NEAR EAST UNIVERSITY

INSTITUTE OF APPLIED AND SOCIAL SCIENCES

CACHING WEB PROXY SERVER USING JAVA APPLICATION

Ahmed Abu Asi

MASTER THESIS

DEPARTMENT OF COMPUTER ENGINEERING

Nicosia 2003

NEU **JURY REPORT**

DEPARTMENT OF COMPUTER ENGINEERING

Academic Year: 2003-2004

STUDENT INFORMATION

Full Name	**Ahmed Abu Asi**		
Undergraduate degree	BSc.	**Date Received**	**Spring** 2000
Institute **Institution**	Near East University	**CGPA**	2.00

THESIS

Title	Caching Web Proxy Server Using Java Application
Description	The aim of this thesis is the develop~~ment~~ of client server based software to implement caching web proxy servers based on network security.

Supervisor	Assoc.Prof.Dr.Doğan İbrahim	**Department**	Computer Engineering

~~**JURY'S DECISION**~~ Decision of the Examining Committee

The jury has decided to accept / ~~reject~~ the student's thesis.
The decision was taken unanimously / ~~by majority~~.

Examining Committee Members
~~**JURY MEMBERS**~~

~~**Number Attending**~~	~~4~~	~~**Date**~~	~~15/12/2003~~

~~Name~~, ~~Surname~~	**Signature**
Assoc. Prof. Dr., Rahib Abiyev, Chairman of the jury	[illegible]
Assist. Prof. Dr. Doğan Haktanır, Member	Doğan Haktanır
Assoc. Prof. Dr. Ilham Huseynov, Member	[illegible]
Assoc.Prof.Dr.Doğan İbrahim, Supervisor	

APPROVALS

Date 15/12/2003	[illegible]	**Chairman of Department** Assoc. Prof. Dr. Doğan İbrahim

DEPARTMENT OF COMPUTER ENGINEERING
DEPARTMENTAL DECISION

Date:15/12/2003

Subject: Completion of M.Sc. Thesis

Participants: Assoc. Prof. Dr. Doğan İbrahim, Assoc. Prof. Rahib Abiyev ,Assist.Prof. Dr. Doğan Haktanir, Assoc.Prof. Dr. Ilham Huseynov, Rami Raba, Alaa Eleyan, Mohammed Janjawa, Ibaid Elsoud .

DECISION

We certify that the student whose number and name are given below, has fulfilled all the requirements for a M .S. degree in Computer Engineering.

		CGPA
960468	Ahmed Abu Asi	3

[illegible]

Assoc. Prof. Dr. Rahib Abiyev, Committee Chairman, Computer Engineering Department, NEU

[illegible]

Assist. Prof. Dr. Doğan Haktanir, Committee Member, Electrical and Electronic Engineering Department, NEU

[illegible]

Assoc. Prof. Dr. Ilham Huseynov, Committee Member, Computer Information System Department, NEU

Assoc. Prof. Dr. Doğan Ibrahim, Supervisor, Chairman of Computer Engineering Department, NEU

Chairman of Department
Assoc. Prof. Dr. Doğan İbrahim

Ahmed Abu Asi: Caching Web Proxy Server Using Java Application

Approval of the Graduate School of Applied and Social Sciences

Prof. Dr. Fakhraddin Mamedov
Director

[illegible]

We certify this thesis is satisfactory for the award of the Degree of Master of Science in Computer Engineering

Examining Committee In Charge:

[illegible]

Assoc. Prof. Dr. Rahib Abiyev, Committee Chairman , Computer Engineering Department, NEU

[illegible]

Assist. Prof. Dr. Doğan Haktanir, Committee Member , Electrical and Electronic Engineering Department, NEU

[illegible]

Assoc. Prof. Dr. Ilham Huseynov, Committee Member, Computer Information System Department, NEU

Assoc. Prof. Dr. Doğan İbrahim, Supervisor, Chairman of Computer Engineering Department, NEU

DEDICATION

I would like dedicating this thesis to my parents especially my dear mom and my brother'' Mohamed'' they were the best provider to me during my study life.

