across publicly accessible networks lines.
- Intrusion - Unauthorized access (usually with privileged access rights) to a network or a computer system that could compromise the integrity and or availability of critical systems and data.
- Some networks services allow access to the host without any password required.
- Some network services allow a user to sign-up a cross the network to access the host.
- Some network services use trusted access based on host IP address that can be spoofed
- Reputation - Confidence of customers , business partners etc . is lost . This perhaps is the biggest risk that e business face.

<u>What is network security?</u>

- Network security consists of the technologies and processes that are deployed to protect internal networks from external threats.
- The primary goal of network security is to provide controls at all points along the network perimeter which allow access to the internal network and only let traffic is authorized ,valid and of acceptable risk
- Network security controls cannot completely eliminate risk . The goal is to minimize the risk as much as possible and to avoid unnecessary risks.
- Without network security the risks of connectivity would be very high.

So, how does firewall factor comes into this equation?

- Firewalls are one of the essential technologies that are used at perimeter of the network to protect internal networks from external threats.
- There are other technologies also like intrusion detection system, VPNs, routers and other uses of encryption .But firewall is most important all of them

CHAPTER 2

NETWORKING BASICS

What is a Protocol?

An agreed - upon format for transmitting data between two devices . The protocol determines the following:

- The type of error checking to be used
- Data compression method, if any
- How the sending device will indicate that it has finished sending a message
- How the receiving device will indicate that it has received a message

There are a variety of standard protocols from which programmers can choose. Each has particular advantages and disadvantages; for example, some are simpler than others, some are more reliable, and some are faster.

From a user's point of view , the only interesting aspect about protocols is that your computer or device must support the right ones if you want to communicate with other computers . The protocol can be implemented either in hardware or in software.

User Datagram Protocol (UDP)

User Datagram Protocol (UDP) provides an unreliable , connectionless datagram transport service for IP . Therefore, this protocol is usually used for transaction-oriented utilities such as the IP standard Simple Network Management Protocol (SNMP) and Trivial File Transfer Protocol (TFTP).

Like TCP , which is discussed in the next section , UDP works with IP to transport messages to a destination and provides protocol ports to distinguish between software applications executing on a single host . However,UDP avoids the overhead of reliable data transfer mechanism by not protecting against datagram loss or duplication, unlike TCP.

Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP) provides a reliable, connection-oriented, transport layer service for IP. Due to its high capability of providing interoperability to dissimilar computer systems and networks, TCP/IP has rapidly extended its reach beyond the academic and technical community into the commercial market. Using a handshaking scheme , this protocol provides the mechanism for establishing , maintaining , and terminating logical connections between hosts. Additionally, TCP provides protocol ports to distinguish multiple programs executing on a single device by including the destination and source port number with each message. TCP also provides reliable transmission of byte streams , data flow definitions , data acknowledgments , data retransmission, and multiplexing multiple connections through a single network connection.

IP Addresses

All the IP-based networks (Internet and LANs and WANs) use a consistent, global addressing scheme. Each host, or server, must have a unique IP address. Some of the main characteristics of this address scheme are:

- Addresses cannot be duplicated, so they won't conflict with other networks on the Internet,
- IP addressing allows an unlimite number of hosts or networks to connect to the Internet and other networks,
- IP addresses allow networks using different hardware addressing schemes to become part of dissimilar networks

IPv4

Internet Protocol version 4 is the fourth iteration of the Internet Protocol (IP) and it is the first version of the protocol to be widely deployed. IPv4 is the dominant network layer protocol on the Internet and apart from IPv6 it is the only protocol used on the Internet.

IPv4 is a data-oriented protocol to be used on a packet switched internetwork (e.g., Ethernet). It is a best effort protocol in that it doesn't guarantee delivery. It doesn't make any guarantees on the correctness of the data; it may result in duplicated packets and/or packets out-of-order.

IPv4 uses 32-bit (4-byte) addresses, which limits the address space to 4,294,967,296 possible unique addresses. However, some are reserved for special purposes such as private networks (~18 million addresses) or multicast addresses (~1 million addresses). This reduces the number of addresses that can be allocated as public Internet addresses. As the number of addresses available are consumed, an IPv4 address shortage appears to be inevitable, however Network Address Translation (NAT) has significantly delayed this inevitability.

IPv6

Internet Protocol version 6 (IPv6) is a network layer protocol for packet-switched internetworks. It is designated as the successor of IPv4, the current version of the Internet Protocol, for general use on the Internet.

The main improvement brought by IPv6 is the increase in the number of addresses available for networked devices, allowing, for example, each mobile phone and mobile electronic device to have its own address. IPv4 supports 2^{32} (about 4.3 billion) addresses, which is inadequate for giving even one address to every living person, let alone supporting embedded and portable devices. IPv6, however, supports approximately 5×10^{28} addresses for *each* of the roughly 6.5 billion people alive today. With such a large address space available, IPv6 nodes can have as many universally scoped addresses as they need, and network address translation is not required.

Rules:

IP addresses are composed of four one-byte fields of binary values separated by a decimal point. For example,

1.3.0.2, 192.89.5.2, 142.44.72.8

An IP address must conform to the following rules:

- The address consists of 32 bits divided into four fields of one byte (eight bits) each.
- It has two parts: a network number and a host or machine number.
- All hosts on the same network must have the same network number.
- No two hosts on the same network can have the same host number.
- No two networks can have the same network number if they are connected in any way.

What is a port?

A ``port'' is ``virtual slot'' in your TCP and UDP stack that is used to map a connection between two hosts , and also between the TCP/UDP layer and the actual applications running on the hosts.

They are numbered 0-65535, with the range 0-1023 being marked as ``reserved'' or ``privlileged'', and the rest (1024-65535) as ``dynamic'' or ``unprivileged''.

There are basically two uses for ports:

- ``Listening'' on a port.

  This is used by server applications waiting for users to connect, to get to some ``well known service'', for instance HTTP (TCP port 80), Telnet (TCP port 23), DNS (UDP and sometimes TCP port 53).
- Opening a ``dynamic'' port.

  Both sides of a TCP connection need to be identified by IP addresses and port numbers. Hence, when you want to ``connect'' to a server process, your end of the communications channel also needs a ``port''. This is done by choosing a port above 1024 on your machine that is not currently in use by another communication s channel , and using it as the ``sender'' in the new connection.

Dynamic ports may also be used as ``listening" ports in some applications , most notably FTP.

Ports in the range 0-1023 are almost always server ports. Ports in the range 1024-65535 are usually dynamic ports (i.e., opened dynamically when you connect to a server port). However , *any* port may be used as a server port , and *any* port may be used as an ``outgoing" port.

So, to sum it up, here's what happens in a basic connection:

- At some point in time , a server application on host 1.2.3.4 decides to ``listen" at port 80 (HTTP) for new connections.
- You (5.6.7.8) want to surf to 1.2.3.4, port 80, and your browser issues a connect call to it.
- The connect call, realizing that it doesn't yet have local port number, goes hunting for one. The local port number is necessary since when the replies come back some time in the future, your TCP/IP stack will have to know to what application to pass the reply. It does this by remembering what application uses which local port number
- Your TCP stack finds an unused dynamic port, usually somewhere above 1024. Let's assume that it finds 1029.
- Your firs t packet is then sent, from your local IP, 5.6.7.8, port 1029, to 1.2.3.4, port 80.
- The server responds with a packet from 1.2.3.4 , port 80 ,to you, 5.6.7.8, port 1029.

CHAPTER 3

FIREWALL BASICS

What is a firewall?

A **Firewall** is a piece of hardware and/or software which functions in a networked environment to prevent some communications forbidden by the security policy , analogous to the function of firewalls in building construction.

A firewall has the basic task of controlling traffic between different zones of trust . Typical zones of trust include the Internet (a zone with no trust) and an internal network (a zone with high trust). The ultimate goal is to provide controlled connectivity between zones of differing trust levels through the enforcement of a security policy and connectivity model based on the least privilege principle.

