FILTERING PROXY SERVER FOR WORLD WIDE WEB

A THESIS

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF

Master Of Engineering
(Computer Science And Engineering)

TO

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY (DEEMED UNIVERSITY)
PATIALA - 147 001

SUPERVISOR
Mr. TARUN MARHWAL
Lecturer, Department of Computer Science and Engineering

SUBMITTED BY
Savita Gupta
ME(CSE) - 137/96(R)/ 1

DEPARTMENT OF COMPUTER SCIENCE
TECHNICAL TEACHERS' TRAINING INSTITUTE
SECTOR 26, CHANDIGARH - 160 019.
1998

CERTIFICATE

Certified that this Thesis report entitled **"Filtering proxy server For World Wide Web"** submitted by **Savita Gupta** in the partial fulfillment of the requirement, for the award of **Master of Engineering (Computer Science and Engineering)** degree of **Thapar Institute of Engineering and Technology, Patiala** is a record of student's own study carried under my supervision and guidance.

This thesis has not been submitted to any other university or institution for the award of any degree.

SUPERVISOR

Tarun

Mr. Tarun Marhwal
Lecturer,
Department of Computer Science
T.T.T.I., Chandgarh

Dr. S. K. Bhattacharya 10.2.98
Prof. & Head,
Department of Computer Science
T.T.T.I., Chandgarh

ACKNOWLEDGMENTS

I am highly grateful to Dr. S. K. Bhattacharya, Principal, Technical Teachers' Training Institute, Chandigarh for providing an opportunity to carry out the present thesis work and for permitting to use the computer laboratory and Internet facility till midnight.

I would like to express a deep sense of gratitude to my thesis guide Mr. Tarun Marhwal, Lecturer, Department of Computer Science and Engineering, TTTI, Chandigarh, for providing help time to time, without his wise counsel and able guidance, it would have been impossible to complete this thesis work.

I am also thankful to other faculty members of Computer Science and Engineering Department, TTTI, Chandigarh for their intellectual support throughout my stay at the institute.

Finally, I am thankful to all my friends for providing constant encouragement and support during the entire work.

Last but not least, I feel myself privileged to express my heartfelt gratitude to my parents, nears and dears for their constant encouragement, support and blessings.

Savita Gupta

Abstract

With the proliferation of ***Internet***, various problems like ***bandwidth*** bottleneck, shortage of ***IP*** addresses, unauthorised access, unsupervised access etc. have crept in. This work aims at development of a ***Filtering Proxy Server*** to address above mentioned problems. It involves the study of various features like killing ***GIF/JPEG*** images, removal of ***Cookie*** headers etc. In addition to that, various security aspects

TABLE OF CONTENTS

Chapter 1 ..... *3*

INTRODUCTION **3**

1.1 Problem definition and Objectives 4

Chapter 2 ..... *6*

FIREWALL **6**

2.1 Benefits of an Internet Firewall 8

2.2 Limitations of an Internet Firewall 9

2.3 Stance of the Firewall 10

2.4 Components of the Firewall System 11

2.4.1 Packet-Filtering Routers 11

2.4.1.1 Service-Dependent Filtering 12

2.4.1.3 Benefits of Packet-Filtering Routers 14

2.4.1.4 Limitations of Packet-Filtering Routers 14

2.4.2 Application-Level Gateways 15

2.4.2.1 Benefits of Application-Level Gateways 19

2.4.2.2 Limitations of Application-Level Gateways 19

2.4.3 Circuit-Level Gateways 19

Chapter 3 ..... *22*

DESIGN AND IMPLEMENTATION **22**

3.1 HTTP Specifications 22

3.1.1 Overall Operations 22

3.1.2 Protocol Parameters 23

3.1.3 HTTP Message 24

3.1.3.1 Request Message 25

3.1.3.2 Response Message 30

3.1.3.3 Entity 33

3.1.4 Security Considerations 34

3.2 Design Aspects 35

3.2.1 Service filtering 35

3.2.2 IP Address filtering 35

3.2.3 Content Filtering 36

3.2.4 IP Address Translation 36

3.2.5 Request Processing 36

3.2.6 Report Generation 37

3.3 Implementation 37

3.3.1 WinSock API 37

3.3.1.1 Introduction 37

3.3.1.2 Client - Server Model 38

3.3.1.3 Operation modes 40

3.3.1.4 Network Application Sketch 41

3.3.2 Flowchart 42

3.3.3 Algorithm 43

Chapter 4..... *53*

CONCLUSION & FUTURE SCOPE.....53

Glossary *55*

References..... *65*

Chapter 1

INTRODUCTION

Internet is a new world which has emerged in present high technological scenario and has gained much popularity in the entire world. Basically, ***Internet is a network of networks based on TCP/IP*** . A ***Network*** is a collection of computers that communicate with one another over a shared medium may be a coaxial cable, UTP cable or some other type of transmission medium. A network allows to share the common resources among all the users in an organisation. When different networks are interconnected to form a big network, it is called ***inter-network***. Similarly, Internet is an interconnection of many small networks like SuraNet, PrepNet, ARCNet etc. With the advent of Internet, the information exchange domain has been increased so tremendously that people call it ***information superhighway***. The Internet offers a variety of services like E-mail, FTP (File Transfer Protocol) , Telnet, Gopher, WWW (World Wide Web) etc., but the WWW is one of the most popular service available on today's Internet [b1]. It accounts for most of the Internet traffic.

Web provides lot of information regarding new trends of market, new development technologies, research papers, latest product information, company advertisement , courses offered by different universities and their services etc. It is implemented over ***HTTP*** (Hyper Text Transfer Protocol) which enables Web clients (browsers) or proxies to fetch pages from the Web server and to return data to the server. To take the advantage of this information superhighway and to enjoy the environment of Web, everybody wants to be on Internet. Many organisations are turning to the Web to increase their presence and visibility in today's marketplace [b1].

While Internet connectivity offers enormous benefits in terms of increased access to information. At the same time, it is sometimes very risky with low levels of security.

The Internet suffers from security problems that could have disastrous results for unprepared sites. Some of the problems with Internet security are the result of inherent vulnerabilities in the services [b8], while others are a result of host configuration and access controls that are poorly implemented. Traffic leaving a sub-network travels through unknown facilities that are under the control of the other organisations involved. Therefore, the end-systems also called ***firewall systems*** (***Bastion Hosts***) [b7] must take the responsibility for the security of Internet communication.

A ***firewall*** can be designed in many ways and may be developed with a wide array of hardware (***Packet Filtering Routers***) and software components. These software components are called ***application proxies*** and are key components for high security *firewalls*. ***Application proxies*** stand on the public side of a network connection and act as custom agents for the internal applications, accepting and authorising all appropriate incoming traffic for those applications prior to relaying it to them for processing. An ***application proxy*** is specially designed to be fortified against attacks from the public network. To the internal user, the proxy appears to be the external network; to the external network, it appears to be the user. Thus, it seamlessly conceals each network and its users from the other. When proxies are used in a *firewall*, no communications are allowed at a system level. All traffic between the networks is required to pass through the application proxies [b7].

1.1 Problem definition and Objectives

In this era of Internet, everybody is worried about connecting their organisation to the Internet, but there are some problems associated after it gets connected to the Internet. For example, the problems which an educational institution comes across are:

1. Traffic Congestion on lines [w4].
2. Shortage of **IP** addresses [w2].
3. Students accessing some pornographic material [b1].
4. Outsiders accessing some material, which otherwise not intended to be used by them [b8].

This thesis titled ***"Filtering Proxy Server For World Wide Web"*** tries to address the solution to some of these problems and develop a re-configurable software. It implements a ***HTTP*** (Hyper Text Transfer Protocol) proxy which is configurable. By configuring the filtering rules of the proxy, any organisation can implement the security policy to protect its internal trusted network from the external network i.e ***Internet***. Moreover, for the past few years the Internet has been experiencing an address space crisis that has made registered IP addresses a less plentiful resource. This means that organisations wanting to connect to the Internet may not be able to obtain enough registered IP addresses to meet the demands of their user population. A Filtering Proxy Server tries to alleviate the address space shortage and eliminate the need to renumber when an organisation changes *Internet Service Providers* (ISPs). Further a ***Proxy server*** provides a perfect point to audit or log Internet usage [w8]. This permits the network administrator to justify the expense of the Internet connection to management, pinpoint potential bandwidth bottlenecks. It also includes the study of various aspects of building a *firewall* such as service filtering [b10], killing the GIF animation, restricting access to certain Web sites addresses, changing user agent header, keeping log of the all accessed URL's (Universal Resource Locator) etc.

Chapter 2

FIREWALL

Security is one of the primary concerns when an organisation connects its private network to the Internet. Regardless of the business, an increasing number of users on private networks are demanding access to Internet services such as the ***World Wide Web (WWW), Internet mail, Telnet,*** *and File Transfer Protocol (FTP)* [b7]. In addition, corporations want to offer ***WWW*** home pages and ***FTP*** servers for public access on the Internet. Network administrators have increasing concerns about the security of their networks when they expose their organisations private data and networking infrastructure to Internet crackers. To provide the required level of protection, an organisation needs a security policy to prevent unauthorised users from accessing resources on the private network and to protect against the unauthorised export of private information. One convenient way to implement the network security policy is ***FIREWALL*** [w8].

In old days, when people used to live in wooden houses, the brick walls were built between buildings so that if a fire broke out, it should not spread from one building to another. These walls were called "***firewalls***". Today, when a network is connected to Internet, a *firewall* is placed between the network and the Internet to establish a controlled link and to erect a security perimeter. The aim of this security perimeter is to protect the network from network -based threats and attacks [b10]. The *firewall* acts as a single choke point which monitors and rejects application level network traffic. The operation of the *firewall* [b10] is shown in fig. 2.1.

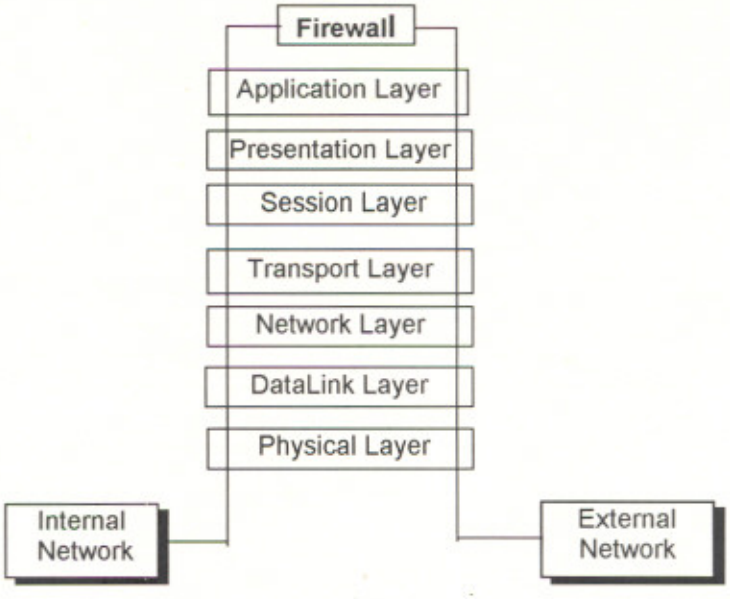


Figure 2.1: Firewall Operations

An ***Internet firewall*** is a system or group of systems that enforces a security policy between an organisation's network and the Internet. The *firewall* determines which inside services may be accessed from the outside, which outsiders are permitted access to the permitted inside services, and which outside services may be accessed by insiders.

For a *firewall* to be effective, all traffic to and from the Internet must pass through the *firewall*, where it can be inspected. The *firewall* must permit only authorised traffic to pass, and the *firewall* itself must be immune to penetration [b7]. Unfortunately, a *firewall system* cannot offer any protection once an attacker has got through or around the *firewall*.