ACKNOWLEDGMENT

*I would like to thanks my supervisor **Assoc.Prof.Dr Doğan İbrahim** for his advice and comments on my thesis, and his suggestions that helped to improve this thesis, All errors are of course my own .*

I would also like to thank my teachers especıally Assoc.Prof.Dr.Rahıb Abiyev who supported me and taught me the true meaning of determination.

Finally, I want to thank my friends who support me and for their suggestions during this thesis.

ABSTRACT

The designed Web proxy server is a specialized HTTP server. The primary use of a proxy server is to allow internal clients access to the Internet from behind a firewall. Anyone behind a firewall can now have full Web access past the firewall host with minimum effort and without compromising security.

The proxy server listens for requests from clients within the firewall and forwards these requests to remote internet servers outside the firewall. The proxy server reads responses from the external servers and then sends them to internal client clients.

In the usual case, all the clients within a given subnet use the same proxy server. This makes it possible for the proxy to cache documents efficiently that are requested by a number of clients.

In this thesis I could implement the authentication protocol,caching, provision for the administartor to enforce rules like denying access to some sites and calculation of RTT and effective bandwidth for each request.

The proxy server the proxy includes two applications. The main application starts up and serves and a proxy server that listens to client's requests on a specific port, and forwards the requests to a web server or to another web proxy (father proxy), then sending the replies back to the clients. This application will be referred to as the *proxy* application.

LIST OF GLOSSARY

Access Control List (ACL)

Associated with physical interface the packet came through.

Certificate Authorities

A trusted third-party organization or company that issues digital certificates used to create digital signatures and public-private key pairs. The role of the CA in this process is to guarantee that the individual granted the unique certificate is, in fact, who he or she claims to be.

UUCP

UUCP (Unix-to-Unix CoPy) was originally developed to connect Unix (surprise!) hosts together. UUCP has since been ported to many different architectures, including PCs, Macs, Amigas, Apple IIs, VMS hosts, everything else you can name, and even some things you can't. Additionally, a number of systems have been developed around the same principles as UUCP.

Content Distribution Network "CDN"

The physical network infrastructure of connecting global locations together which allows for Web content to be distributed to the edges of the entire network infrastructure.

Cryptographic algorithms

A cryptographic system that uses two keys - a public key known to everyone and a private or secret key known only to the recipient of the message. An important element to the public key system is that the public and private keys are related in such a way that only the public key can be used to encrypt messages and only the corresponding private key can be used to decrypt them. Moreover, it is virtually impossible to deduce the private key if you know the public key.

Decrypt

The process of decoding data has been encrypted into a secret format. Decryption requires a secret key or password.

Digital Certificate

An attachment to an electronic message used for security purposes. The most common use of a digital certificate is to verify that a user sending a message is who he or she claims to be, and to provide the receiver with the means to encode a reply.

An individual wishing to send an encrypted message applies for a digital certificate from a Certificate Authority.

Encrypt

The translation of data a secret code. Encryption is the most effective way to achieve data security.

Unencrypted data is called plain text encrypted data is referred to as cipher text.

DES

Data Encryption Standard, an encryption algorithm used by the U.S. Government.

DSA

Digital Signature Algorithm, part of the digital authentication standard used by the U.S. Government.

KEA

Key Exchange Algorithm, an algorithm used for key exchange by the U.S. Government.

MD5

Message Digest algorithm developed by Rivest.

RC2 and RC4

Rivest encryption ciphers developed for RSA Data Security.

RSA key exchange

A key-exchange algorithm for SSL based on the RSA algorithm.

SHA-1

Secure Hash Algorithm, a hash function used by the U.S. Government.

SKIPJACK

A classified symmetric-key algorithm implemented in FORTEZZA-compliant hardware used by the U.S. Government. (For more information, see FORTEZZA Cipher Suites.)

Triple-DES

DES applied three times.

HTTP

Short for Hypertext Transfer Protocol, the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

HTTPS

By convention, Web pages that require an SSL connection start with https: instead of http.