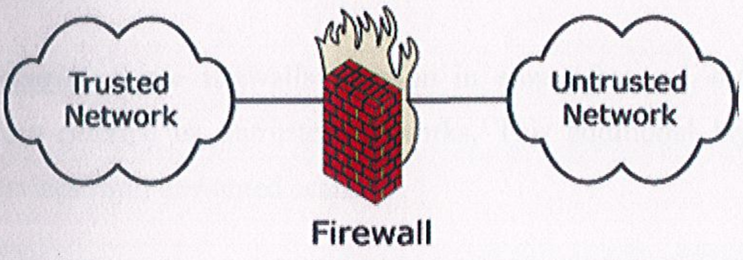


Figure 1.

History of firewalls

Firewall technology first began to emerge in the late 1980s when the Internet was still a fairly new technology in terms of its global usage and connectivity. The original idea was formed in response to a number of major internet security breaches, which occurred in the late 1980s. The Morris Worm was the first large scale attack on Internet security, which the online community neither expected, nor were prepared for . The internet community made it a top priority to combat any future attacks from happening and began to collaborate on new ideas, systems and software to make the internet safe again. The first paper published on firewall technology was in 1988, when Jeff Mogul from Digital Equipment Corp. developed filter systems known as packet filter firewalls. From 1980-1990 two colleagues from AT&T Bell Laboratories, Dave Presetto and Howard Trickey, developed the second generation of firewalls known as circuit level firewalls. In 1994 an Israeli company called Check Point Software Technologies built this in to readily available software known as FireWall-1 . A second generation of proxy firewalls was based on Kernel Proxy technology.

What a firewall can do?

Auditing and logging - Firewalls can provide auditing and logging capabilities . By configuring a firewall to log and audit activity, information may be kept and analyzed at a later date. Firewalls can generate statistics based on the information they collect. These statistics can be useful in making policy decisions that relate to network access and utilization.

Security - Some firewalls function in a way that can hide internal or trusted networks from external or untrusted networks. This additional layer of security can help shield services from unwanted scans.

Traffic bottlenecks - In some networks, firewalls create a traffic bottleneck. By forcing all network traffic to pass through the firewall, there is a greater chance that the network will become congested.

Single point of failure - Firewalls can create a single point of failure . In most configurations where firewalls are the only link between networks , if they are not configured correctly or are unavailable , no traffic will be allowed through.

Increased management responsibilities - A firewall often adds to network management responsibilities and makes network troubleshooting more complex . If network administrators don't take time to respond to each alarm and examine logs on a regular basis , they will never know if the firewall is doing its job. All firewalls require ongoing administrative support, general maintenance, software updates, security patches .

What firewalls cannot do?

A firewall cannot and does not guarantee that your network is 100% secure. To achieve greater protection ,a firewall should be used in conjunction with other security measures. Even then, there is no guarantee that the network will be 100% secure.

Firewalls cannot offer any protection against inside attacks. For a firewall to be effective, all traffic must pass through it. Users on the internal or trusted network often have access to the protected services without having to go through the firewall. A high percentage of security incidents today come from inside the trusted network.

Firewalls cannot protect against unwanted or unauthorized access through back doors on your network . Back doors are typically created when an internal user dials out from an unauthorized modem and establishes a connection to an untrusted network. This behavior can be innocent in that the user doesn't even realize they are opening a back door, but it is just as threatening as shutting down the firewall.

Firewalls can't protect very well against things like viruses or malicious software (malware). There are too many ways of encoding binary files for transfer over networks, and too many different architectures and viruses to try to search for them all. In other words, a firewall cannot replace security-consciousness on the part of your users. In general, a firewall cannot protect against a data-driven attack--attacks in which something is mailed or copied to an internal host where it is then executed. This form of attack has occurred in the past against various versions of *sendmail*, *ghostscript*, scripting mail user agents like *Outlook*, and Web browsers like *Internet Explorer*.

Organizations that are deeply concerned about viruses should implement organization-wide virus control measures. Rather than only trying to screen viruses out at the firewall, make sure that every vulnerable desktop has virus scanning software that is run when the machine is rebooted. Blanketing your network with virus scanning software will protect against viruses that come in via floppy disks, CDs, modems, and the Internet.

Types of firewall techniques:

- ***Packet filter:***

Looks at each packet entering or leaving the network and accepts or rejects it based on user-defined rules. Packet filtering is fairly effective and transparent to users, but it is difficult to configure. In addition, it is susceptible to IP spoofing.

- ***Application gateway:***

Applies security mechanisms to specific applications, such as FTP and Telnet servers. This is very effective, but can impose a performance degradation.

- ***Circuit-level gateway:***

Applies security mechanisms when a TCP or UDP connection is established. Once the connection has been made, packets can flow between the hosts without further checking.

- ***Proxy server:***

Intercepts all messages entering and leaving the network . The proxy server effectively hides the true network addresses.

Network Layer Firewall

In computer networks , a network layer firewall works as a packet filter by deciding what packets will pass the firewall according to rules defined by the administrator. Filtering rules can act on the basis of source and destination address and on ports , in addition to whatever higher level network protocols the packet contain. Network layer tend to operate very fast and transparently to user.

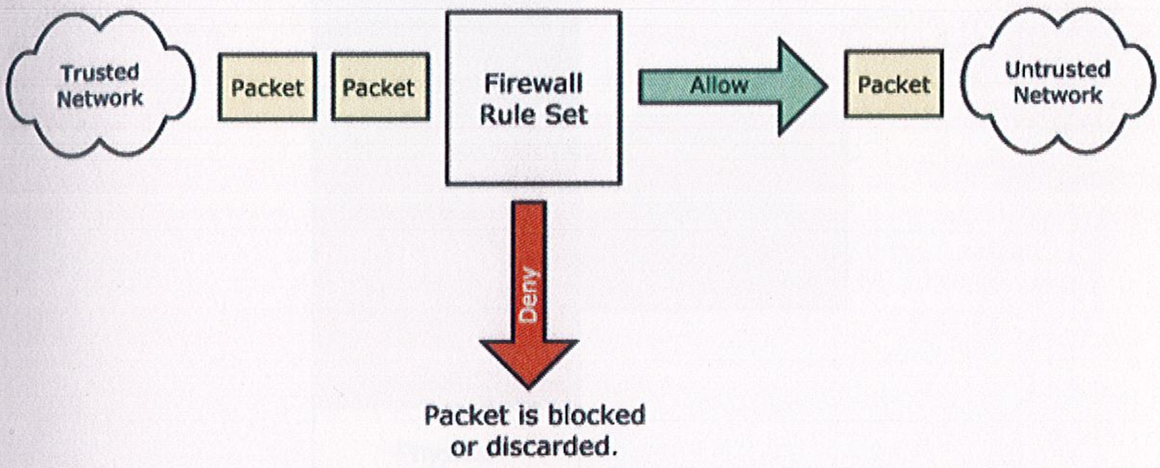
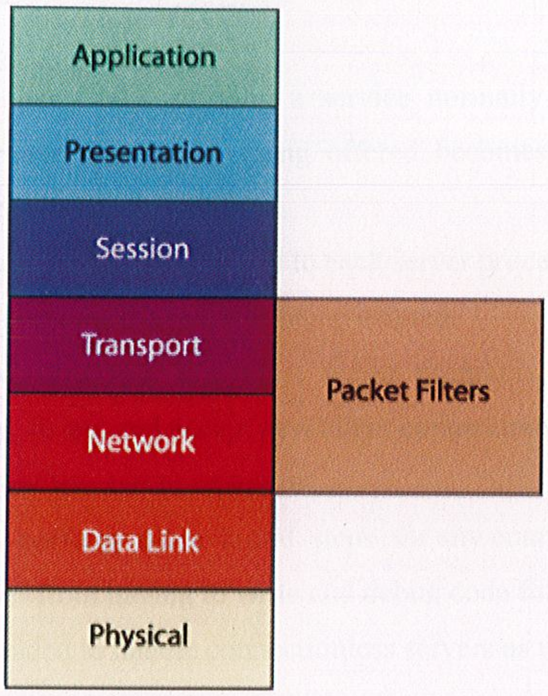


Figure 2 : Packet Filter Firewall

Network layer firewall generally fall into two sub-categories , stateful and non-stateful. Stateful firewalls hold some information on the state of connections (for example : established or not , initiation , handshaking ,data or breaking down connection) as a part of their rules(e.g. only host inside the firewall can establish connection on a certain port).