Internet *firewall* is not just a *router*, a *bastion host*, or a combination of devices that provides security for a network. The *firewall* is part of an overall security policy that creates a perimeter defence designed to protect the information resources of the organisation. This security policy must include published security guidelines to inform users of their responsibilities. *Firewall* can be implemented in hardware (e.g. *router*) or software or a combination of both [w8].

2.1 Benefits of an Internet Firewall

Firewall manages access between the Internet and an organisation's private network. Some of the benefits are [w8] as follows.

1. Concentrates Network Security
2. Serves Centralised access 'Choke Point'
3. Monitors & logs Internet Usage
4. Good Location for ***NAT***
5. Good location for ***WWW & FTP*** servers

Without a *firewall*, each host system on the private network is exposed to attacks from other hosts on the Internet. This means that the security of the private network would depend on the "hardness" of each host's security features and would be only as secure as the weakest system.

Internet *firewall* allow the network administrator to define a centralised "choke point" that keeps unauthorised users such as hackers, crackers, vandals, and spies out of the protected network; prohibits potentially vulnerable services from entering or leaving the protected network; and provides protection from various types of routing attacks. An Internet *firewall* simplifies security management, since network security is consolidated on the *firewall* systems rather than being distributed to every host in the entire private network.

Firewall offers a convenient point where Internet security can be monitored and alarms generated. It should be noted that for organisations that have connections to the Internet, the question is not whether but, when attacks will occur. Network administrators must audit and log all significant traffic through the *firewall*. If the network administrator doesn't take the time to respond to each alarm and examine logs on a regular basis, there is no need for the *firewall*, since the network administrator will never know if the *firewall* has been successfully attacked! [b7].

For the past few years, the Internet has been experiencing an address space crisis that has made registered IP addresses a less plentiful resource [w2]. This means that organisations wanting to connect to the Internet may not be able to obtain enough registered IP addresses to meet the demands of their user population. An Internet *firewall* is a logical place to deploy a *Network Address Translator (NAT)* that can help alleviate the address space shortage [w8] and eliminate the need to renumber when an organisation changes *Internet service providers (ISPs)*.

An Internet *firewall* is the perfect point to audit or log Internet usage. This permits the network administrator to justify the expense of the Internet connection to management, pinpoint potential *bandwidth* bottlenecks [w4], and provide a method for departmental charge-backs if this fits the organisation's financial model.

An Internet *firewall* can also offer a central point of contact for information delivery service to customers. The Internet *firewall* is the ideal location for deploying *World Wide Web and FTP (File Transfer Protocol)* servers. The *firewall* can be configured to allow Internet access to these services, while prohibiting external access to other systems on the protected network.

Finally, some might argue that the deployment of an Internet *firewall* creates a single point of failure. It should be emphasised that if the connection to the Internet fails, the organisation's private network will still continue to operate--only Internet access is lost. If there are multiple points of access, each one becomes a potential point of attack that the network administrator must *firewall* and monitor regularly.

2.2 Limitations of an Internet Firewall

An Internet *firewall* cannot protect against attacks that do not go through the *firewall*. For example, if unrestricted dial-out is permitted from inside the protected network, internal users can make a direct ***SLIP*** or ***PPP*** connection to the Internet [w8]. Moreover, users who become irritated with the additional authentication required by *firewall* proxy servers may be tempted to circumvent the security system by purchasing a direct ***SLIP*** or ***PPP***

connection to an ISP. Since these types of connections bypass the security provided by the most carefully constructed *firewall*, they create a significant potential for back-door attacks. Users must be made aware that these types of connections are not permitted as part of the organisation's overall security architecture.

Internet *firewall* cannot protect against the types of threats posed by traitors or unwitting users. Firewall does not prohibit traitors or corporate spies from copying sensitive data onto floppy disks or PCMCIA cards and removing them from a building.

Firewall do not protect against attacks where a hacker, pretending to be a supervisor or a new employee, persuades a less sophisticated user into revealing a password or granting them "temporary" network access. Employees must be educated about the various types of attacks and about the need to guard and periodically change their passwords [b7].

Internet *firewall* cannot protect against the transfer of virus-infected software or files. Since there are so many different viruses, operating systems, and ways of encoding and compressing binary files, an Internet *firewall* cannot be expected to accurately scan each and every file for potential viruses. Concerned organisations should deploy anti-viral software at each desktop to protect against their arrival from floppy disks or any other source.

Finally, Internet *firewall* cannot protect against data-driven attacks. A data-driven attack occurs when seemingly harmless data is mailed or copied to an internal host and is executed to launch an attack. For example, a data-driven attack could cause a host to modify security-related files, making it easier for an intruder to gain access to the system. The deployment of proxy servers on a bastion host is an excellent means of prohibiting direct connections from the outside and reducing the threat of data-driven attacks.

2.3 Stance of the Firewall [b10]

The stance of a *firewall* system describes the fundamental security philosophy of the organisation. An Internet *firewall* may take one of two diametrically opposed stances:

- Everything not specifically permitted is denied. This stance assumes that a *firewall* should block all traffic and that each desired service or application should be implemented on a case-by-case basis. This is the recommended approach. It creates a very secure environment, since only carefully selected services are supported. The disadvantage is that it places security ahead of ease of use, limiting the number of options available to the user community.
- Everything not specifically denied is permitted. This stance assumes that a *firewall* should forward all traffic, and that each potentially harmful service should be shut off on a case-by-case basis. This approach creates a more flexible environment, with more services available to the user community. The disadvantage is that it puts ease of use ahead of security, putting the network administrator in a reactive mode and making it increasingly difficult to provide security as the size of the protected network grows.

2.4 Components of the Firewall System [w8]

A typical *firewall* is composed of one or more of the following building blocks:

1. Packet-filtering router
2. Application-level gateway (or proxy server)
3. Circuit-level gateway

2.4.1 Packet-Filtering Routers[b10]

A packet-filtering router makes a permit/deny decision for each packet that it receives. The router examines each datagram to determine whether it matches one of its packet-filtering rules. The filtering rules are based on the packet header information that is made available to the IP forwarding process. This information consists of the *IP* source address, the IP destination address, the encapsulated protocol (*TCP, UDP, ICMP, or IP* Tunnel), the *TCP/UDP* source port, the *TCP/UDP* destination port, the *ICMP* message type, the incoming interface of the packet, and the outgoing interface of the packet. If a match is found and the rule permits the packet, the packet is forwarded according to the information in the routing table. If a match is found and the rule denies the packet, the

packet is discarded. If there is no matching rule, a user-configurable default parameter determines whether the packet is forwarded or discarded.

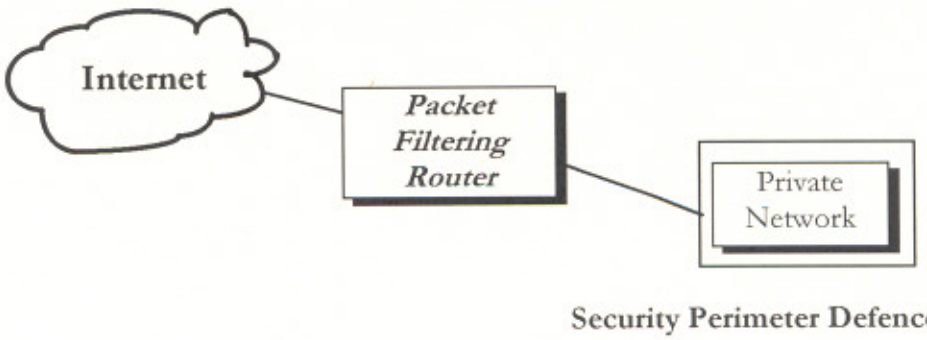


Figure 2.2: Packet Filtering Router

2.4.1.1 Service-Dependent Filtering [w8]

The packet-filtering rules allow a router to permit or deny traffic based on a specific service, since most service listeners reside on well-known *TCP/UDP* port numbers. For example, a Telnet server listens for remote connections on *TCP* port 23 and an *SMTP* server listens for incoming connections on *TCP* port 25. To block all incoming *Telnet* connections, the router simply discards all packets that contain a *TCP* destination port value equal to 23. To restrict incoming *Telnet* connections to a limited number of internal hosts, the router must deny all packets that contain a *TCP* destination port value equal to 23 and that do not contain the destination *IP* address of one of the permitted hosts. Some typical filtering rules include [b10]:

- Permit incoming *Telnet* sessions to a specific list of internal hosts.
- Permit incoming *FTP* sessions only to specific internal hosts.
- Permit all outbound *Telnet* sessions.
- Permit all outbound *FTP* sessions.
- Deny all incoming traffic from specific external networks.
- Deny all outgoing *SMTP* packets.

2.4.1.2 Service-Independent Filtering [w8]

There are certain types of attacks that are difficult to identify using basic packet header information because the attacks are service independent. Routers can be configured to protect against these types of attacks, but they are more difficult to specify since the filtering rules require additional information that can be learned only by examining the routing table, inspecting for specific *IP* options, checking for a special fragment offset, and so on. Examples of these types of attacks are [w5]:

Source IP Address Spoofing Attacks.

For this type of attack, the intruder transmits packets from the outside that pretend to originate from an internal host: the packets falsely contain the source IP address of an inside system. The attacker hopes that the use of a spoofed source IP address will allow penetration of systems that employ simple source address security where packets from specific trusted internal hosts are accepted and packets from other hosts are discarded. Source spoofing attacks can be defeated by discarding each packet with an inside source IP address if the packet arrives on one of the router's outside interfaces.

Source Routing Attacks.

In a source routing attack, the source station specifies the route that a packet should take as it crosses the Internet. This type of attack is designed to bypass security measures and cause the packet to follow an unexpected path to its destination. A source routing attack can be defeated by simply discarding all packets that contain the source route option.

Tiny Fragment Attacks.

For this type of attack, the intruder uses the IP fragmentation feature to create extremely small fragments and force the TCP header information into a separate packet fragment. Tiny fragment attacks are designed to circumvent user-defined filtering rules; the hacker hopes that a filtering router will examine only the first fragment and allows all

other fragments to pass. A tiny fragment attack can be defeated by discarding all packets where the protocol type is TCP and the IP Fragment Offset is equal to 1.

2.4.1.3 Benefits of Packet-Filtering Routers [w8]

The majority of Internet *firewall* systems are deployed using only a packet-filtering router. Other than the time spent planning the filters and configuring the router, there is little or no cost for implementing packet filtering since the feature is included as part of standard router software releases. Since Internet access is generally provided over a WAN interface, there is little impact on router performance if traffic loads are moderate and few filters are defined. Finally, a packet-filtering router is generally transparent to users and applications, so it does not require specialised user training or that specific software be installed on each host.

2.4.1.4 Limitations of Packet-Filtering Routers [b10]

Defining packet filters can be a complex task because network administrators need to have a detailed understanding of the various Internet services, packet header formats, and the specific values they expect to find in each field. If complex-filtering requirements must be supported, the filtering rule set can become very long and complicated, making it difficult to manage and comprehend. Finally, there are few testing facilities to verify the correctness of the filtering rules after they are configured on the router. This can potentially leave a site open to untested vulnerabilities.

Any packet that passes directly through a router could potentially be used to launch a data-driven attack. A data-driven attack occurs when seemingly harmless data is forwarded by the router to an internal host. The data contains hidden instructions that cause the host to modify access control and security-related files, making it easier for the intruder to gain access to the system.

Generally, the packet throughput of a router decreases as the number of filter increases. Routers are optimised to extract the destination IP address from each packet, make a relatively simple routing table lookup, and then forward the packet to the proper interface for transmission. If filtering is enabled, the router must not only make a

forwarding decision for each packet, but also apply all of the filter rules to each packet. This can consume CPU cycles and impact the performance of a system.