IP

Abbreviation of Internet Protocol, pronounced as two separate letters. IP specifies the format of packets, also called datagrams, and the addressing scheme. Most networks combine IP with a higher-level protocol called Transport Control Protocol (TCP) that establishes a virtual connection between a destination and a source.

IPSec

A secure network starts with a strong security policy that defines the freedom of access to information and dictates the deployment of security in the network.

ISP

Short for Internet Service Provider. A company that provides connection and services on the Internet, such as remote dial-in access, DSL connections and Web hosting services.

ISO

The International Standards Organization (ISO) Open Systems Interconnect (OSI) Reference Model defines seven layers of communications types

OSI

Short for Open System Interconnection, an ISO standard for worldwide communications that defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer in one station, proceeding to the bottom layer, over the channel to the next station and back up the hierarchy.

PIN

Short for Personal Identification Number. Typically PIN's are assigned by financial institutes to validate the identity of a person during a transaction.

RSA

An public key encryption technology developed by RSA Data Security, Inc. The acronym stands for Rivest, Shamir, and Adelman, the inventors of the technique. The RSA algorithm is based on the fact that there is no efficient way to factor very large numbers. Deducing an RSA key, therefore, requires an extraordinary amount of computer processing power and time.

SSL

Short for Secure Sockets Layer, a protocol developed by Netscape for transmitting private documents via the Internet. SSL works by using a private key to encrypt data that's transferred over the SSL connection.

SSL performs a negotiation between the two parties who are exchanging information, the negotiation process involves understanding the key pairs, the protocols, and the type of data request.

TCP

Abbreviation of Transmission Control Protocol, and pronounced as separate letters. TCP is one of the main protocols in TCP/IP networks. Whereas the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data.

UDP

Short for User Datagram Protocol, a connectionless protocol that, like TCP, runs on top of IP networks.

VPNs

(Virtual Private Networks). Traditionally, for an organization to provide connectivity between a main office and a satellite one, an expensive data line had to be leased in order to provide direct connectivity between the two offices.

Web

A system of Internet servers that support specially formatted documents. The documents are formatted in with HTML (Hypertext Mark up Language) that supports links to other documents, as well as graphics, audio, and video files.

Web Server Accelerators

A system that services Web servers by offloading TCP/IP connections, responding to Web Client requests, replicating the Web content for availability and surge protection, and enhancing Web server performance.

Web Sever

A computer that delivers (serves up) is Web pages. Every Web server has an IP address and possibly a domain name.

TABLE OF CONTENTS

INTRODUCTION

Network security involves any and all countermeasures taken to protect a network from threats to its integrity. As modem networks have continued to grow and as more and more networks have been connected to the public Internet, the threats to the integrity and privacy of a company's networks have also grown. The attacks that are made on a network are increasingly more complex and pervasive, and the tools used for such purposes are easy to acquire. For example, anyone can log on to an Internet search engine and perform a search on hacking and be presented with an immense amount of sites that offer information and tools on hacking. Therefore, the need for network security is obvious.

The main reason for using a proxy server is to give access to the Internet from within a firewall. An application-level proxy makes a firewall safely permeable for users in an organization, without creating a potential security hole through which one might get into the subnet. The proxy can control services for individual methods, host and domain, and more-filtering client transactions. A very important thing about proxies is that even a client without DNS can use the Web It needs only the IP address of the proxy. Application level proxy facilitates caching at the proxy. Usually, one proxy server is used by all clients connected to a subnet. This is why the proxy is able to do efficient caching of documents that are requested by more than one client. The fact that proxies can provide efficient caching makes them useful even when no firewall machine is in order. Configuring a group to use a caching proxy server is easy (Most popular Web client programs already have proxy support built in), and can decrease network traffic costs significantly, because once the first request was made for a certain document, the next ones are retrieved from a local cache.

The aim of the thesis is the development of java application software for caching web proxy servers based on network security. The java application based program has been developed by the author, which applies caching proxy based network security.