Stateless firewall has packet filtering capabilities but cannot make more complex decision on what stage communication between the hosts have reached . Stateless firewalls therefore offer less security . Stateless firewalls therefore resembles a router in their ability to filter packets.

A packet filtering firewall is often called a network layer firewall because the filtering is primarily done at the network layer (layer three) or the transport layer (layer four) of the OSI reference model.



<u>Figure 3</u> : Packet Filtering OSI Layers

CHAPTER 4

IMPLEMENTATION OF FIREWALL

Using the inetd daemon

Each server running under UNIX offering a service normally executes as a separate process . When the number of services being offered becomes large , however , this becomes a burden to the system.

This is because resources must be allocate to each server process running , even when there are no current requests for the services being offered.

Additionally , it can be observed that most server programs use the same general procedure to create, bind, listen, and accept new client connections. A similar observation can be made for connectionless server operation.

The inetd daemon can perform these initial steps for any connection-oriented server, saving the server writer from having to write and debug code for these steps. The inetd daemon idea can be extended to handle connectionless servers as well.

Introducing inetd

When the inetd daemon is started for the first time, it must know what Internet services it must listen for and what servers to pass the request off to when a request arrives. This is defined within the startup file /etc/inetd.conf.

The /etc/inetd.conf configuration file

The general file layout of the /etc/inetd.conf file is organized as a text file, with each text line representing one record, which describes one Internet service. Lines starting with # are simply comment lines and are ignored.

The* /etc/inetd.conf *Configuration Record

Field #	*Description*	*Example*
1.	Internet service name	telnet (this might also be a port number)
2.	Socket type	stream or dgram
3.	Protocol	tcp or udp
4.	Flags	nowait or wait
5.	Userid to use	root or nobody
6.	Pathname of executable	/usr/sbin/in.telnetd
7.	Server arguments	in.telnetd

Internet Service Name Field

The Internet service name field within the /etc/inetd.conf record is simply an Internet service name from the /etc/services file.
Alternatively, you can simply supply a port number.

The Socket Type Field

Although the Linux inetd daemon can accept a number of socket types here, only the types stream or dgram will be used in this firewall implementation.
The stream type corresponds to the SOCK_STREAM socket type for the socket function call. The value dgram requests a SOCK_DGRAM socket type.

The Protocol Field

As you might guess, this selects the protocol to be used for the socket. This value must be a valid entry that appears in the /etc/protocols file . Two often used selections are

- tcp for the TCP protocol
- udp for the UDP protocol

Other possibilities also exist, but these are the most commonly used.

The Flags Field

This field is intended for datagram sockets only. Non datagram sockets (such as stream tcp, for example) should specify the value nowait. Datagram-oriented servers come in two types. They are

- Servers that keep reading UDP packets until they timeout and exit (specify wait for these).
- Servers that read one packet and exit (specify nowait for these).

This information is needed by inetd because the handling of dgram traffic is more complex than it is for stream-oriented protocols. This helps the daemon determine how it should handle future dgram connects while the server for that service is running.

The UserID Field

The inetd daemon runs under the root account. This gives it the capability to change its identity to another user account, if it chooses to do so .It is recommended to run servers with the least amount of privilege necessary to carry out their job, for security purposes. Consequently , servers often run under a more limited userID such as nobody , for example.

The Pathname Field

This field simply informs inetd what the full pathname of the executable file is. This is the executable file that is executed by exec after the daemon calls fork.

The Server Arguments Field

All remaining fields on the /inetd.conf configuration line are provided as command-line arguments to the server being invoked with exec. One common source of confusion is that these arguments start with the argument argv[0]. This allows the command name to differ from the pathname. This is useful when one executable exhibits different personalities depending upon its name.

Using the simple elegance of UNIX, the started server is handed the client socket on the following file units (file descriptors):

• File unit 0 has client socket for standard input

• File unit 1 has client socket for standard output

• File unit 2 has client socket for standard error

With this design in place, it is possible that some servers will not require a single socket function call . All of the server I/O can be performed on the normal standard inputs, output, and error file units.

<u>Configuring /etc/inetd.conf to invoke a new server</u>

<u>Establishing the Service</u>

For this test, add one line to the /etc/inetd.conf file

```
9099 stream tcp nowait root /tmp/inetdserv inetdserv
```

Now, let's review what this last line means to inetd:

• Because your new service does not have a name in the /etc/services file , the first field simply contains the port number you want to use. Port 9099 was chosen here.

• The second field contains stream so that TCP stream sockets will be used.

• The third field contains tcp to indicate that we want a TCP stream, as opposed to some other protocol stream.

• The fourth field is specified as nowait, which is what is required for TCP stream entries.

• The fifth field is given as root in this example. Your normal userID could be used here (but be sure that appropriate permission to execute /tmp/inetdserv exists, however).

• The pathname /tmp/inetdserv is given as the sixth field. This is the pathname of the executable that will be executed when a connect arrives on the socket.

• The seventh field is specified as inetdserv in this example. In this particular case, our server program pays no attention to the value of argv[0], and just about any value would do here.

Datagram Servers with inetd

When datagram server ports are established by inetd, a special consideration is added. Let's review the inetd steps used as they apply to UDP servers:

1. The inetd server listens on the UDP port that your UDP server will service requests on.
2. The select call used by inetd indicates that a datagram has arrived on the socket (note that inetd does not read this datagram).
3. The inetd server calls fork and exec to start your UDP server.
4. Your UDP server uses file unit zero (stdin) to read one UDP packet.
5. Exit (terminate)

Understanding wait and nowait

A datagram server that simply processes one datagram and then exits should use the nowait flag word. This tells inetd that it may launch additional server processes when additional datagrams arrive. This is necessary because each process started is going to process only one datagram.

For other datagram servers that attempt to read more datagrams, you should use the wait flag word. This is necessary because the server process that inetd starts is going to process subsequent datagrams until it terminates. The wait flag word tells inetd not to launch any more servers for that port until the wait system call informs inetd (with the help of signal SIGCHLD) that your datagram server has terminated.

Understanding the TCP Wrapper Concept

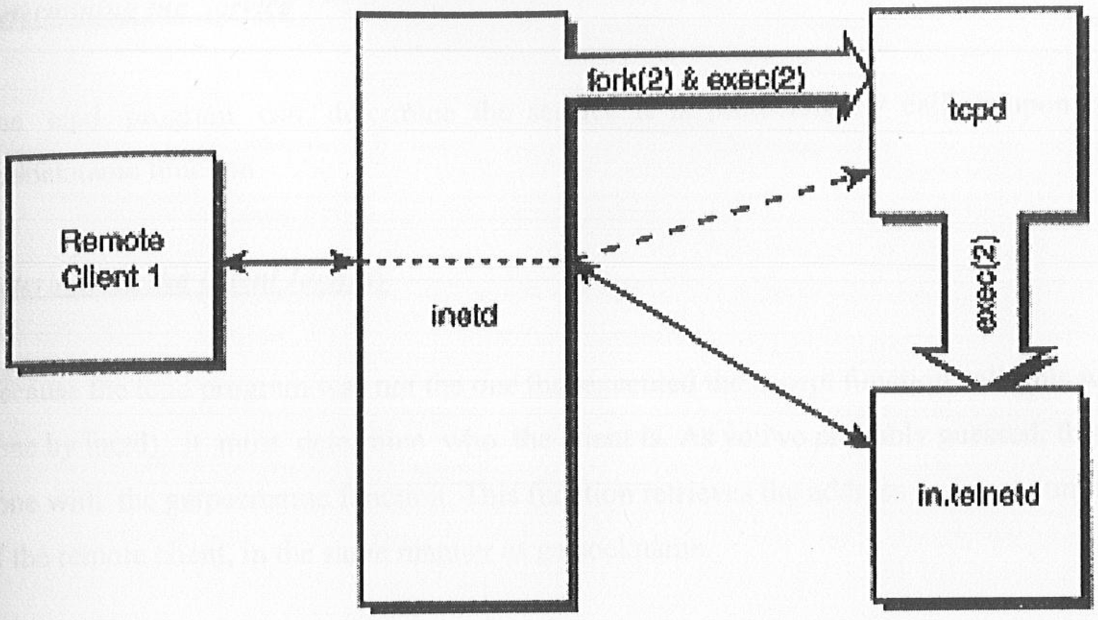


Figure 4

Let's review the process of a remote client connecting to your in.telnetd server:

1. The client uses his telnet client command to issue a connect request to your machine's telnet daemon.

2. Your Linux host is using inetd, which has been configured to listen on port 23 for telnet requests. It accepts the connection request from step 1.