IP packet filters may not be able to provide enough control over traffic. A packet-filtering router can permit or deny a particular service, but it is not capable of understanding the context/data of a particular service. For example, a network administrator may need to filter traffic at the application layer in order to limit access to a subset of the available *FTP* or *Telnet* commands, or to block the import of mail or news-groups concerning specific topics. This type of control is best performed at a higher layer by proxy services and application-level gateways.

2.4.2 Application-Level Gateways [b7]

An *application-level gateway* allows the network administrator to implement a much stricter security policy than with a packet-filtering router. Rather than relying on a generic packet-filtering tool to manage the flow of Internet services through the *firewall*, special-purpose code (a proxy service [w8]) is installed on the gateway for each desired application. If the network administrator does not install the proxy code for a particular application, the service is not supported and cannot be forwarded across the *firewall*. Also, the proxy code can be configured to support only those specific features of an application that the network administrator considers acceptable while denying all other features.

This enhanced security comes with an increased cost in terms of purchasing the gateway hardware platform, the proxy service applications, the time and knowledge required to configure the gateway, a decrease in the level of service that may be provided to users, and a lack of transparency resulting in a less user-friendly system. As always, the network administrator is required to balance the organisation's need for security with the user community's demand for ease of use.

Users are permitted access to the proxy services, but they are never permitted to log in to the *application-level gateway*. If users are permitted to log in to the *firewall* system, the security of the *firewall* is threatened, since an intruder could potentially perform some

activity that compromises the effectiveness of the *firewall.* For example, the intruder could gain root access, install Trojan horses to collect passwords, and modify the security configuration files of the *firewall.*

Bastion Host Unlike packet-filtering routers, which allow the direct flow of packets between inside systems and outside systems, application-level gateways allow information to flow between systems but do not allow the direct exchange of packets. The chief risk of allowing packets to be exchanged between inside systems and outside systems is that the host applications residing on the protected network's systems must be secured against any threat posed by the allowed services.

An ***application-level gateway*** is often referred to as a **"*bastion host*"** because it is a designated system that is specifically armoured and protected against attacks. Several design features are used to provide security for a bastion host [b8]:

- The *bastion host* hardware platform executes a "secure" version of its operating system. For example, if the bastion host is a UNIX® platform, it executes a secure version of the UNIX operating system that is specifically designed to protect against operating system vulnerabilities and ensure *firewall* integrity.

- Only the services that the network administrator considers essential are installed on the bastion host. The reasoning is that if a service is not installed, it can't be attacked. Generally, a limited set of proxy applications such as *Telnet, DNS, FTP, SMTP,* and user authentication are installed on a bastion host.

- The *bastion host* may require additional authentication before a user is allowed access to the proxy services. For example, the bastion host is the ideal location for installing strong authentication using a one-time password technology where a smart card cryptographic authenticator generates a unique access code. In addition, each *proxy* service may require its own authentication before granting user access.

- Each proxy is configured to support only a subset of the standard application's command set. If a standard command is not supported by the proxy application, it is simply not available to the authenticated user.

- Each *proxy* is configured to allow access only to specific host systems. This means that the limited command/feature set may be applied only to a subset of systems on the protected network.

- Each *proxy* maintains detailed audit information by logging all traffic, each connection, and the duration of each connection. The audit log is an essential tool for discovering and terminating intruder attacks.

- Each *proxy* is a small and uncomplicated program specifically designed for network security. This allows the source code of the proxy application to be reviewed and checked for potential bugs and security holes. For example, a typical UNIX mail application may contain over 20,000 lines of code, while a mail proxy may contain fewer than 1000!

- Each *proxy* is independent of all other proxies on the bastion host. If there is a problem with the operation of any proxy, or if a future vulnerability is discovered, it can be un-installed without affecting the operation of the other proxy applications. Also, if the user population requires support for a new service, the network administrator can easily install the required proxy on the bastion host.

- A *proxy* generally performs no disk access other than to read its initial configuration file. This makes it difficult for an intruder to install *Trojan* horse sniffers or other dangerous files on the bastion host.

- Each *proxy* runs as a non-privileged user in a private and secured directory on the bastion host.

The figure 2.3 [b10] illustrates the operation of a ***Telnet proxy*** on a ***bastion host***. For example, the outside client wants to Telnet to an inside server protected by the *application-level gateway*.

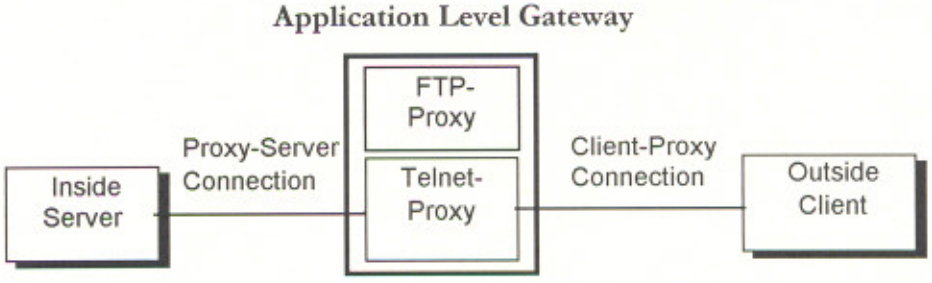


Figure 2.3: Telnet Proxy

The *Telnet proxy* never allows the remote user to log in or have direct access to the internal server. The outside client *Telnets* to the bastion host, which authenticates the user employing one-time password technology. After authentication, the outside client gains access to the user interface of the Telnet proxy. The Telnet proxy permits only a subset of the *Telnet* command set and determines which inside hosts are available for Telnet access. The outside user specifies the destination host and the Telnet proxy makes its own connection to the inside server and forwards commands to the inside server on behalf of the outside client. The outside client believes that the *Telnet proxy* is the real inside server, while the inside server believes that the *Telnet proxy* is the outside client.

Authentication can be based on either something the user knows (like a password) or something the user physically possesses (like a smart card) [w9]. Both techniques are subject to theft, but using a combination of both methods increases the likelihood of correct user authentication. In the Telnet example, the proxy transmits a challenge and the user, with the aid of a smart card, obtains a response to the challenge. Typically, a user unlocks the smart card by entering their PIN number and the card, based on a

shared "secret" encryption key and its own internal clock, returns an encrypted value for the user to enter as a response to the challenge.

2.4.2.1 Benefits of Application-Level Gateways [w8]

There are many benefits to the deployment of application-level gateways. They give the network manager complete control over each service, since the proxy application limits the command set and determines which internal hosts may be accessed by the service. Also, the network manager has complete control over which services are permitted, since the absence of a proxy for a particular service means that the service is completely blocked.

Application-level gateways have the ability to support strong user authentication and provide detailed logging information. Finally, the filtering rules for an application-level gateway are much easier to configure and test than for a packet-filtering router.

2.4.2.2 Limitations of Application-Level Gateways [b7]

The greatest limitation of an *application-level gateway* is that it requires either that users modify their behaviour, or that specialised software be installed on each system that accesses proxy services. For example, *Telnet* access via an *application-level gateway* requires two user steps to make the connection rather than a single step. However, specialised end-system software could make the *application-level gateway* transparent by allowing the user to specify the destination host rather than the *application-level gateway* in the *Telnet* command.

2.4.3 Circuit-Level Gateways [b10]

A *circuit-level gateway* is a specialised function that can be performed by an application-level gateway. A *circuit-level gateway* simply relays *TCP* connections without performing any additional packet processing or filtering as shown in Fig. 2.4.

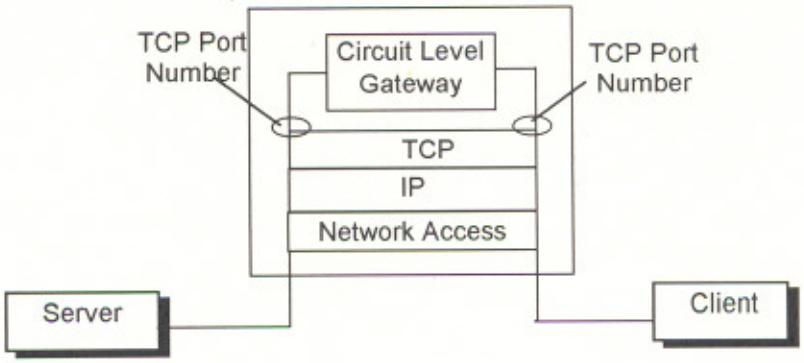


Figure 2.4: Circuit-Level Gateway

The figure 2.5 [w8] illustrates the operation of a typical *Telnet* connection through a *circuit-level gateway*. The *circuit-level gateway* simply relays the *Telnet* connection through the *firewall* but does no additional examination, filtering, or management of the *Telnet* protocol. The *circuit-level gateway* acts like a wire, copying bytes back and forth between the inside connection and the outside connection. However, because the connection appears to originate from the *firewall* system, it conceals information about the protected network.

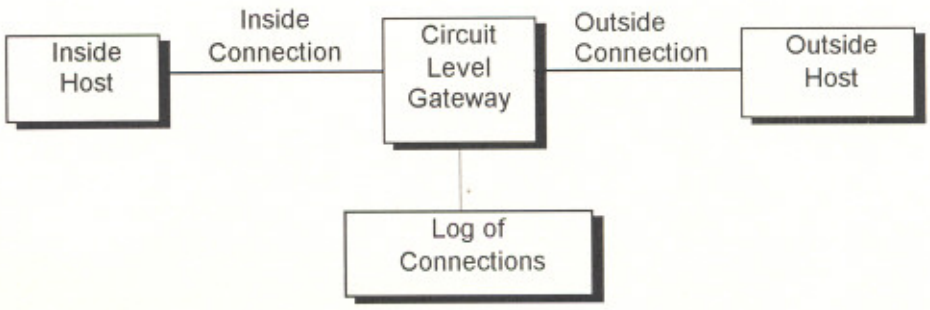


Figure 2.5: Telnet Connection through Circuit Level Gateway

Circuit-level gateways are often used for outgoing connections where the system administrator trusts the internal users. Their chief advantage is that a bastion host can be configured as a hybrid gateway supporting application-level or proxy services for inbound connections and circuit-level functions for outbound connections. This makes the *firewall* system easier to use for internal users who want direct access to Internet

services, while still providing the *firewall* functions needed to protect the organisation from external attack.

There is no single correct answer for the design and deployment of Internet *firewalls*. Each organisation's decision will be influenced by many different factors such as their overall security policy, the technical background of their staff, cost, and the perceived threat of attack. Since the benefits of connecting to the global Internet probably exceed its costs, network managers should proceed with an awareness of the dangers and an understanding that, with the proper precautions, their networks can be as safe as they need them to be [p2].

Chapter 3

DESIGN AND IMPLEMENTATION

As we are designing a *Proxy Server* for *World Wide Web (WWW)* service of Internet and the *WWW* services are implemented over *Hypertext Transfer Protocol (HTTP)*. Hence, before moving to the design and implementation, we should look at the application layer protocol (*HTTP*) specifications.

3.1 HTTP Specifications [f4]

HTTP is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. A feature of *HTTP* is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

HTTP is used for communication between user agents and various gateways, allowing hypermedia access to existing Internet protocols like ***SMTP, NNTP, FTP, Gopher, and WAIS. HTTP*** is designed to allow such gateways, via *proxy servers*, without any loss of the data conveyed by those earlier protocols.