Chapter 1 In this chapter we'll cover some of the foundations of computer networking and network security,

Chapter 2 This chapter briefly describes what Internet firewalls can do for your overall site security.

Chapter 3 Describes The network traffic proxy servers security problems due to the repeated retrieving of objects from remote Web servers on the Internet.

Chapter 4 This chapter describes the Public-key cryptography and related standards and techniques underlie security features of authentication and authorization. -key cryptography.

Chapter 5 Describes the designed and implemented a web proxy server, which can be configured remotely over the Internet by its administrator.

Finally, the program developed by the author is given in an appendix at the end of the thesis.

1. NETWORK SECURITY

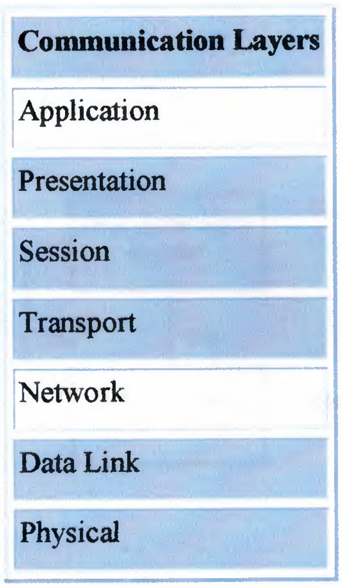
1.1. Overview

Network security is a complicated subject, historically only tackled by well-trained and experienced experts. However, as more and more people become ``wired", an increasing number of people need to understand the basics of security in a networked world. This chapter was written with the basic computer user and information systems manager in mind, explaining the concepts needed to read through the hype in the marketplace and understand risks and how to deal with them. Some history of networking is included, as well as an introduction to TCP/IP and internetworking . We go on to consider risk management, network threats, firewalls, and more special-purpose secure networking devices. This is not intended to be a ``frequently asked questions" reference, nor is it a ``hands-on" document describing how to accomplish specific functionality. It is hoped that the reader will have a wider perspective on security in general, and better understand how to reduce and manage risk personally, at home, and in the workplace.

1.2. Security Policy

As was mentioned above firewall must inspect all the packets that come to and leave the Local Network and filter out those packets that do not conform to the Security Policy adopted for the Local Network.

Remember the ISO seven layers protocol model. The packet inspection can take place on any of the layers. But packet inspection is most commonly implemented at Application layer by Application layer firewalls and at Network layer by Network layer firewalls. When talking about TCP/IP protocol suite the Application layer firewalls are commonly called Application Gateways or Proxies (further Proxies) and Network layer firewalls Filtering Routers or Screening Routers (further Filtering Routers).



Communication Layers
Application
Presentation
Session
Transport
Network
Data Link
Physical

Figure 1.1 The ISO/OSI Reference Model layers

1.2.1. How packets are filtered out?

The function of ordinary IP router is receives IP datagram extracts destination IP address and consults the routing table for next hop for this data gram. As its name indicates Filtering Router in addition to routing function performs a filtering of the packets it receives, that is before consulting the routing table it must decide whether this packet should be forwarded towards its destination.
The filtering decision is made according to the Access Control List (further ACL) associated with physical interface the packet came through.

ACL consists of the entries. Each entry specifies values for particular header fields and action to be taken if arrived packet matches these values.

Each arrived packet is matched successively against the entries in the ACL if the match occurs the action is taken. [1]