3. The /inetd.conf configuration file directs your inetd server to fork a new process. The parent process goes back to listening for more connects.

4. The child process from step 3 now calls exec to execute the /usr/sbin/tcpd TCP wrapper program.

5. The tcpd program determines whether the client should be given access or not. This is determined by the combination of the socket addresses involved and the configuration files ipban.log

6. If access is to be denied, tcpd simply terminates (this causes file units 0, 1, and 2 to be closed, which are the socket file descriptors).

Determining the Service

The tcpd program can determine the service it is protecting by calling upon the getsockname function.

Determining the Client Identity

Because the tcpd program was not the one that executed the accept function call (this was done by inetd), it must determine who the client is. As you've probably guessed, this is done with the getpeername function. This function retrieves the address and port number of the remote client, in the same manner as getsockname.

Determining the Datagram Client Identity

Determining the identity of a datagram client is a bit trickier. This is because datagrams do not use the accept function call. It is also not possible to use getpeername on datagram sockets because each datagram can potentially come from different clients. The client's address is returned by the recvfrom function call. The client's address and port number can be determined by calling recvfrom using the flag option MSG_PEEK.

```
int z;
struct sockaddr_in adr_clnt;/* AF_INET */
int len_inet; /* length */
int s; /* Socket */
char dgram[512]; /* Recv buffer */
len_inet = sizeof adr_clnt;

z = recvfrom(s, /* Socket */
dgram, /* Receiving buffer */
sizeof dgram, /* Max recv buf size */
MSG_PEEK, /* Flags: Peek at data */
(struct sockaddr *)&adr_clnt,/* Addr */
&len_inet); /* Addr len, in & out */
```

Notice the flag option MSG_PEEK. This option directs the kernel to carry out the recvfrom call as normal except that the datagram is not to be removed from the queue as "read." This allows the tcpd program to "peek" at the datagram that the server will subsequently read, if access is granted.

If the datagram is read through recvfrom using MSG_PEEK flag with a value equal to 0, the datagram packet is simply discarded from the socket queue. This is done as follows :

```
z = recvfrom(s, /* Socket */
dgram, /* Receiving buffer */
sizeof dgram, /* Max recv buf size */
0, /* Flags: Peek at data */
(struct sockaddr *)&adr_clnt,/* Addr */
&len_inet); /* Addr len, in & out */
```

IP Spoofing

A common method of attack, called IP spoofing involves imitating the IP address of a "trusted" host or router in order to gain access to protected information resources. One avenue for a spoofing attack is to exploit a feature in IPv4 known as source routing, which allows the originator of a datagram to specify certain, or even all intermediate routers that the datagram must pass through on its way to the destination address. The Destination router must send reply datagrams back through the same intermediate routers. By carefully constructing the source route, an attacker can imitate any combination of hosts or routers in the network, thus defeating an address-based or domain-name-based authentication scheme.

Therefore, you can say that you have been "spoofed" when someone, by-passing source routing, trespasss it by creating packets with spoofed IP addresses. Yeah, but what is this "IP spoofing" anyway?

Basically, spoofing is a technique actually used to reduce network overhead, especially in wide area networks (WAN). By spoofing you can reduce the amount of bandwidth necessary by having devices, such as bridges and routers, answer for the remote devices. This technique fools (spoofs) the LAN device into thinking the remote LAN is still connected, even though it is not . However, hackers use this same technique as a form of attack on your site.

Another way for you to detect IP spoofing is by comparing the process accounting logs between systems on your internal network. If there has been an IP spoofing, you might be able to see a log entry showing a remote access on the target machine without any corresponding entry for initiating that remote access.

As mentioned before, the best way to prevent and protect your site from IP spoofing is by installing a filtering router that restricts the input to your external interface by not allowing a packet through if it has a source address from your internal network.

<u>Determining the hostname of the client</u>

When the getsockname () function retrieves the ip address of the client that is sending the request, to detect an incident of ip spoofing the hostname of the client needs to be resolved and retrieved. This is done using the function gethostbyaddr() details of which are provided later. This function retrieves all the major information that is needed to determine and detect the spoofing.

CHAPTER 5

STRUCTURES USED IN IMPLEMENTATION

<u>The sockaddr_in Structure</u>

```
#include <netinet/in.h>

struct sockaddr_in {
sa_family_t sin_family; /* Address Family */
uint16_t sin_port; /* Port number */
struct in_addr sin_addr; /* Internet address */
unsigned char sin_zero[8]; /* Pad bytes */
};
struct in_addr {
uint32_ t s_ addr; /* Internet address */
};
```

The above structure can be described as follows:

- The sin_family member occupies the same storage area that sa_family does in the generic socket definition. The value of sin_family is initialized to the value of AF_INET.
- The sin_port member defines the TCP/IP port number for the socket address. This value must be in network byte order.
- The sin_addr member is defined as the structure in_addr, which holds the IP number in network byte order. If you examine the structure in_addr, you will see that it consists of one 32- bit unsigned integer.
- Finally , the remainder of the structure is padded to 16 bytes by the member sin_zero[8] for 8 bytes. This member does not require any initialization and is not used.

The struct hostent Structure

```
struct hostent {
char *h_name; /* official name of host */
char **h_aliases; /* alias list */
int h_addrtype; /* host address type */
int h_length; /* length of address */
char **h_addr_list; /* list of addresses */
};
/* for backward compatibility */
#define h_addr h_addr_list[0]
```

The hostent h_name Member :

The h_name entry within the hostent structure is the official name of the host that your are looking up. It is also known as the *canonical* name of the host. If you provided an alias, or a hostname without the domain name, then this entry will describe the proper name for what you have queried. This entry is useful for displaying or logging your result to a log file.

The hostent h_aliases Member :

The hostent h_aliases member of the returned structure is an array of alias names for the hostname that you have queried. The end of the list is marked by a NULL pointer.

The hostent h_addrtype Member :

The value presently returned in the member h_addrtype is AF_INET. However, as IPv6 becomes fully implemented, the name server will also be capable of returning IPv6 addresses. When this happens, h_addrtype will also return the value AF_INET6 when it is appropriate.

The purpose of the h_addrtype value is to indicate the format of the addresses in the list h_addr_list, which will be described next.

The hostent h length Member :

This value is related to the h_addrtype member. For the current version of the TCP/IP protocol (IPv4), this member always contains the value of 4 , indicating 4-byte IP numbers. However, this value will be 16 when IPv6 is implemented, and IPv6 addresses are returned instead.

The hostent h addr list Member

When performing a name-to-IP-number translation , this member becomes your most important piece of information . When member h_addrtype contains the value of AF_INET, each pointer in this array of pointers points to a 4-byte IP address. The end of the list is marked by a NULL pointer.

The recvfrom function

```
#include <sys/types.h>
#include <sys/socket.h>

z = recvfrom(s, /* Socket */
    dgram, /* Receiving buffer */
    sizeof dgram,/* Max rcv buf size */
    0, /* Flags: no options */
    (struct sockaddr *)&adr, /* Addr */
    &x); /* Addr len, in & out */
```

It is used to receive a datagram packet or to eat a datagram packet.

The `recvfrom` arguments are

1. The socket `s` to receive the datagram from.
2. The buffer pointer `dgram` to start receiving the datagram into.
3. The maximum length (`sizeof dgram`) in bytes of the receiving buffer `dgram`.
4. Option flag bits `flags`.

5. The pointer to the receiving socket address buffer , which will receive the sender's address (pointer argument `from`).

6. The pointer to the maximum length (`x`) in bytes of the receiving socket address buffer `from`. Note that the integer that this pointer points to must be initialized to the maximum size of the receiving address structure `from`, prior to calling the function.

The getsockname() function

If function that you wrote receives a socket as an input argument, then you will not know what the socket address of that socket is. This is because your function did not create the socket; and, unless the socket address is also passed to your function as input, you will not know what the address is. The function getsockname permits your function to obtain it.