3.1.1 Overall Operations

The HTTP protocol is based on a request/response paradigm. A requesting program (termed a client) establishes a connection with a receiving program (termed a server) and sends a request to the server in the form of a request method, URI (**Uniform Resource Identifier**), and protocol version, followed by a *MIME* (***Multipurpose Internet Mail Extension***) message containing request modifiers, client information, and possible body content. The server responds with a status line, including its protocol version and a success or error code, followed by a *MIME-like* message containing server

information, entity meta-information, and possible body content. On the Internet, the communication generally takes place over a *TCP/IP* connection. The default port is *TCP* 80 , but other ports can be used. This does not preclude the *HTTP*/1.0 protocol from being implemented on top of any other protocol on the Internet, or on other networks. *HTTP* only presumes a reliable transport; any protocol that provides such guarantees can be used.

Current practice requires that the connection be established by the client prior to each request and closed by the server after sending the response. Both clients and servers must be capable of handling cases where either party closes the connection prematurely, due to user action, automated time-out, or program failure. In any case, the closing of the connection by either or both parties always terminates the current request, regardless of its status.

3.1.2 Protocol Parameters

3.1.2.1 HTTP Version

HTTP uses a "<major>.<minor>" numbering scheme to indicate versions of the protocol. The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further *HTTP* communication, rather than the features obtained via that communication. The version of an *HTTP* message is indicated by an *HTTP*-Version field in the first line of the message. If the protocol version is not specified, the recipient must assume that the message is in the simple *HTTP*/0.9 format. Applications sending Full-Request or Full-Response messages, must include an *HTTP*-Version of "***HTTP/1.0***".

Proxies must be careful in forwarding requests that are received in a format different than that of the proxy's native version. Since the protocol version indicates the protocol capability of the sender, a proxy must never send a message with a version indicator which is greater than its native version; if a higher version request is received, the proxy must either downgrade the request version or respond with an error.

3.1.2.2 Uniform Resource Identifiers (URI)

As far as *HTTP* is concerned, Uniform Resource Identifiers are simply formatted strings which identify--via name, location, or any other characteristic, a network resource. *URIs in HTTP/1.0* can be represented in absolute form or relative to some known base *URI*, depending upon the context of their use e.g absolute URI are used with proxies whereas clients use relative *URI's*.

3.1.2.3 http URL

The "http" scheme is used to locate network resources via the *HTTP* protocol.

http_URL = "http:" "//" host_name [":" port] abs_path

If the port is empty or not given, port 80 is assumed. The semantics are that the identified resource is located at the server listening for *TCP* connections on that port of that host, and the *Request-URI* for the resource is **abs_path**. If the abs_path is not present in the URL, it must be given as "/" when used as a Request-URI.

3.1.3 HTTP Message

HTTP messages consist of requests from client to server and responses from server to client. Both messages may include optional header fields and an entity body. The entity body is separated from the headers by a null line (i.e., a line with nothing preceding the CR LF).

Formats of the Request and Response Messages are as follows:

Request = Request-Line
Request-Header|Entity-Header
CRLF
[Entity-Body]

Response = Status-Line

Response-Header | Entity-Header

CRLF

[Entity-Body]

3.1.3.1 Request Message

A request message from a client to a server includes, within the first line of that message, the **method** to be applied to the resource requested, the identifier of the resource, and the protocol version in use.

Request-Line

The **Request-Line** begins with a method token(GET,HEAD & POST), followed by the **Request-URI** and the protocol version, and ending with **CRLF**. The elements are separated by **SP** (White Space) characters. No CR or LF are allowed except in the final CRLF sequence. Format of **Request Line** is as shown below.

Request-Line = *Method* SP *Request-URI* SP *HTTP-Version* CRLF

- **Method**

The ***Method*** token indicates the method to be performed on the resource identified by the **Request-URI**. The **method** is case-sensitive.

Method = "GET" | "HEAD" | "POST"

The **GET** method means retrieve whatever information (in the form of an entity) is identified by the **Request-URI**. If the **Request-URI** refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and

not the source text of the process, unless that text happens to be the output of the process. The semantics of the **GET** method changes to a "conditional GET" if the request message includes an **If-Modified-Since** header field. A conditional **GET** method requests that the identified resource be transferred only if it has been modified since the date given by the **If-Modified-Since** header. The conditional GET method is intended to reduce network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring unnecessary data.

The **HEAD** method is identical to **GET** except that the server must not return any Entity-Body in the response. The meta-information contained in the *HTTP* headers in response to a **HEAD** request should be identical to the information sent in response to a **GET** request. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

The **POST** method is used to request that the destination server accept the entity enclosed in the request as a new subordinate of the resource identified by the **Request-URI** in the Request-Line. **POST** is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- Providing a block of data, such as the result of submitting a form to a data-handling process;
- Extending a database through an append operation.

The actual function performed by the **POST** method is determined by the server and is usually dependent on the **Request-URI**. A successful **POST** does not require that the entity be created as a resource on the origin server or made accessible for future reference. If a resource has been created on the origin server, the response should be

201 (created) and contain an entity (preferably of type "text/html") which describes the status of the request and refers to the new resource.

A valid Content-Length is required on all **POST** requests. An ***HTTP*** server should respond with a 400 (bad request) message if it cannot determine the length of the request message's content.

Servers should return the status code 501 (not implemented) if the **method** is unknown or not implemented.

- **Request-URI**

The **Request-URI** is a Uniform Resource Identifier and identifies the resource upon which to apply the request.

Request-URI = absolute-URI | abs_path

The two options for **Request URI** are dependent on the nature of the request. The **absolute-URI** form is only allowed when the request is being made to a proxy server. The proxy is requested to forward the request and return the response. If the request is **GET or HEAD** and a response is cached, the proxy may use the cached message if it passes any restrictions in the Expires header field. An example **Request-Line** can be

GET http://www.w3.org/hypertext/WWW/TheProject.html HTTP/1.0

For example, a client wishing to retrieve the above resource directly from the origin server would create a TCP connection to port 80 of the host "www.w3.org" and send the following request line:

GET /hypertext/WWW/TheProject.html HTTP/1.0

followed by the remainder of the Request.

Request Header

The request header fields allow the client to pass additional information about the request, and about the client itself, to the server. All header fields are optional and conform to the generic HTTP-header syntax.

Request-Header = Authorization

| From

| If-Modified-Since

| Host

| Referer

| User-Agent

| Proxy-Connection

- **Authorization**

A user agent that wishes to authenticate itself with a server, uses the **Authorization** request-header field with the request. The **authorization** field value consists of *credentials* containing the authentication information of the user agent for the realm of the resource being requested. If a request is authenticated and a realm is specified, the same credentials should be valid for all other requests within this realm.

- **From**

The **From** request header field, if given, should contain an Internet e-mail address for the human user who controls the user agent. For example

From : ttti1@chd.nic.in

This header field may be used for logging purposes and as a means for identifying the source of valid or unwanted requests. The client should not send the From header field

without the user's approval, as it may conflict with the user's privacy interests or their site security policy.

- **Host**

This field gives the name or IP address of the *HTTP* server on which the requested resource is to be searched. Proxies use this field to get the address of the server for forwarding the client's request.

- **If-Modified-Since**

The **If-Modified-Since** header field is used with the **GET** method to make it conditional: if the requested resource has not been modified since the time specified in this field, a copy of the resource will not be returned from the server; instead, a 304 (not modified) response will be returned without any Entity-Body. The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead.

- **Referer**

The **Referer** request header field allows the client to specify, for the server's benefit, the address (***URI***) of the resource from which the Request-URI was obtained. This allows a server to generate lists of back-links to resources for interest, logging, optimized caching, etc. It also allows obsolete or mistype links to be traced for maintenance. For example:

Referer: http://www.w3.org/hypertext/DataSources/Overview.html

- **User-Agent**

The **User-Agent** field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user

agent limitations. Although it is not required, user agents should always include this field with requests. For example:

User-Agent: MOZILA-GOLD/3.0

3.1.3.2 Response Message

After receiving and interpreting a request message, a server responds in the form of an HTTP response message. The response message contains the **status line** followed by the header fields and optional entity body.

Status-Line

The first line of a Response message is the Status-Line, consisting of the protocol version followed by a numeric status code and its associated textual phrase, with each element separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

Status-Line = *HTTP-Ver* SP *Status-Code* SP *Reason-Phrase* CRLF

- **Status Codes and Reason Phrases**

The **Status-Code** element is a 3-digit integer result code of the attempt to understand and satisfy the request. The **Reason-Phrase** is intended to give a short textual description of the **Status-Code**. The **Status-Code** is intended for use by automata and the Reason-Phrase is intended for the human user.

The first digit of the **Status-Code** defines the class of response. The last two digits do not have any categorisation role. There are 5 values for the first digit:

1xx: Informational - Not used, but reserved for future use

2xx: Success - The action was successfully received, understood and accepted.

3xx:Redirection - Further action must be taken in order to complete the request

4xx: Client Error - The request contains bad syntax or cannot be fulfilled

5xx: Server Error - The server failed to fulfill an apparently valid request

The individual values of the numeric status codes defined for HTTP/1.0, and an example set of corresponding Reason-Phrase's, are presented below.

Status-Code = "200" ; OK

| "201" ; Created

| "202" ; Accepted

| "204" ; No Content

| "301" ; Moved Permanently

| "302" ; Moved Temporarily

| "304" ; Not Modified

| "400" ; Bad Request

| "401" ; Unauthorized

| "403" ; Forbidden

| "404" ; Not Found

| "500" ; Internal Server Error

| "501" ; Not Implemented

| "502" ; Bad Gateway

| "503" ; Service Unavailable

Response Header

The **response header** fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields are not intended to give information about an Entity-Body returned in the response, but about the server itself.

Response-Header = Server | WWW-Authenticate

- **Server**

The **Server** response header field contains information about the software used by the origin server to handle the request. For example:

Server: MICROSOFT IIS/2.0

If the response is being forwarded through a ***proxy***, the ***proxy*** application should not add its data to the product list. Revealing the specific software version of the server may allow the server machine to become more vulnerable to attacks against software that is known to contain security holes. Server implementors should make this field a configurable option by taking into account the security considerations..

- **WWW-Authenticate**

The **WWW-Authenticate** header field must be included in 401 (unauthorized) response messages. ***Proxies*** must be completely transparent regarding user agent authentication. That is, they must forward the **WWW-Authenticate** and **Authorization** headers untouched, and must not cache the response to a request containing **Authorization**. *HTTP* does not provide a means for a client to be authenticated with a *proxy*.

3.1.3.3 Entity

Both Request and Response messages may transfer an entity within some requests and responses. An entity consists of **Entity-Header** fields and an **Entity-Body**.

Entity Header

Entity-Header fields define optional meta-information about the Entity-Body or, if no body is present, about the resource identified by the request.

Entity-Header = Content-Encoding

| Content-Length

| Content-Type

Entity Body

An **entity-body** is included with a request message only when the request method calls for one. This specification defines one request method **POST**, that allows an **entity-body**. In general, the presence of an entity-body in a request is signaled by the inclusion of a **Content-Length** header field in the request message headers. *HTTP*/1.0 requests containing content must include a valid **Content-Length** header field.

For response messages, whether or not an entity-body is included with a message is dependent on both the request **method** and the response **status-code**. All responses to the **HEAD** request method must not include a body, even though the presence of content header fields may lead one to believe they do. The responses 204 (no content) and 304 (not modified) must not include a message body.

- **Content Type**

When an **Entity-Body** is included with a message, the data type of that body is determined via the header fields **Content-Type** and **Content-Encoding**. A

Content-Type specifies the media type of the underlying data whereas **Content-Encoding** may be used to indicate coding applied to the type, usually for the purpose of data compression, that is a property of the resource requested.

- **Content Length**

Content-Length header field value in bytes represents the length of the **Entity-Body**. Otherwise, the body length is determined by the closing of the connection by the server. Closing the connection cannot be used to indicate the end of a request body, since it leaves no possibility for the server to send back a response. Therefore, *HTTP* **POST** requests must include a valid **Content-Length** header field. If a request contains an entity body and **Content-Length** is not specified, then the server should send a 400 (bad request) response.