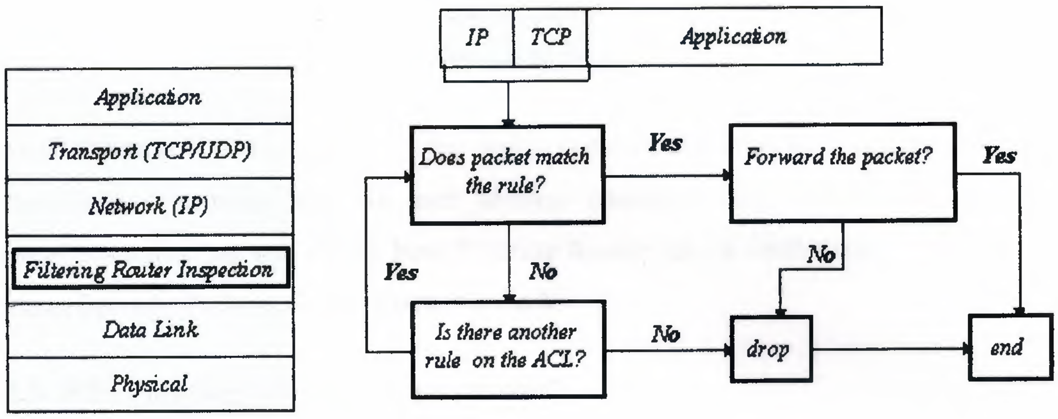


Figure 1.2 Steps of filtering decision (that which is not expressively permitted is prohibited approach).

The question arises : "What is done with the packet that does not match any entry in the ACL?".

In this situation two different approaches may be adopted:

1. That which is not expressively permitted is prohibited, that is these packets will be dropped by the Filtering Router.
2. That which is not expressively prohibited is permitted, that is these packets will forwarded by the Filtering Router.

1.2.2. What information is used for filtering decision?

The portions that are parsed by the filtering router are IP header, and transport protocol header whether TCP or UDP. Therefore the header fields that can be used in ACL entries are:

- Source IP address (IP header)
- Destination IP address (IP header)
- Protocol Type (IP header, specifies whether the data encapsulated in the IP datagram belongs to TCP, UDP or ICMP protocol)
- Source port (TCP or UDP header)

- Destination port (TCP or UDP header)
- ACK. bit (TCP header, this bit specifies whether the packet is the acknowledgment for received TCP packet)

Not all filtering routers support all the fields listed above. Moreover not all filtering routers support separate ACL for each physical interface (port). Bellow you can find several examples that will clarify how Filtering Router can be configured to implement various Security Policies for the Local Network.

1.2.3. Why Filtering Routers are not enough?

Protocols requiring setup of inbound connections Problem arises when dealing with protocols that require connection setup to ports not known in advance on hosts on the Local Network. Connection (client_port, client_IP_address, 21, server_IP_address) is setup by FTP client and is used for control flow Connection (20, server_IP_address, client_port, client_IP_address) is setup by FTP server and is used for file transfer. For Filtering Router to allow outbound (from Local Network to the Internet) FTP traffic it must permit connections made from port number 20 to any port on a host on Local Network, thus allowing outsiders access to any services running on any host on the Local Network. You can ask: Why access to a server can pose any security problem? First of all there are services that you do not want the outsider to access. But even if the access to the server does not pose any problem, a server software is a complicated one and as such contains many bugs which can be exploited to compromise the Local Network. It is better not to allow outsiders to know your network topology. Such knowledge can help compromise the Local Network. By inspecting the packets leaving Local Network many things can be learned about it topology. Even more can be learned through the access to your DNS server.

1.3. What is a Network?

A ``network" has been defined as ``any set of interlinking lines resembling a net, a network of roads an interconnected system, a network of alliances." This definition suits our purpose well: a computer network is simply a system of interconnected computers.

How they're connected is irrelevant, and as we'll soon see, there are a number of ways to do this.

1.4. Some Popular Networks?

Over the last 25 years or so, a number of networks and network protocols have been defined and used. We're going to look at two of these networks, both of which are ``public" networks. Anyone can connect to either of these networks, or they can use types of networks to connect their own hosts (computers) together, without connecting to the public networks. Each type takes a very different approach to providing network services.

1.4.1. UUCP

UUCP (Unix-to-Unix CoPy) was originally developed to connect Unix (surprise!) hosts together. UUCP has since been ported to many different architectures, including PCs, Macs, Amigas, Apple IIs, VMS hosts, everything else you can name, and even some things you can't. Additionally, a number of systems have been developed around the same principles as UUCP.