The function synopsis for `getsockname` is as follows:

```
#include <sys/socket.h>

int getsockname(int s, struct sockaddr *name, socklen_t
*namelen)
```

This function takes the following three input arguments:

1. The socket s to query for the socket address.
2. The pointer to the receiving buffer (argument `name`).
3. Pointer to the maximum length variable. This variable provides the maximum length in bytes that can be received in the buffer (argument `namelen`). This value is updated with the actual number of bytes written to the receiving buffer.

The getpeername() function

This function is used when your code wants to determine the remote address that your socket is connected to.

```
#include <sys/socket.h>

int getpeername(int s, struct sockaddr *name, socklen_t
*namelen);
```

You can see that the function arguments are identical to the `getsockname` function. For completeness, the arguments are described again as follows:

1. The socket `s` to query for the socket address.
2. The pointer to the receiving buffer (argument `name`).
3. Pointer to the maximum length variable. This variable provides the maximum length in bytes that can be received in the buffer (argument `namelen`). This value is updated with the actual number of bytes written to the receiving buffer. The function returns zero if the operation succeeds. If an error occurs, the value `-1` is returned and the value `errno` will contain the reason for the error.

It is used to retrieve client data from a connected TCP socket.

The gethostbyaddr( ) Function

There are times where you have an Internet address, but you need to report the hostname instead of the IP number. A server might want to log the hostname of the client that has contacted it, instead of the IP number alone. The function synopsis for gethostbyaddr( ) is as follows:

```
#include <sys/socket.h> /* for AF_INET */
struct hostent *gethostbyaddr(
const char *addr, /* Input address */
int len, /* Address length */
int type); /* Address type */
```

The gethostbyaddr function accepts three input arguments. They are:

1. The input address (addr) to be converted into a hostname. For address type AF_INET, this is the pointer to the sin_addr member of the address structure.
2. The length of the input address (len). For type AF_INET, this will be the value 4 (4 bytes). For type AF_INET6, this value will be 16.
3. The type of the input address (type), which is the value AF_INET or AF_INET6.

The inet_ntoa( ) Function

There are times when a socket address represents the address of a user that has connected to your server, or represents the sender of a UDP packet. The job of converting a network sequenced 32-bit value into dottedquad notation is inconvenient. Hence, the inet_ntoa( ) function has been provided. The synopsis of the function is as follows:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

char *inet_ntoa(struct in_addr addr);
```

If a socket address addr exists in your program as a sockaddr_in structure, then following code shows how to use inet_ntoa to perform the conversion. The IP number is converted to a string and reported, using the printf function:

```
struct sockaddr_in addr; /* Socket Address */
printf("IP ADDR: %s\n",inet_ntoa(addr.sin_addr));
```

SOURCE CODES

log.c

```
/* This code is used for logging purposes as and when
required by the system */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <errno.h>

int log_open(const char *pathname);
void log(const char *format,...);
void log_close(void);
void bail(const char *on_what);

static FILE *logf = NULL; /* Log File */

/*
* Open log file for append:
* RETURNS:
* 0 Success
* -1 Failed.
*/

int log_open(const char *pathname)
{
 logf = fopen(pathname,"a");
 return logf ? 0 : -1;
}
const char *format;

/*
* Log information to a file:
*/
 void log(const char *format,...)
{
 va_list ap;

 if ( !logf )
 return;                            /* No log file open */

/* retrieves the pid of the process that is serving the
particular request */
 fprintf(logf,"[PID %ld] ", (long)getpid());

 va_start(ap,format);
 vfprintf(logf,format,ap);
 va_end(ap);
 fflush(logf);
 }

 /*
 * Close the log file:
 */
 void  log_close(void) {

 if ( logf )
 fclose(logf);
 logf = NULL;
 }
```

```
/*
* This function reports the error to
* the log file and calls exit(1).
*/
void bail(const char *on_what) {
if ( logf )
{                              /* Is log open? */
if ( errno )                   /* Error? */
log("%s: ",strerror(errno));
log("%s\n",on_what); /* Log msg */
log_close();
}
exit(1);
}
```

runinetd.c

```
#include<stdio.h>
#include<unistd.h>
int main(int argc,char *argv[],char **envp)
    {
          char
*argv1[]={"/usr/sbin/inetd","/home/knoppix/project1/inetd.c
onf",NULL};
          execve("/usr/sbin/inetd",argv1,envp);
          return 0;
    }
```

inetd.c

```
#include<stdio.h>
#include "log.c"
int main()
{
      FILE *fpinet;


fpinet=fopen("/home/knoppix/project1/inetd.conf","a");

     char port[6];
     char typesocket[10];
     char proto[10];
     char wait1[10];

     char pathexectcp[]="/home/knoppix/project1/tcps1";
     char pathexecudp[]="/home/knoppix/project1/dgrams1";
     char serv_arg[2];
     char serverPath[40];
     printf("\n Proceed to modify the configuration
     file for inet daemon.......\n\n");

     printf("\nEnter the port : ");
     scanf("%s",port);

     printf("\nEnter the type of socket : ");
     scanf("%s",typesocket);

     printf("\nEnter the potocol : ");
     scanf("%s",proto);
```

```
        printf("\nEnter wait :");
        scanf("%s",wait1);

        printf("\nEnter the server argument : \n");
        scanf("%s",serv_arg);


        fprintf(fpinet,"%s",port);
        fprintf(fpinet,"%c",' ');

        fprintf(fpinet,"%s",typesocket);
        fprintf(fpinet,"%c",' ');

        fprintf(fpinet,"%s",proto);
        fprintf(fpinet,"%c",' ');

        fprintf(fpinet,"%s",wait1);
        fprintf(fpinet,"%c",' ');

        fprintf(fpinet,"%s","root");
        fprintf(fpinet,"%c",' ');

        if((strcmp(proto,"tcp"))==0)
            fprintf(fpinet,"%s",pathexectcp);

        else
        fprintf(fpinet,"%s",pathexecudp);
        fprintf(fpinet,"%c",' ');
        fprintf(fpinet,"%s",serv_arg);
        fprintf(fpinet,"%c",'\n');
        return 0;
    }
```

tcp serv blk1.c

```
#include <stdio.h>
#include <sys/socket.h>
#include <resolv.h>
#include <arpa/inet.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include "log.c"
#include <netinet/in.h>
#include <netdb.h>

#define PATHNAME "/home/knoppix/project1/tcp.log"
#define PATHNAME2 "/home/knoppix/project1/alias.log"
#define PATHNAME1 "/home/knoppix/project1/tcpbanip.log"

void checkifbannedhostname(struct hostent *hp , struct
sockaddr_in *client_addr)
{
FILE *fp1;
fp1=fopen(PATHNAME2,"r");
char a[20];
log_open(PATHNAME);
int alen=sizeof *client_addr;
char *str_addr;
while(!feof(fp1))
{
    fscanf(fp1,"%s",a);
    int i=0;

        while(hp->h_aliases[i]!= NULL)
            {
            log("Aliaasss %s",hp->h_aliases[i]);
              if (strcmp(hp->h_aliases[i],a)==0)
              {
              log("An IP spoofin detcted(change of IP
  detected)......dropping request by %s",hp->h_name);
              fclose(fp1);
              log_close();
              }
              i++;
          }
        }
  }
  int main()
  {
   struct hostent *hp;

  char *str_addr;
  char a[20];
  FILE *fp;

  fp=fopen(PATHNAME1,"r");
  log_open(PATHNAME);
  int sd , connfd ,addrlen , z;
  char buffer[1024];
  struct sockaddr_in  client_addr , addr;

int sin_size=sizeof(struct sockaddr_in);
```

```
z=getsockname(0,(struct sockaddr*)&client_addr,&sin_size);
printf("%d",z);

hp=gethostbyaddr((char *)&client_addr.sin_addr,sizeof
client_addr.sin_addr,client_addr.sin_family);

str_addr = inet_ntoa(client_addr.sin_addr);
//printf("connection attempted from
%s\n",inet_ntoa(client_addr.sin_addr));

while(!feof(fp))
{
fscanf(fp,"%s",a);
int i=0;
while(inet_ntoa(*(struct in_addr *)hp->h_addr_list[i]))
     {
           if(strcmp(inet_ntoa(*(struct in_addr *)hp-
>h_addr_list[i]),a)==0)
             {

          log("connection closed as this ip %s is
banned",inet_ntoa(client_addr.sin_addr));
          log("client addresses %s",inet_ntoa(*(struct
in_addr *)hp->h_addr_list[0]));
          log("Hostname of client is %s",hp->h_name);

          fclose(fp);
          log_close();
          //close(connfd);
          //break;
          return 0;
```