3.1.4 Security Considerations

Like any generic data transfer protocol, *HTTP* cannot regulate the content of the data that is transferred, nor is there any a priori method of determining the sensitivity of any particular piece of information within the context of any given request. Therefore, applications are encouraged to supply as much control over this information as possible to the provider of that information. Three header fields are worth special mention in this context: **Server, Referer and From.** Revealing the specific software version of the server may allow the server machine to become more vulnerable to attacks against software that is known to contain security holes. Implementers are encouraged to make the Server header field a configurable option. The **Referer** field allows reading patterns to be studied and reverse links drawn. Although it can be very useful, its power can be abused if user details are not separated from the information contained in the **Referer**. The information sent in the From field might conflict with the user's privacy interests or their site's security policy, and hence it should not be transmitted without the user being able to disable, enable, and modify the contents of the field. The user must be able to set the contents of this field within a user preference or application defaults configuration.

3.2 Design Aspects

Before we discuss the implementation details, it is important to discuss the various design aspects being considered while designing a filtering proxy server.

Since a *Proxy Server* is an intermediary program which acts as both a server and a client for the purpose of forwarding requests. It doesn't just forward requests on to the real Internet services . The *proxy server* can control what user's do, because it can make decisions about the requests it processes and can filter the requests based on the filtering rules. Depending on the site's security policy, requests may be allowed or refused [b7].

So main consideration in designing a *Proxy Server* is to see, how the underlying protocol specifications and the filtering rules can be implemented. The overall design process can be divided into the following steps:

1. Service filtering.
2. IP Address Filtering.
3. Content Filtering
4. IP Address Translation
5. Request Processing
6. Report Generation

3.2.1 Service filtering [b10]

We know that various application level services of ***TCP*** are distinguishable by the ***TCP*** Port numbers. For example , port no. 21 is reserved for ***FTP***, 80 for ***HTTP***, 25 for ***SMTP*** and so on. By matching the port number of the incoming request against the filtering rules, the request may be blocked or forwarded by the application proxy. For example, all ***FTP*** requests may be blocked by the proxy server.

3.2.2 IP Address filtering [b7]

Filtering in this way lets to restrict the flow of packets based on the source and destination IP addresses of the packets i.e through IP Address Filtering, we can block

access to certain undesirable web-sites. It can be implemented by maintaining a database of the denied source/destination IP Addresses. If the requested IP address from client, matches with one of the denied IP Address then no further request processing is done by the proxy server and a "connection refused" message is generated for the client.

3.2.3 Content Filtering

HTTP server header contains a "**Content-Type**" header field, which specifies the type of resource or data to be accessed. For example if the accessed data stream is a ***gif*** image then this header field will contain the information **image/gif**, and for a HTML document , this field will be **text/html**. By parsing this header field, any type of content filtering can be done. For example we can filter the ***GIF (Graphics Interchange File Format)*** images to eliminate the GIF animation.

3.2.4 IP Address Translation [w8]

Our *Proxy Server* translates the internal addresses to the address of the system where it is running. Address translation feature helps to overcome the following limitations :

- ***Limited allocation of IP address*** : Private network addressing helps to extend the existing IP address space.
- ***Invalid internal address*** : Translate the unregistered address to a legal IP Address.
- ***Concealing internal addresses from the Internet***: Private network addressing helps to hide the network's internal architecture from the Internet.

3.2.5 Request Processing [b10]

As *Proxy Server* is handling messages from both Clients (within the LAN) and Servers (on the Internet), so it has to behave both as a server as well as client. Initially it acts as a server for receiving the requests from clients. *HTTP* Request headers sent by clients are parsed according to the *HTTP* specifications [f4]. During **Request-Line** parsing the

request-URI is converted in to the **absolute path** and the **Proxy-Connection** header field is removed from the Request-header. After rewriting the Request header, proxy become client and forwards the new header to the HTTP server on the Internet. The address of this *HTTP* server is obtained from the **HOST** Address field of the request header sent by the client.

3.2.6 Report Generation [w8]

Because *proxy servers* understand the underlying protocol, they allow logging to be performed in a effective way e.g instead of logging all of the data transferred, an HTTP proxy can log only the accessed URL's and the server responses received. This results in a much smaller and more useful log. This logging information enables the administrator to monitor Internet usage.

3.3 Implementation

World wide web services are implemented using the **H**yper **T**ext **T**ransfer **P**rotocol (HTTP). *HTTP* enables clients (browsers) or proxies to fetch pages from the HTTP server and to return data to the server. Since *HTTP* communication generally takes over *TCP/IP* connections, so to implement the proxy server, we need some Network ***Application Programming Interface(API)*** which supports *TCP/IP* programming. Although many network *API's* like ***Inet32, Berkeley Sockets, Windows Sockets*** etc. are available, but to implement our design, we have chosen the Windows sockets (WinSock) API under Visual C++ environment.

3.3.1 WinSock API [b4]

3.3.1.1 Introduction

WinSock is an open interface for network programming under *Microsoft windows*. It is open in the sense that the WinSock specification, header file and libraries all we need to develop applications are publicly available.

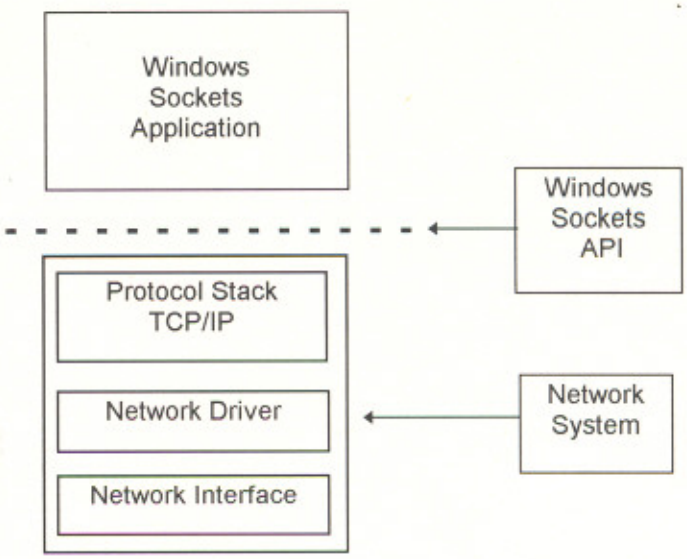


Figure. 3.1 WinSock Network Model

The *WinSock API* provides standard access to the network services of an underlying protocol stacks (*TCP/IP, IPX/SPX* etc.) to any Microsoft windows application. The functions call, data structures provided by the *WinSock API* isolates the user from the lower level details of the network processing. The *API* divides the labour between network application and network protocol stack, i.e. it is the applications job to provide a good user interface and to format & parse data, whereas the *WinSock* Protocol stack's job is to send or receive that data using standard transport protocols, drivers and network media. The *WinSock* network model is shown in fig. 3.1.

3.3.1.2 Client - Server Model

WinSock API is suitable for client-server model based applications. Every network application has a communication end point, called ***socket*** in *WinSock API*. There are two types of end points, client socket & server socket..

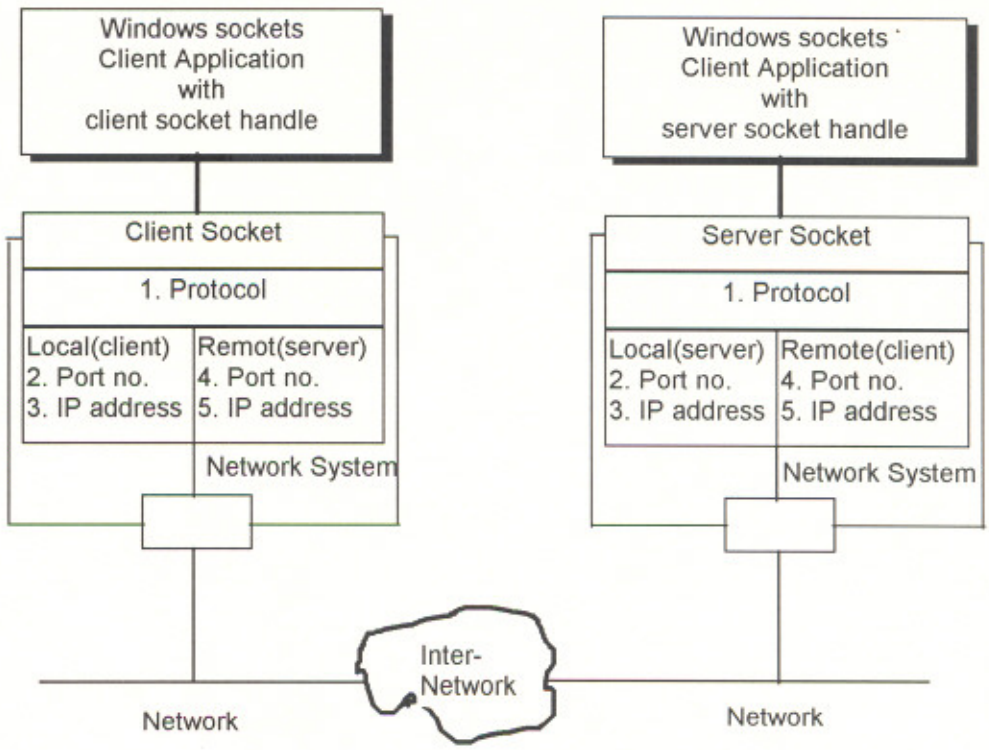


Figure. 3.2 Client Server Model

The server application execute first and waits to receive whereas client execute second and sends the first network packet. After initial contact either of the two is capable of sending or receiving the network packets. For two socket to communicate as client & server they must have the same socket types i.e either both sockets must be stream ***(TCP)*** sockets or both must be data-gram ***(UDP)*** sockets. A schematic of client-server model is shown in Fig.3.2. The brief description of various functions of WinSock API is given in Appendix I

Client application must be able to locate and identify a server socket, so a server application names its socket to establish its identity. Each socket name consists of the IP address, Port number & Protocol type. When the client socket successfully connects a server socket their two names combine to form an association. This association

establishes the identity of both sockets thereafter. The information in an association identifies and guides each packet through the network from network application to its peer application. The association information comprises of five elements :

1. Client IP address
2. Server IP address
3. Client port no.
4. Server port no.
5. Protocol (same for both client & server)

In case of stream sockets the life of an association correspond directly to the creation and destruction of the TCP virtual circuits between a client & server. Since data-gram sockets are connection less so each data-gram packet transmitted creates and destroy an association.

3.3.1.3 Operation modes

Applications operation modes determine its design and capabilities as much as its role as a client or server. WinSock API supports three operating modes.

1. Blocking Mode
2. Non-Blocking Mode
3. Asynchronous Mode

In blocking operation mode, WinSock functions returns only when the operation completes. In non-blocking mode function return immediately without waiting for the operation to be completed. Asynchronous mode is a special type of non-blocking mode. Non-blocking mode requires polling to detect the operation completion whereas in asynchronous mode the WinSock DLL will send a message to notify the application when a pending operation completes or when to retry an operation.

Three modes can be compared to each other in terms of their resulting code complexity, application flexibility and system overhead. For example more system over head is incurred in non-blocking mode which adversely affect the data throughput.