1.4.1.1. Batch-Oriented Processing

UUCP and similar systems are batch-oriented systems: everything that they have to do is added to a queue, and then at some specified time, everything in the queue is processed.

1.4.1.2. Implementation Environment

UUCP networks are commonly built using dial-up (modem) connections. This doesn't have to be the case though: UUCP can be used over any sort of connection between two computers, including an Internet connection.

Building a UUCP network is a simple matter of configuring two hosts to recognize each other, and know how to get in touch with each other. Adding on to the network is simple. if hosts called A and B have a UUCP network between them, and C would like to join the network, then it must be configured to talk to A and/or B. Naturally, anything that C talks to must be made aware of C's existence before any connections will work. Now, to connect D to the network, a connection must be established with at least one of the hosts on the network, and so on. Figure 1.3 shows a sample UUCP network.

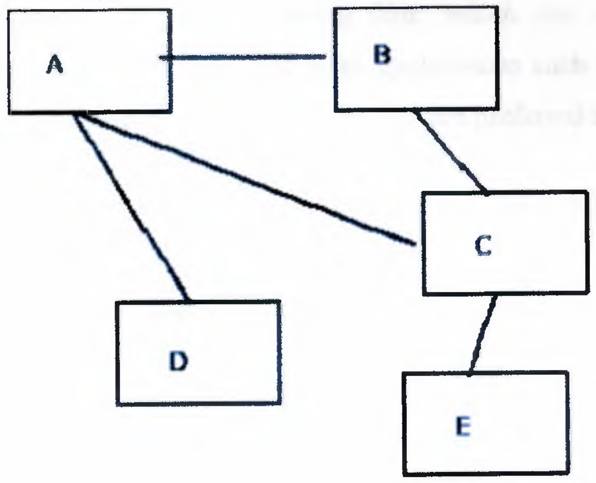


Figure 1.3 A Sample UUCP Network

In a UUCP network, users are identified in the format host!userid. The ``!'' character (pronounced ``bang'' in networking circles) is used to separate hosts and users. A bangpath is a string of host(s) and a userid like A!cmcurtin or C!B!A!cmcurtin. If I am a user on host A and you are a user on host E, I might be known as A!cmcurtin and you as E!you. Because there is no direct link between your host (E) and mine (A), in order for us to communicate, we need to do so through a host (or hosts!) that has connectivity to both E and A. In our sample network, C has the connectivity we need. So, to send me a file, or piece of email, you would address it to C!A!cmcurtin. Or, if you feel like taking the long way around, you can address me as C!B!A!cmcurtin.

The ``public'' UUCP network is simply a huge worldwide network of hosts connected to each other.

1.4.1.3 Popularity

The public UUCP network has been shrinking in size over the years, with the rise of the availability of inexpensive Internet connections. Additionally, since UUCP connections are typically made hourly, daily, or weekly, there is a fair bit of delay in getting data from one user on a UUCP network to a user on the other end of the network. UUCP isn't very flexible, as it's used for simply copying files (which can be netnews, email, documents, etc.) Interactive protocols (that make applications such as the World Wide Web possible) have become much more the norm, and are preferred in most cases.

However, there are still many people whose needs for email and netnews are served quite well by UUCP, and its integration into the Internet has greatly reduced the amount of cumbersome addressing that had to be accomplished in times past.

1.4.1.3. Security

UUCP, like any other application, has security tradeoffs. Some strong points for its security is that it is fairly limited in what it can do, and it's therefore more difficult to trick into doing something it shouldn't. it's been around a long time, and most its bugs have been discovered, analyzed, and fixed. and because UUCP networks are made up of occasional connections to other hosts, it isn't possible for someone on host E to directly make contact with host B, and take advantage of that connection to do something naughty.[4]

On the other hand, UUCP typically works by having a system-wide UUCP user account and password. Any system that has a UUCP connection with another must know the appropriate password for the uucp or nuucp account. Identifying a host beyond that point has traditionally been little more than a matter of trusting that the host is who it claims to be, and that a connection is allowed at that time. More recently, there has been an additional layer of authentication, whereby both hosts must have the same sequence number , that is a number that is incremented each time a connection is made.