```
                }
            i++;
            if(hp->h_addr_list[i]==NULL)
                break;
            }
            }

    log("Hostname of client is %s",hp->h_name);

    checkifbannedhostname(hp,&client_addr);
            return 0;

    }
```

dgram_serv.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "log.c"
#define PATHNAME "/home/knoppix/project1/udp.log"
#define PATHNAME1 "/home/knoppix/project1/tcpbanip.log"

 int main(int argc,char **argv,char **envp ) {
 int z;
 char a[20];
 char *srvr_addr = NULL;
 struct sockaddr_in adr_inet;/* AF_INET */
 struct sockaddr_in adr_clnt;/* AF_INET */
 int len_inet; /* length */
 int s; /* Socket */
 char dgram[512]; /* Recv buffer */

 log_open(PATHNAME);
 FILE *fp;
 fp=fopen(PATHNAME1,"r");
,
 /*
 * Create a UDP socket to use:
```

```
 len_inet = sizeof adr_inet;
/*
 * Now wait for requests:
 */

/*
 * Block until the program receives a
 * datagram at our address and port:
 */
 len_inet = sizeof adr_clnt;
 z = recvfrom(0, /* Socket */
 dgram, /* Receiving buffer */
 sizeof dgram, /* Max recv buf size */
 MSG_PEEK, /*gs: no options */
 (struct sockaddr *)&adr_clnt,/* Addr */
 &len_inet); /* Addr len, in & out */
 if ( z < 0 )
 printf("not received");

else{
while(!feof(fp))
{

fscanf(fp,"%s",a);
int i=0;

if(strcmp(inet_ntoa(adr_clnt.sin_addr),a)==0)
{

log("Connection attempted and denied as ip %s is
banned",inet_ntoa(adr_clnt.sin_addr));
log("Dropping Packet.......");
/*
 * We must read this packet now without
* the MSG_PEEK option to discard dgram:
 */
 z = recvfrom(0, /* Socket */
 dgram, /* Receiving buffer */
 sizeof dgram, /* Max rcv size */
 0, /* No flags!! */
 (struct sockaddr *)&adr_clnt,
 &len_inet);

 if ( z < 0 )
 bail("recvfrom(2), eating dgram");
 exit(1);
 fclose(fp);
 log_close();

    }
 else
     {
     execve("/home/knoppix/project1/dgram_pri",argv,envp);
     }
   }
 }
 return 0;
 }
```

dgram_pri.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
# include "log.c"

#define PATHNAME "/home/knoppix/project1/udp.log"

 int main(int argc,char **argv) {
 int z;
 char a[20];
 char *srvr_addr = NULL;
 struct sockaddr_in adr_inet;/* AF_INET */
 struct sockaddr_in adr_clnt;/* AF_INET */
 int len_inet; /* length */
 int s; /* Socket */
 char dgram[512]; /* Recv buffer */

 log_open(PATHNAME);

 len_inet = sizeof adr_inet;

/*
 * Now wait for requests:
```

```
* Block until the program receives a
* datagram at our address and port:
*/
len_inet = sizeof adr_clnt;
z = recvfrom(0, /* Socket */
dgram, /* Receiving buffer */
sizeof dgram, /* Max recv buf size */
MSG_PEEK, /*gs: no options */
(struct sockaddr *)&adr_clnt,/* Addr */
&len_inet); /* Addr len, in & out */
if ( z < 0 )
printf("not received");