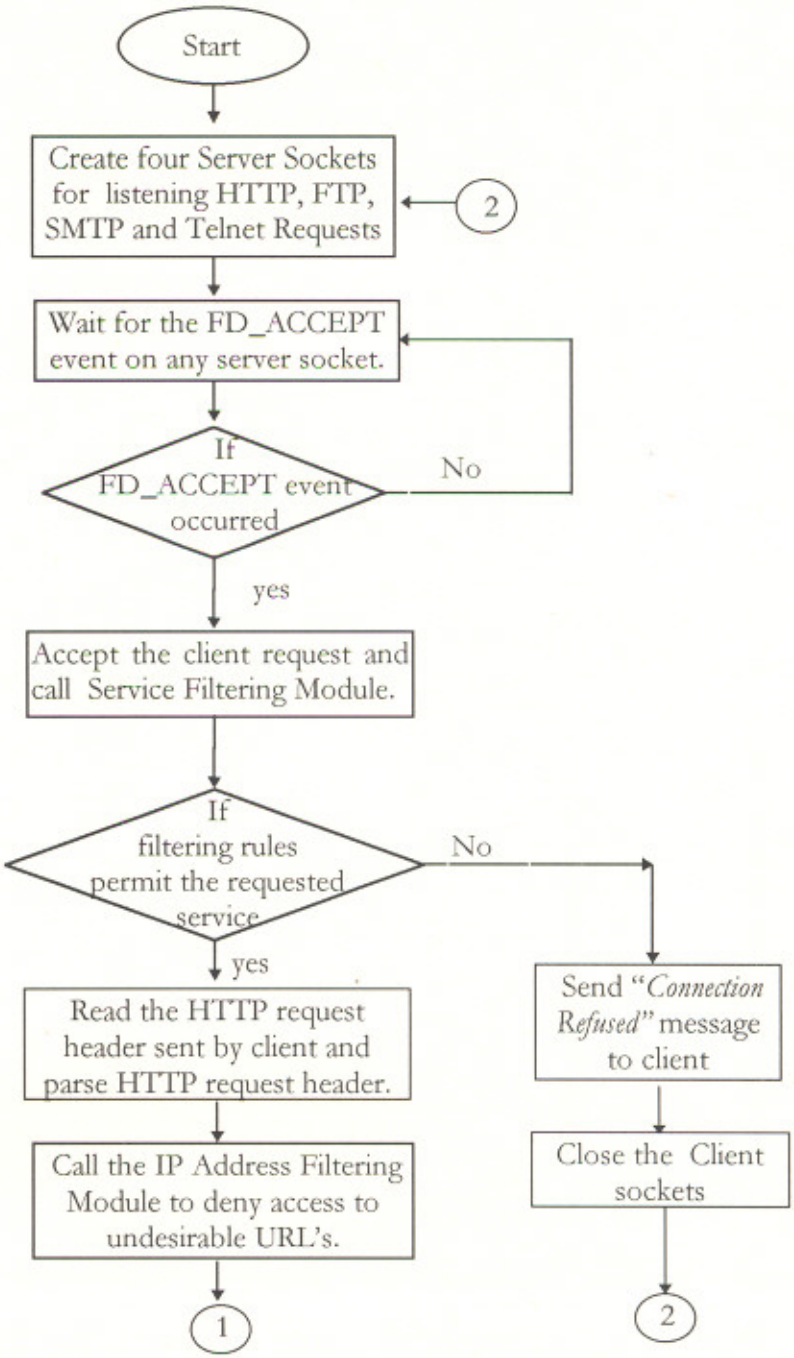
To implement proxy server design, discussed in previous chapter, we have chosen the Asynchronous operating mode of WinSock because the design requires the simultaneous processing of multiple sockets, as well as fast & efficient throughput of bulk data. The algorithm of the program is given below.

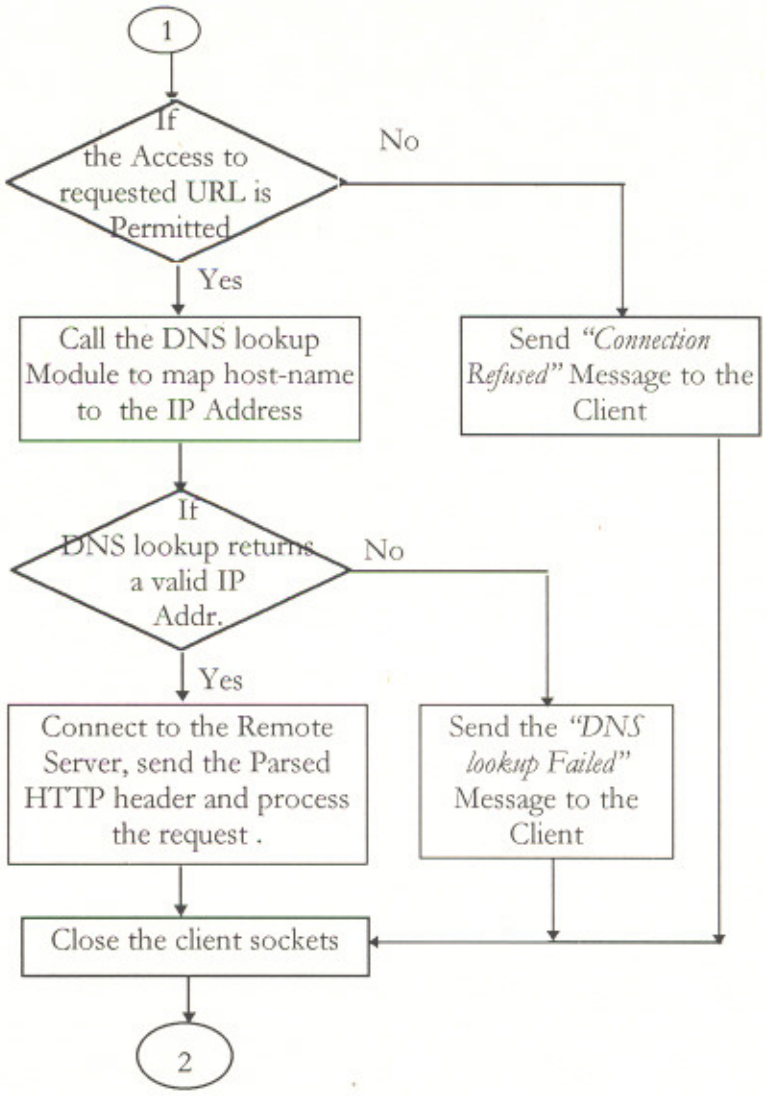
3.3.1.4 Network Application Sketch

All network application client & server follow the following steps.

1. Open a Socket
2. Name the Socket
3. Associate with another socket
4. Send & receive between sockets
5. Close the socket

3.3.2 Flowchart




1
If the Access to requested URL is Permitted
No
Yes
Call the DNS lookup Module to map host-name to the IP Address
Send "Connection Refused" Message to the Client
If DNS lookup returns a valid IP Addr.
No
Yes
Connect to the Remote Server, send the Parsed HTTP header and process the request .
Send the "DNS lookup Failed" Message to the Client
Close the client sockets
2

3.3.3 Algorithm

Create Four Server sockets for listening HTTP, FTP, TelNet and SMTP requests. These sockets act as doorman and waits for the incoming client requests. The following 'C' Language code and flowchart illustrates the implementation of this step.

```
SOCKET   CreatServerSocket(HWND hWnd, u_short port_no)
{
  SOCKET hLstnSocket ;     // Socket Descriptor
  int nFuncStatus;         //  Return code of the function to check for errors

// Create a Stream Socket [ b4 ]

  hLstnSocket = CreateSocket();

// Set the socket in Asynchronous mode to sense the FD_ACCEPT event.

  nFuncStatus =WSAAsyncSelect(hLstnSocket, hWnd, WSA_ASYNC, FD_ACCEPT);

// Check the return code for various WinSock Errors
  if(nFuncStatus == SOCKET_ERROR)
        ReportError("InitLstnSocket",WSAGetLastError());

 // Bind the Socket  to local machine IP Address and the user defined Port_no.

  BindSocket(hLstnSocket,port_no);

// Set the Socket in the Listening Mode.
  SetListeningSocket(hLstnSocket,nPendingConnection);

  return(hLstnSocket);

} /* end of CreateServerSocket() */
```

- Accept the client request and Apply Service Filtering. Whenever a request is received by a server socket, it creates a new socket for enabling communication between the client machine and the server, whereas the server socket remains intact and continuously waits for more client requests. Once the association is established, the new socket handle contains all the information about the peer and local socket. By extracting and comparing the port number of the local socket with the user

defined Proxy Server Ports, the service filtering is performed i.e all the FTP, Telnet and SMTP requests are blocked by the proxy server.

```
SOCKET AcceptConn(SOCKET s,HWND hWnd)
{
        SOCKET      new_socket;     // New Socket handle for Client
        SOCKADDR    RmtName;        // structure to store the peer Information.
        SOCKADDR_IN sockAddr;
        LPCONNDATA lpstConn;
        int nSelectStatus;

  // Accept client request when FD_ACCEPT event triggers Server Socket.
      new_socket = accept(s,(LPSOCKADDR)&stRmtName,&addrlength);

  // Check for the Winsock Errors, Report Error If Any.

      if(new_socket == SOCKET_ERROR)
            ReportError("AcceptConn Refused",WSAGetLastError());


      BOOL flag=FALSE;

  // Call Service Filtering Module

      flag = setFilter(new_socket);

  // If Filtering Rules Permits the Request Processing then
  // Store the new Socket Information into the Linked List and
  // Set the FD_READ Event on the new Socket to read the Request Headers.

      if(flag) {
            if(new_socket != INVALID_SOCKET) {

              // Create a node in the link list to Insert the Socket info
                    lpstConn = NewConn(new_socket);

             // If the new Node can not be created then close the client socket

                    if (!lpstConn)
                         CloseConn(new_socket, hWnd);

             // Set the FD_READ Event on the new socket
                    nSelectStatus=WSAAsyncSelect(new_socket,hWnd,
                                          WSA_ASYNC,FD_READ);
```

```
            // Report WinSock Error if any

                    if(nSelectStatus == SOCKET_ERROR)
                        ReportError("InitLstnSocket",WSAGetLastError());

            }   // end of if(new_socket!= INVALID_SOCKET)
        }  // end of  if(flag)

 return(new_socket);

} // end of  AcceptConn()


//  Module to implement the service Filtering
BOOL setFilter(SOCKET s)
{
      char tempBuf[512]= "HTTP/1.0 200 OK\nContent-Type: text/html\n
                          Accept-Ranges: bytes\nContent-Length:36\n\r";
      char chBuf[512];
      int addrlen = sizeof(SOCKADDR);
      int nRet;
      u_short port_no;
      SOCKADDR_IN  sockAddr;

//  Get  the  local socket Information and store in  SOCKADDR structure
      nRet = getsockname(s, (LPSOCKADDR)&sockAddr,&addrlen);

//  Convert the the port number from Network byte order to the host byte order
      port_no =  ntohs(sockAddr.sin_port);

//  If  the  request is received by HTTP proxy then permit  the request
      if(port_no == HTTP_PORT_NUM)
            return(TRUE);

//  otherwise send the Connection Refused Message to the Client

      else  {
          nRet=SendData(s,(LPSTR)tempBuf,strlen(tempBuf));
          strcpy(chBuf,"<html><head><h1>Connection refused !
                                              <h1><head><html>");
           nRet=SendData(s,(LPSTR)chBuf,36);
           CloseConn(s, hWnd);
          return(FALSE);
     }   // end  of  else part
} // end of set_filter
```

- Read the HTTP request Header from the Client and Parse the Header according to the HTTP Specifications [w10]. This parsing involves the conversion of **Absolute-URI to absolute-Path ,** extraction of **Method (Get, Head or Post) ,** removal of **Proxy-Connection** field as well as extraction of the Host-Name/IP Address from the **Host** Field. A log of all the HTTP request headers is also maintained.