Hence, if I run host B, I know the **uucp** password on host A. If, though, I want to impersonate host C, I'll need to connect, identify myself as C, hope that I've done so at a time that A will allow it, and try to guess the correct sequence number for the session. While this might not be a trivial attack, it isn't considered very secure.

1.5. The Internet

This is a word that we've heard way too often in the last few years. Movies, books, newspapers, magazines, television programs, and practically every other sort of media imaginable have dealt with the Internet recently.

1.5.1. What is the Internet?

The Internet is the world's largest network of networks. When you want to access the resources offered by the Internet, you don't really connect to the Internet. you connect to a network that is eventually connected to the Internet backbone, a network of extremely fast (and incredibly overloaded!) network components. This is an important point: the Internet is a network of networks - not a network of hosts. A simple network can be constructed using the same protocols and such that the Internet uses without actually connecting it to anything else. I might be allowed to put one of my hosts on one of my employer's networks. We have a number of networks, which are all connected together on a backbone that is a network of our networks. Our backbone is then connected to other networks, one of which is to an Internet Service Provider (ISP) whose backbone is connected to other networks, one of which is the Internet backbone. If you have a connection ``to the Internet" through a local ISP, you are actually connecting your computer to one of their networks, which is connected to another, and so on. To use a service from my host, such as a web server, you would tell your web browser to connect to my host. Underlying services and protocols would send packets (small datagrams) with your query to your ISP's network, and then a network they're connected to, and so on, until it found a path to my employer's backbone, and to the exact network my host is on. My host would then respond appropriately, and the same would happen in reverse: packets would traverse all of the connections until they found their way back to your computer, the network shown in Figure 1.4 is designated ``LAN 1" and shown in the bottom-right of the picture. This shows how the hosts on that network are provided connectivity to other hosts on the same LAN, within the same company, outside of the company, but in the same ISP cloud , and then from another ISP somewhere on the Internet.

The Internet is made up of a wide variety of hosts, from supercomputers to personal computers, including every imaginable type of hardware and software. How do all of these computers understand each other and work together?