else
log("Message received was : %s", dgram);
return 0;
}
```

Log Files :

The log files used in this project include basically 4 log files :-

1. ***tcpbanip.log***

   This log file consists of a list of ip addresses that lie amongst the not trusted ip addresses according to the rules laid down by the firewall administrator. All the ip addresses are checked against the rules in the codes for tcp and udp requests separately.

2. ***alias.log***

   This log file contains the list of aliases or hostnames for a particular ip address that is being tracked with reference to the banned ip list from the tcpbanip.log file.

3. ***tcp.log***

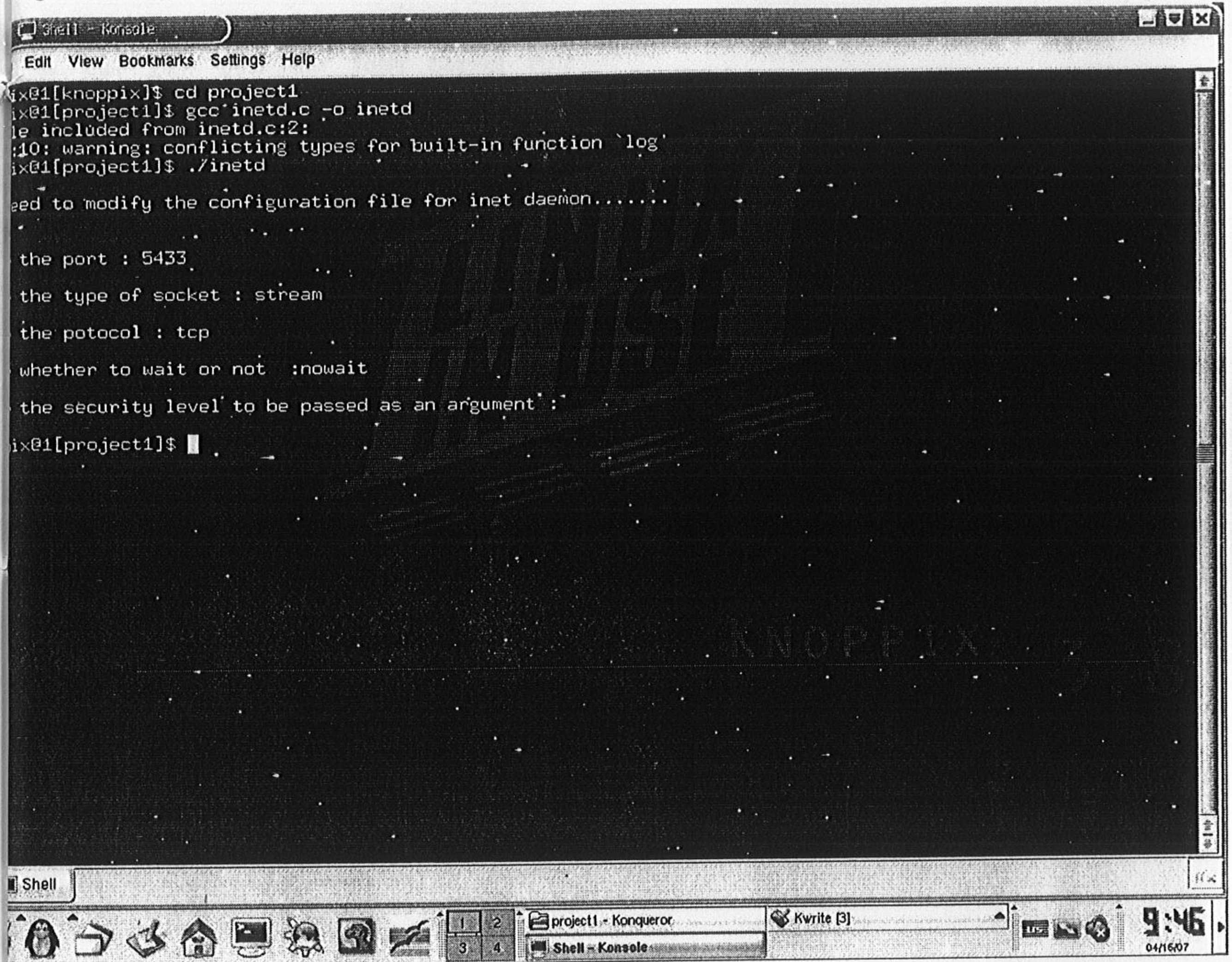
   This log file is used to log all the tcp request attempts made on the monitored ports by any client. Any attempt to get through the port by a client is logged in this file as the network ip address making the request and as to what particular action was taken regarding that request.

4. ***udp.log***

   This file is used to log all the udp requests made on the monitored ports by any client. Any attempt to get through the port by a client is logged in this file as the network ip address making the request and as to what particular action was taken regarding that request.

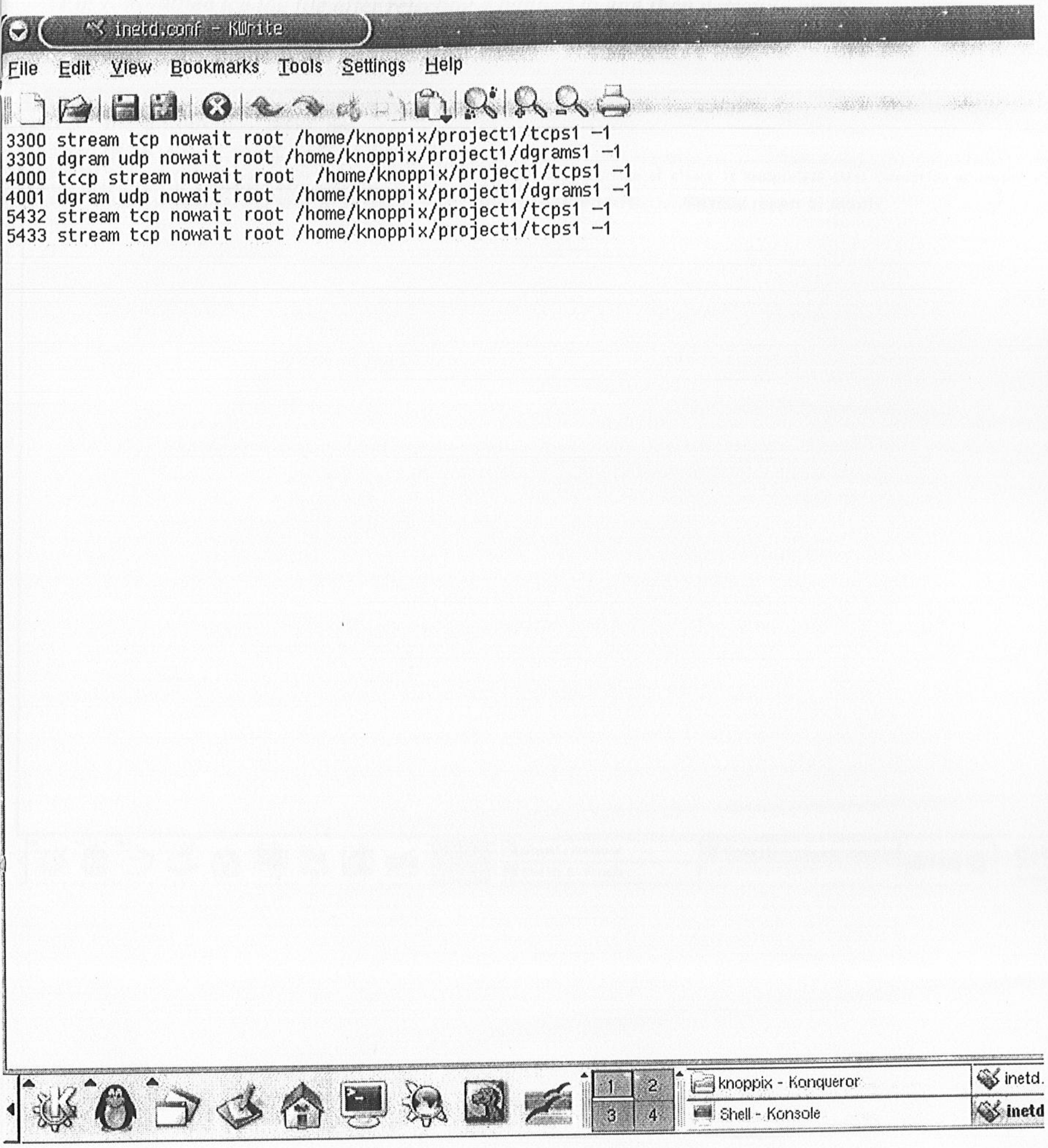
Snapshots :

Fig 5. Modify configuration file inetd.conf



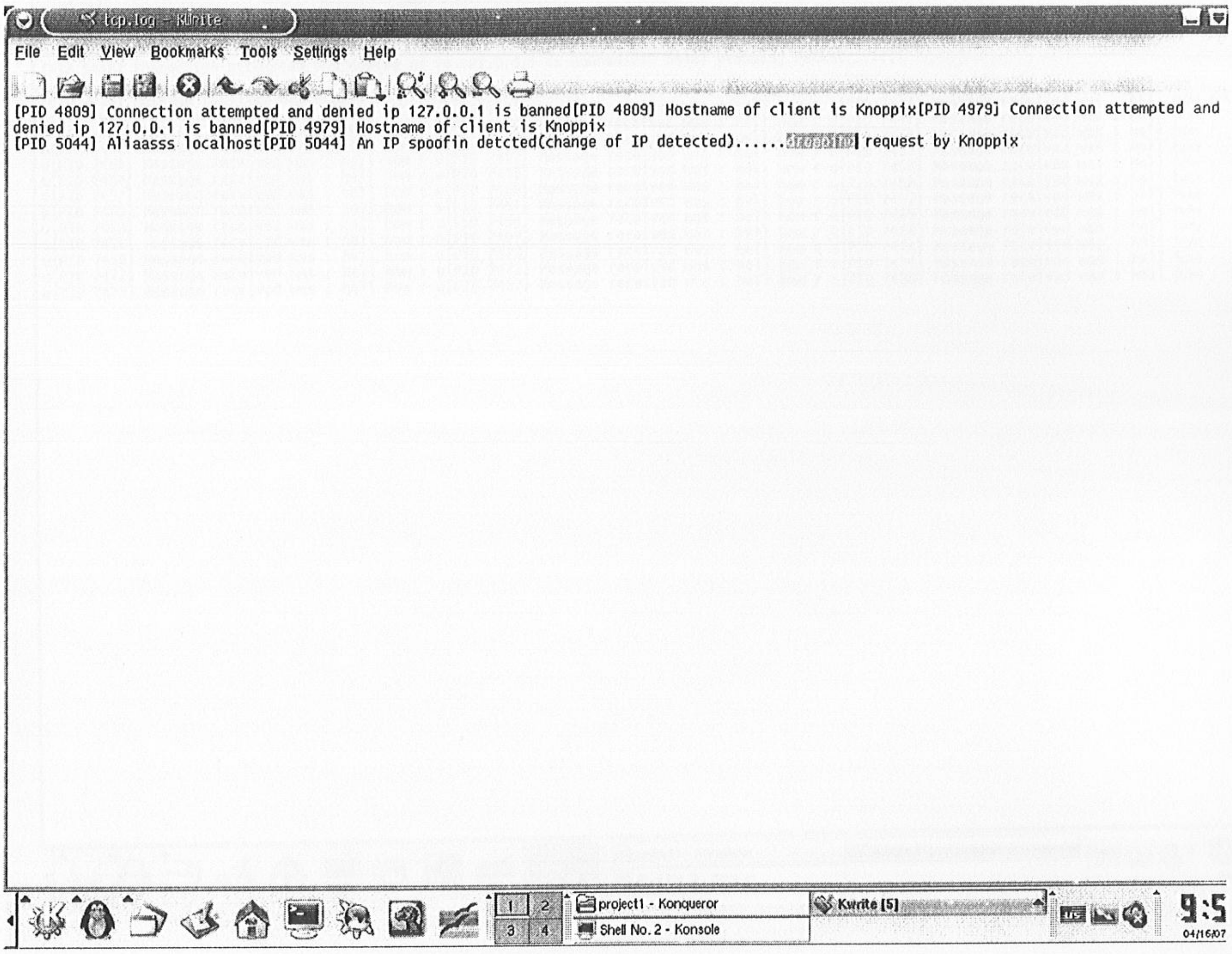
```
ix@1[knoppix]$ cd project1
ix@1[project1]$ gcc inetd.c -o inetd
le included from inetd.c:2:
:10: warning: conflicting types for built-in function `log'
ix@1[project1]$ ./inetd

eed to modify the configuration file for inet daemon.......