// This Function takes as arguments the size of HTTP request header size and //a far pointer to the link-list node which contains the info of the client socket

```
LPSTR  ParseClientRequest(LPCONNDATA lpstConn, int nRequestSize)
{

 char    buf[MAX_LINE_SIZE];          // buffer to store one line of header
 char    szIPAddress[33];             // buffer to store host-name/IP Addr
 LPSTR  lineBuf=NULL;                  // pointer to  current line of header
 LPSTR  wordBuf=NULL;                 // pointer to the extracted word
 LPSTR  IPAddress= NULL;
 LPSTR tempBuf= (LPSTR)lpstConn->chRequestBuf;
 int      bufIndex = 0;
 int      tempBufIndex=0;

 while(nRequestSize != 0)
 {

// Extract one line from the header by checking the CRLF sequence.
      lineBuf = ReadTillNewLine(tempBuf);

// If Empty line then Skip it
      if (lineBuf == NULL)   {
           MessageBox(GetFocus(),"Empty Line","Parsing",MB_OK);
           break;
      }  // end of if(lineBuf==NULL)

      strcpy(buf,lineBuf);
      bufIndex = strlen(buf);

// Write the Parsed line into the Log File (Proxy.log)
      _lwrite(hLogFile,(LPSTR)buf,bufIndex);

// Ckeck for the end of Request Header (CRLF sequence At line start)
      if(bufIndex == 2) {
           strncpy(lpstConn->chTempBuf+tempBufIndex,tempBuf,nRequestSize);
          tempBufIndex += nRequestSize;
          break;
```

```
    }  // end of (bufIndex==2)

// Extract the first word of the line
    wordBuf = ReadTillBlankChar(buf);

// Increment buf index to point to second line
    tempBuf +=bufIndex;
    nRequestSize-= bufIndex;

    if(wordBuf != NULL)  {

 // Remove the Proxy-Connection Header Field from the Header
        if( strcmp(wordBuf,"PROXY-CONNECTION:")==0)
            continue;
//  Extract the Method Name from the Request Line

        if((strcmp(wordBuf,"GET")==0)||(strcmp(wordBuf,"POST")==0))
          {

    // Convert Absolute-URI to Absolute-Path  e.g
    //  http://www.microsoft.com/dafault.htm  to /default.htm

            LPSTR szAbsUri=NULL;
            strcpy(URL, buf);

            if(strcmp(wordBuf,"GET")==0)
                szAbsUri = (LPSTR)(buf+4);
            else
                szAbsUri = (LPSTR)(buf+5);
            while(*szAbsUri != ':')
                szAbsUri++;

           // Skip  Two Forward Slashes
              szAbsUri += 3;

           // Skip Host Name or IP Address
              while(*szAbsUri != '/')
                  szAbsUri++;

           // Copy Absolute path to Buffer

            if(strcmp(wordBuf,"GET")==0)
                strcpy((buf+4), szAbsUri);
            else
                strcpy((buf+5), szAbsUri);
            } // end  of  strCmp(GET or POST)
```

```
    // Extract Host-name/IP-Address from the HOST field
        else{
        if((strcmp(wordBuf,"HOST:"))==0){
              int i = bufIndex-6;
            strcpy(szIPAddress,buf+6);
            szIPAddress[i-2] = '\0';
        }  // end of  else
    }  //  end of  strcmp(GET or POST)
  } // end of if(wordBuf != NULL)

// Rewrite the HTTP Header
   strncpy(lpstConn->chTempBuf+tempBufIndex,buf,strlen(buf));
    tempBufIndex += strlen(buf);

 } // end of while

 lpstConn->chTempBuf[tempBufIndex] = '\0';
 IPAddress = (LPSTR)szIPAddress;

 // Return the Host Name/IP Address
 return(IPAddress);
}  // end of ParseClientRequest()
```

- Perform IP Address Filtering by comparing the extracted Host-name/IPAddress against the list of Denied URL's. If the access to the requested URL is Denied then generate the Connection Refused Message to the client.

```
BOOL FilterIP(LPSTR tempStr)
{
 char str[200];        // buffer to store the requested URL
 int i;

 strcpy(str,tempStr);

// Compare the requested URL against the list of Denied URL's

 for(i=0; i<URLIndex; i++)
    if(strcmp(str,szDeniedUrls[i])==0)
        return(FALSE);

 return(TRUE);
}  // end of  FilterIP()
```

- Call the WinSock Domain Naming Services (DNS) to map the host name to the IP Address. To save the DNS look up time, we are maintaining a data base of previously accessed host-names and their IP Addresses. If the current URL is already present in the database then no DNS lookup is performed.

```
LPSTR DnsLookUp(LPSTR tempstr)
{
    LPHOSTENT  lpHostEntry;      // structure to store  Hostname /IP address
    DWORD       dwIPAddress;     // 32 bit doted address
    LPSTR       szIPAddress;     // pointer to IP address string
    char        str[80];         // temporary buffer


    static char szHostName[10][80];    // array to store the Host Names
    static char  szHostAddr[10][33];   // array to store the IP Addresses
    static int nAddrCount=0;


    strcpy(str,tempstr);

//   convert the string to the 32 bit dotted notation
     dwIPAddress = inet_addr(str);

     szIPAddress = inet_ntoa(*(LPIN_ADDR)&dwIPAddress);

// if the string is host name rather than an IP address  then
// Perform map to the IP Address either from database or by
// calling  DNS   lookup

     if(dwIPAddress == -1) {

// Check the host name into the data base

          int i;
          for(i=0; i<nAddrCount; i++)

      // If found then return the IP Address
               if(strcmp(str,szHostName[i])==0)
                    return((LPSTR)szHostAddr[i]);

//  Call DNS  service to map the host-name to IP Address
          lpHostEntry = gethostbyname(str);
```

```
// If DNS lookup is not successful  then
// Generate message DNS lookup Failed
      if(lpHostEntry == NULL)
            return(NULL);

// otherwise store the host name and IP Address into the database
        else{
            szIPAddress = inet_ntoa(*(LPIN_ADDR)(lpHostEntry->h_addr));
            strcpy(szHostAddr[nAddrCount],szIPAddress);
            strcpy(szHostName[nAddrCount],str);
            if(nAddrCount < 9)
                  nAddrCount++;
            else nAddrCount = 0;
          // Return the IP Address
            return(szIPAddress);
          } // end of else
 } //end of if(dwIPAddress==-1)

 else
    return(szIPAddress);

} // end of DnsLookUp()
```

- Process the Client request. This step involves the creation of Client socket so as to connect to the *HTTP* server on the *Internet*. Here, the proxy acts as client and connects to the external server to fulfil the client request. The external server sees only the address of the machine on which the proxy is running, Hence NAT (Network address Translation) is also Performed at this Step. The proxy server receives the data from the external server and passes on to the internal client till the connection is not closed by the external server.

```
BOOL ConnectRmtServer(HWND hWnd, LPCONNDATA lpstConn,
                                                int nRequestSize)
{
    LPSTR          IPAddress=NULL;     // pointer to the IP Address String
    SOCKET         hSocket;            // socket handle
    SOCKADDR_IN    sockAddr;           // structure to store Remote
                                       // Server IP Address and port no.
    DWORD          dwIPAddress;        // 32 bit address in network order
    int            nRet=0;             // Return code to check errors

 // Create a stream socket as this will act as client Socket so,
 //  no explicit binding is required.
    hSocket = CreateSocket();
```

```
// Convert the IP address string into 32 bit dotted notation
    dwIPAddress = inet_addr(IPAddress);
// Initialise the SOCKADDR structure with the remote server info.
   sockAddr.sin_addr =  *(LPIN_ADDR)&dwIPAddress;
   sockAddr.sin_port = htons(STANDARD_HTTP_PORT);
   sockAddr.sin_family = AF_INET;

// send a Connect request to the remote server
   nRet = connect(hSocket,(LPSOCKADDR)&sockAddr,sizeof(SOCKADDR));

// if connection is refused then report error
   if(nRet == SOCKET_ERROR) {
        MessageBox(GetFocus(),"Connection Timed Out : Server Could
                                Be Down","Proxy Server",
                                MB_OK|MB_ICONINFORMATION);
        return(FALSE);
   } // end of if

// otherwise store the server socket info in the link list node corresponding to
// the client who sent the request
   else
         lpstConn->hServerSocket = hSocket;

// Send the parsed HTTP request header to the external server
   if(hSocket!=INVALID_SOCKET)
         nRet= SendData(hSocket,(LPSTR)lpstConn->chTempBuf,
                                          strlen(lpstConn->chTempBuf));

// Set the FD_READ and FD_CLOSE events on the Server Socket
   nRet=WSAAsyncSelect(hSocket,hWnd,WSA_ASYNC1,FD_READ|FD_CLOSE);

// Report error if any
   if (nRet == SOCKET_ERROR)
       ReportError("Error In Setting Events",WSAGetLastError());

return(TRUE);

} // end of ConnectRemoteServer ()
```

Chapter 4

CONCLUSION & FUTURE SCOPE

This thesis work implemented successfully the following features on 32-bit windows platform:

- Access to popular Internet services like *HTTP, FTP* without assigning registered Internet IP addresses to various stations on the *LAN (Local Area Network)*. Each station is given ***Intranet*** *IP* Addresses and all the users on the LAN can access *HTTP* services through the host on which the proxy server is running. Only the proxy host needs to have an Internet IP address, hence, the problem of assignment of multiple *IP* addresses is solved.

- The software also implemented the external IP address filtering; that is, suppose the institute wants some web sites not to be accessed (may contain some objectionable material); then the IP addresses of those hosts can be given to the proxy server and the data corresponding to those sites will be blocked by the proxy server. In addtion to this, the software also implemented the service filtering which permits the blocking of a particular TCP service like *FTP*.

- The software also records a log file of *URL's* accessed, *HTTP* headers to enable network administrators to take some actions like prohibiting a user who uses Internet excessively for not so obvious purposes.

The software still lacks some desirable features like :

- Individual machines/users can be assigned rights to use different services.

- Support for streamed applications like audio, video.

- Killing *GIF/JPEG* images
- Blocking of Cookie Headers
- Support for *Socks*.

There are proxy servers available in the market; none of which still offers a comprehensive solution. Author purposes to implement the above said features in next version of the software.

Glossary

ARPAnet (Advanced Research Projects Agency Network)

The predecessor to the Internet, officially phased out in 1990. Began in 1969, linking four computer sites (Stanford Research Institute, the University of Utah, the University of California at Los Angeles, and the University of California at Santa Barbara) that were doing research for the U.S. Department of Defense. Initially a computer science experiment designed to provide communications between government agencies, military facilities, defense contractors, and universities, with the goal that the network should be operational even when important parts were unavailable (ie: not vulnerable to "nuclear decapitation"). The first network to use TCP/IP.

ATM (Asynchronous Transfer Mode)

A new, very high speed networking protocol with throughput as high as 1 gigabyte per second. Adoption by the business community has been slow, partly due to high cost and partly to competing standards such as FDDI and Fast Ethernet. ATM has strong appeal for real-time video streaming applications.

Bandwidth

Bandwidth refers to how much data can be transferred over a given network connection in a given time; in other words, how fast people are connecting to the Internet. It's a particularly relevant consideration when designing Web sites, as file sizes, especially multimedia files, can take a long time to download to a browser for vewing.

CERN

The European Laboratory for Particle Physics, located in Geneva, Switzerland, where the basic infrastructure and programming conventions for the World Wide Web were created.

CGI (Common Gateway Interface)

A standard set of protocols that allow a Web server to exchange information with an external "gateway" program such as a Web browser. A CGI script can be written in a variety of languages. Fill-out forms, guestbooks, and counters are some common examples of CGI programs.

Client/Server

Clients are programs that reside on your PC. Programs like Netscape and Explorer are Web clients, Eudora is an e-mail client, etc. These programs interact with the servers to pass information (Web pages, e-mail) back and forth to each other. This relationship is referred to as a client/server relationship.