 the port : 5433

 the type of socket : stream

 the potocol : tcp

 whether to wait or not  :nowait

 the security level to be passed as an argument :

ix@1[project1]$ 
```

Fig 6. Modified inetd.conf file :



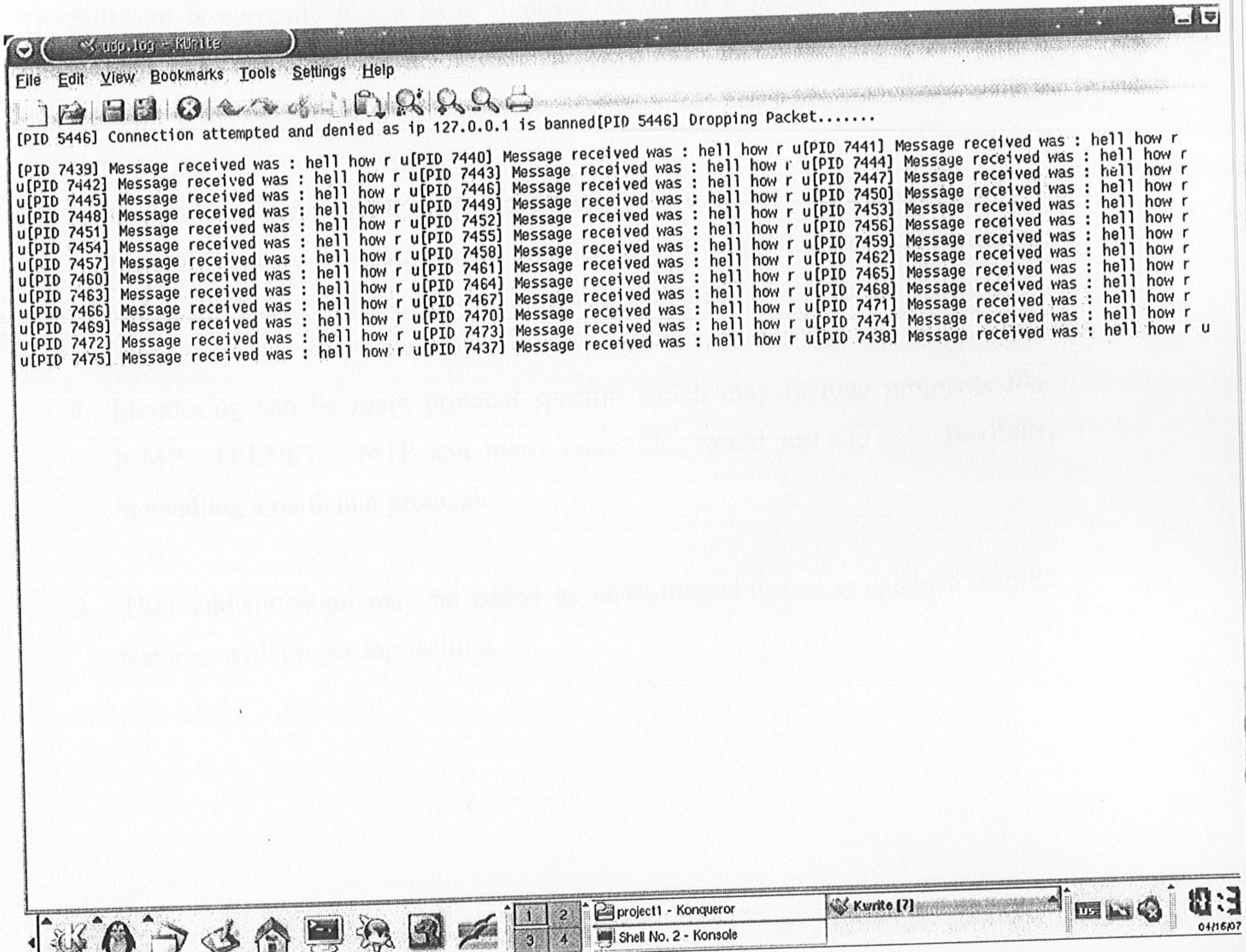
```
3300 stream tcp nowait root /home/knoppix/project1/tcps1 -1
3300 dgram udp nowait root /home/knoppix/project1/dgrams1 -1
4000 tccp stream nowait root  /home/knoppix/project1/tcps1 -1
4001 dgram udp nowait root  /home/knoppix/project1/dgrams1 -1
5432 stream tcp nowait root /home/knoppix/project1/tcps1 -1
5433 stream tcp nowait root /home/knoppix/project1/tcps1 -1
```

Fig 7. Modified tcp.log file after rejecting a banned ip and then detecting an ip spoof instance :



```
[PID 4809] Connection attempted and denied ip 127.0.0.1 is banned[PID 4809] Hostname of client is Knoppix[PID 4979] Connection attempted and
denied ip 127.0.0.1 is banned[PID 4979] Hostname of client is Knoppix
[PID 5044] Aliaasss localhost[PID 5044] An IP spoofin detcted(change of IP detected)......Dropping request by Knoppix
```

Fig 8. Modified udp.log file after denying request to a banned ip and then receiving datagram from a trusted ip address :



```
[PID 5446] Connection attempted and denied as ip 127.0.0.1 is banned[PID 5446] Dropping Packet.......

[PID 7439] Message received was : hell how r u[PID 7440] Message received was : hell how r u[PID 7441] Message received was : hell how r
u[PID 7442] Message received was : hell how r u[PID 7443] Message received was : hell how r u[PID 7444] Message received was : hell how r
u[PID 7445] Message received was : hell how r u[PID 7446] Message received was : hell how r u[PID 7447] Message received was : hell how r
u[PID 7448] Message received was : hell how r u[PID 7449] Message received was : hell how r u[PID 7450] Message received was : hell how r
u[PID 7451] Message received was : hell how r u[PID 7452] Message received was : hell how r u[PID 7453] Message received was : hell how r
u[PID 7454] Message received was : hell how r u[PID 7455] Message received was : hell how r u[PID 7456] Message received was : hell how r
u[PID 7457] Message received was : hell how r u[PID 7458] Message received was : hell how r u[PID 7459] Message received was : hell how r
u[PID 7460] Message received was : hell how r u[PID 7461] Message received was : hell how r u[PID 7462] Message received was : hell how r
u[PID 7463] Message received was : hell how r u[PID 7464] Message received was : hell how r u[PID 7465] Message received was : hell how r
u[PID 7466] Message received was : hell how r u[PID 7467] Message received was : hell how r u[PID 7468] Message received was : hell how r
u[PID 7469] Message received was : hell how r u[PID 7470] Message received was : hell how r u[PID 7471] Message received was : hell how r
u[PID 7472] Message received was : hell how r u[PID 7473] Message received was : hell how r u[PID 7474] Message received was : hell how r u
u[PID 7475] Message received was : hell how r u[PID 7437] Message received was : hell how r u[PID 7438] Message received was : hell how r u
```

CONCLUSION

The Software is currently just a basic implementation of a packet filter firewall for LINUX systems, and thus covers only simple safety basics in this regard. The scope of this concept may be extended in the future, and some key areas of which may include:

1. Design of a much user friendly graphical interface to ease user's interaction with the tool. This needs to be done in compliance with LINUX systems in particular, which still has a few available techniques for doing so.

2. Monitoring can be more protocol specific which may include protocols like ICMP , TELNET , SMTP and many more. This would just add more flexibility in handling a particular protocol.

3. User authentication may be added as an additional feature to enhance security features, with proper log facilites.

BIBLIOGRAPHY

Web Pages

1. http://retran.com/beej/index.html
2. Linux Firewall-Linux Security-Linux Forums
3. http://www.unet.univie.ac.at/aix/cmds/aixcmds3/inetd.htm
4. www.wikipedia.com

Books

1. QUE – Linux Socket Programming By Example BY Warren W Gay
2. Unix Network Programming By W. Richard Stevens
3. Firewalls Complete

Research Papers

1. Firewall Basics By Manu Arian
2. Beej's Guide to Network Programming Using Internet Sockets By Brian "Beej" Hall