Cookies

These harmless-sounding "Cookies" are actually coded records that track your movements around the Web. When you visit a site equipped to send you a cookie, and most are, a record of exactly where you went on that site is written in the MagicCookie file on your hard drive by both Netscape Navigator and Internet Explorer. It resides in the preferences folder (Mac) or Windows Dir (Win). These files can be read by most Web servers. Want to know more?

Cyberspace

Coined by William Gibson to describe a computer network that can be directly connected to peoples' minds, the term is now used to represent the Web or Internet.

DNS (Domain Name Server)

A computer that keeps track of names of other machines and their numeric IP addresses. When you refer to a machine by name, the domain name server translates that information into the numeric IP address necessary to make the connection.

Domain Name

All computers on the Internet have two kinds of addresses. One is a numeric address called the IP address which the computers use to talk to each other. The other is a more people-friendly address called the domain name.All Domain names must be registered with InterNIC. The Domain Name is that part of an e-mail address to the right of the "@" sign. For example, Magic Door's domain name is magicdoor.com.

Ethernet

A coaxial cable, local area network first developed at Xerox PARC in 1976. Specified by DEC, Intel and XEROX and now recognised as the industry standard. Data is broken into packets which are transmitted until they arrive at the destination without colliding with any other. A node is either transmitting or receiving at any instant. The bandwidth is about 10 Mbit/s. Disk-Ethernet-Disk transfer rate with TCP/IP is typically 30 kilobyte per second.

FDDI (Fibre Distributed Data Interface)

The high-speed network backbone of choice for corporate America. For years, it was the only standard for 100Mbps LAN networking, and its fault-tolerant and straightforward nature have contributed to its success.

Finger

An Internet software tool for locating people on other Internet sites. Finger is also sometimes used to give access to non-personal information, but the most common use is to see if a person has an account at a particular Internet site. Many sites do not allow incoming Finger requests, but many do.

Firewall

A computer and software configuration that provides site security by filtering incoming and outgoing traffic between the Internet and a secured LAN or Intranet.

FTP (File Transfer Protocol)

A standard method of moving files between two Internet sites. FTP is a special way to login to another Internet site for the purposes of retrieving and/or sending files. There are many Internet sites that have established publicly accessible repositories of material that can be obtained using FTP, by logging in using the account name "anonymous." Thus these sites are called anonymous ftp servers.

GIF (Graphics Interchange Format)

A platform-independent graphics file format developed by CompuServe. It's great advantage on Web pages is that it can be set with a transparent background, giving the illusion of an irregularly shaped image on the page.

Gopher

A popular distributed document retrieval system which started as the Campus Wide Information System at the University of Minnesota. Not supported by the major browsers, and has declined in popularity in favor of FTP-based document retreival.

Host

Any computer on a network that is a repository for services available to other computers on the network. It is quite common to have one host machine provide several services, such as WWW and USENET.

HTML (HyperText Markup Language)

The language of the World Wide Web. Every document on the web is written in HTML, as are all document formatting, clickable hyperlinks, graphical images, multimedia documents, fill-in forms and other web features.

HTTP (HyperText Transfer Protocol)

The simple document transmission standard used to send HTML documents across the World Wide Web.

Hypertext

A term invented by visionary Ted Nelson to describe non-linear writing in which you follow associative paths through a world of textural documents. (ie: What you read when you surf the Web.)

IAP (Internet Access Provider)

A company that provides Internet access to large clients, including ISPs. Such companies feature large, climate controlled and secure computer rooms, with direct access to the major Internet backbone truck supplied primarily by MCI, Sprint, AT&T, UUNet and WorldNet. (Salt Lake companies: Brooks Fiber and Electric Lightwave)

InterNIC

InterNIC Information Services is an initiative that is partially funded by the National Science Foundation to: Provide Internet information services; supervise the registration of Internet addresses and DNS names; assign RFC numbers and; help users make use of, and obtain access to, the Internet

Intranet

Netscape defines an Intanet as an organization-based computer network that uses "standard Internet technologies to deploy a rich, full-function, ubiquitous environment for information sharing, communication, and applications, built on top of open networking technologies and on an open network-based application platform."

IP Number (Internet Protocol Number)

Sometimes called a dotted quad. A unique number consisting of 4 parts separated by dots, (e.g. 205.178.242.2) Every machine on the Internet has a unique IP number - if a machine does not have an IP number, it is not really on the Internet. Most machines also have one or more Domain Names that are easier for people to remember.

ISDN (Integrated Services Digital Network)

A digital subsciber line provided by most local telephone companies, such as US West, that allows internet connection speeds of up to 128 kbs. Special ISDN modems are required.

ISP (Internet Service Provider)

A company that provides businesses and homes with Internet access. Typically, your modem dials a local phone number which is answered by a modem at the ISP location. The call is then routed onto the Internet backbone. Voila, you're connected. ISPs also provide other essential services such as e-mail, domain name registration, web site hosting and technical support.

JPEG (Joint Photgraphic Experts Group)

An ISO/CCITT standard designed for compressing either full-color or grey-scale digital images of "natural," real-world scenes. JPEG uses the JPEG File Interchange Format, or JFIF. The scheme reduces the size of image files by up to 20 times at only slightly reduced quality. Almost all images seen on the Web are either GIFs or JPEGs.

LAN (Local Area Network)

Two or more computers, usually in the same building or geographic area, connected together with network cables.

MIME (Multipurpose Internet Mail Extensions)

A standard for transmitting nontext e-mail message attachments via SMPT. Most proprietary mail systems must translate any received MIME attachments through an SMTP gateway.

NFS (Network File System)

A protocol developed by Sun Microsystems, and defined in RFC 1094, which allows a computer to access files over a network as if they were on its local disks. This protocol has been incorporated in products by more than two hundred companies, and is now a de facto standard. NFS is implemented using a connectionless protocol (UDP) in order to make it stateless.

NNTP (Network News Transfer Protocol)

The protocol used by Internet users to post and retreive messages to and from news servers which host discussions (Newsgroups). Also used by news servers to replicate newsgroup discussions.

PNG (Portable Network Graphics)

A picture format like GIF and JPEG but with better compression and improved clarity according to its supporters. PNG, pronounced "Ping", is endorsed by the World Wide Web Consortium but the standard has yet to receive wide acceptance. Both Netscape and Microsoft say that the next updates to their browsers will read the PNG format without a plug-in.

POP3 (Post ffice Protocol, Versio 3)

An established protocol that lets Internet users send and retreive e-mail to and from mail servers. POP3 provides simple store-and-forward e-mail functionality.

PPP (Point to Point Protocol)

A protocol for TCP/IP routers and PCs to communicate over dial-up and leased-line WAN connections. Establishes a standard way for routers and computers connected over a synchronous or asynchronous WAN link to establish, monitor, and terminate a session, as well as exchange information.

Proxy Services

A software application that is allowed to pass information through a firewall. A firewall is a security system in which one computer, a secure host machine, is the only one in an organization that is actually connected to the Internet. Everyone in the organization must go through the host machine to connect to the Internet, and vice versa.

Router

A router is a hardware device that joins a Local Area Network (LAN) to a Wide Area Network (WAN), often the Internet itself. Routers spend all their time looking at the destination addresses of the packets passing through them and deciding which route to send them on.

SMTP (Simple Mail Transfer Protocol)

Standard protocol that defines how e-mail messages are transferred between servers. SMTP defines only ASCII text content, relying on the MIME standard for nontext attachments.

SNMP (Simple Network Management Protocol)

A set of standards for communication with devices connected to a TCP/IP network. Examples of these devices include routers, hubs, and switches. Devices that are SNMP

compatible contain SNMP "agent" software to receive, send, and act upon SNMP messages.

TCP/IP (Transmission Control Protocol/Internet Protocol)

A packet-based communication protocol that forms the foundation of the Internet.

Telnet

An application developed at the University of Illinois that serves as an access application to other programs running on Internet servers.

Terminal Server

A special purpose computer that has places to plug in many modems on one side, and a connection to a LAN or host machine on the other side. Thus the terminal server does the work of answering the calls and passes the connections on to the appropriate node. Most terminal servers can provide PPP or SLIP services if connected to the Internet.

URL (Uniform Resource Locator)

The addressing standard used to identify the type and location of files on the Internet specifying the type of server, the host name of the computer the file is on, and the complete path to the file. (e.g.: http://www.magicdoor.com)

W3C (World Wide Web Consortium)

An international standards body for the development and endorsement of Web-related technology, such as HTML.

WAN (Wide Area Network)

A data communications network that spans any distance and is usually provided by a public carrier. The user gets access to the two ends of a circuit; the carrier does everything in between--which is typically drawn as a "grey cloud," since you don't know

(or usually care) how the carrier implements it. In contrast, a LAN typically has a diameter (the maximum distance between any two stations) limited to less than a few miles and is entirely owned by the user.

Web Server

The hardware and software used to store and deliver HTML documents for use on the World Wide Web.

References

a) Textbooks (b)

1. *Introduction to Internet*, a course material prepared by Deptt. Of Computer Science, TITI, Chd. , 1997.

2. Douglas E. Comer,.*Internetworking with TCP/IP*, Volume I, PHI

3. Roger Stevens, *TCP/IP Illustrated*, Volume I, Addison Wesley Publishing.

4. Bob Quinn and Dave Shute,*Windows Sockets Network Programming* , Addison Wesley Publishing.

5. Kris Jamsa, Internet Programming, Galgotia publications

6. Andrew S. Tanenbaum, *Computer Networks*, Third Edition, PHI.

7. D.Brent Chapman and Elizabeth Zwicky, *Building Internet Firewalls*, O'Reilly & associates, 1995.

8. Bill Cheswick and Steve Bellovin, *Firewalls and Internet Security: Repelling the Wily Hacker,* Addison Wesley, 1994.

9. Cooper and Hughes, *Implementing Internet Security,* New Rider publishing.

10. Karanjit Siyan, *Internet Firewalls and Network Security*, New Rider Publishing.

11. Jennifer Seberry & Josef Pieprzyk, Cryptography: An Introduction to Computer Security, Prentice Hall.

b) Request For Comments (f)

1. RFC 1244: *Site Security Handbook*, P. Holbook and J. Reynolds, July 1991.

2. RFC 1636: *Report of IAB Workshop on Security in the Internet Architecture*, S. Crocker and R. Braden, June 1994.

3. RFC 1704: *On Internet Authentication*, N. Haller and R. Atkinson, October 1994.

4. RFC 1945: *HyperText Transfer Protocol—HTTP/1.0*, T. Berners-Lee and R. Fielding, May 1996.

c) Web Site Addresses (w)

1. Internet Glossary- http://www.squareonetech.com/glosaryf.html

2. Internet Glossary - http://www.ghofn.org/~tlewis/glossary.html

3. Understanding IP Addresses - http://www.3com.com/nsc/501302s.html

4. Bandwidth Management- http://www.3com.com/nsc/500631.html

5. IP Spoofing - http://www.ionet.net/

6. IP Spoofing - http://home.earthlink.net/~gitan617/ipspoof.html

7. Security Problems in the TCP/IP Protocol suite - http://www.securityServer.com/cgi-local/ssis.pl/docs/tcpip1.html.

8. Internet Firewalls and Security- http://www.3com.com/

9. Cryptography - http://www.club.Innet.be/~year1286/intro_crypt.html

10. HTTP Specifications available via anonymous ftp on //sunsite.unc.edu/

d) Firewall and Security Papers (p)

1. Steve Bellovin, Security Problems in the TCP/IP Protocol Suite, Computer Communication review, Vol. 19, No. 2, April 1989 pages 32-48.

2. Rolf Oppliger, Internet Security: Firewalls and Bey, Communications of ACM, May 1997/ Vol. 40, No. 5.

3. Steven M. Bellovin and William R. Cheswick, Network Firewalls, IEEE Communications Magazine, September 1994.