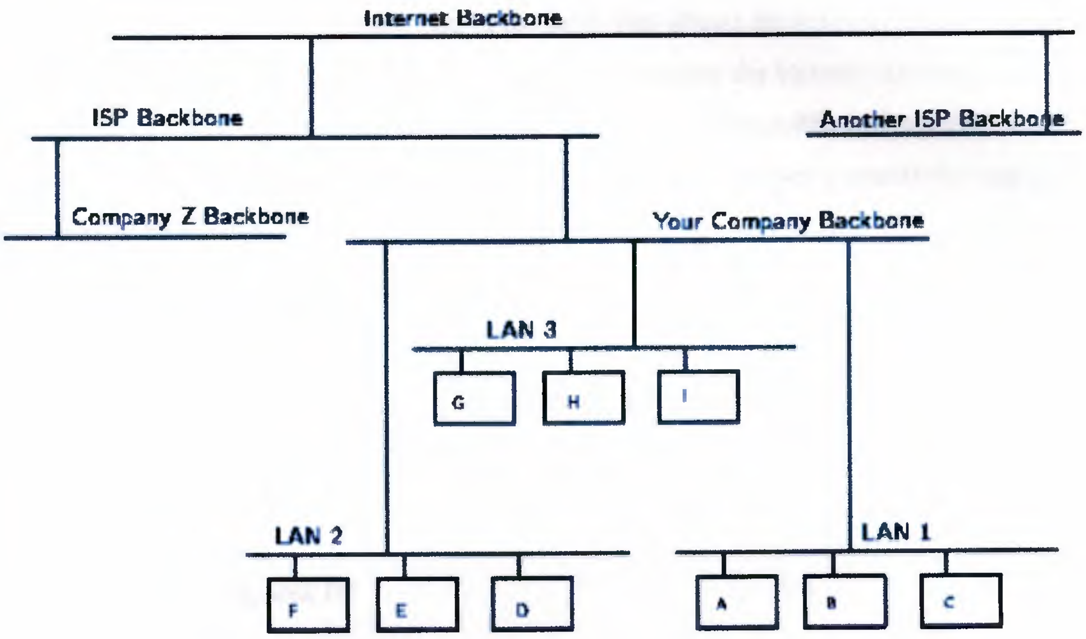


Figure 1.4 A Wider View of Internet-connected Networks

1.6. TCP/IP The Language of the Internet

TCP/IP (Transport Control Protocol/Internet Protocol) is the ``language" of the Internet. Anything that can learn to ``speak TCP/IP" can play on the Internet. This is functionality that occurs at the Network (IP) and Transport (TCP) layers in the ISO/OSI Reference Model. Consequently, a host that has TCP/IP functionality (such as Unix, OS/2, MacOS, or Windows NT) can easily support applications (such as Netscape's Navigator) that uses the network. [3]

1.6.1. Open Design

One of the most important features of TCP/IP isn't a technological one: The protocol is an ``open" protocol, and anyone who wishes to implement it may do so freely.

Engineers and scientists from all over the world participate in the IETF (Internet Engineering Task Force) working groups that design the protocols that make the Internet work. Their companies typically donate their time, and the result is work that benefits everyone.

1.6.2 IP

IP is a ``network layer" protocol. This is the layer that allows the hosts to actually ``talk" to each other. Such things as carrying datagrams, mapping the Internet address (such as 10.2.3.4) to a physical network address (such as 08:00:69:0a:ca:8f), and routing, which takes care of making sure that all of the devices that have Internet connectivity can find the way to each other.

1.6.2.1. Understanding IP

IP has a number of very important features that make it an extremely robust and flexible protocol. For our purposes, though, we're going to focus on the security of IP, or more specifically, the lack thereof.

1.6.2.2. Attacks Against IP

A number of attacks against IP are possible. Typically, these exploits the fact that IP does not perform a robust mechanism for authentication, which is proving that a packet came from where it claims it, did. A packet simply claims to originate from a given address, and there isn't a way to be sure that the host that sent the packet is telling the truth. This isn't necessarily a weakness, per se, but it is an important point, because it means that the facility of host authentication has to be provided at a higher layer on the ISO/OSI Reference Model. Today, applications that require strong host authentication (such as cryptographic applications) do this at the application layer.

1.6.2.3. IP Spoofing

This is where one host claims to have the IP address of another. Since many systems (such as router access control lists) define which packets may and which packets may not pass based on the sender's IP address, this is a useful technique to an attacker: he can send packets to a host, perhaps causing it to take some sort of action.

Additionally, some applications allow login based on the IP address of the person making the request (such as the Berkeley r-commands)[2]. These are both good examples how trusting untrustable layers can provide security .

1.6.2.4. IP Session Hijacking

This is a relatively sophisticated attack, first described by Steve Bellovin [3]. This is very dangerous, however, because there are now toolkits available in the underground community that allow otherwise unskilled bad-guy-wannabes to perpetrate this attack. IP Session Hijacking is an attack whereby a user's session is taken over, being in the control of the attacker. If the user was in the middle of email, the attacker is looking at the email, and then can execute any commands he wishes as the attacked user. The attacked user simply sees his session dropped, and may simply login again, perhaps not even noticing that the attacker is still logged in and doing things.

For the description of the attack, let's return to our large network of networks in Figure 1.4. In this attack, a user on host A is carrying on a session with host G. Perhaps this is a telnet session, where the user is reading his email, or using a Unix shell account from home. Somewhere in the network between A and B sits host H which is run by a naughty person. The naughty person on host H watches the traffic between A and G, and runs a tool which starts to impersonate A to G, and at the same time tells A to shut up, perhaps trying to convince it that G is no longer on the net (which might happen in the event of a crash, or major network outage). After a few seconds of this, if the attack is successful, naughty person has ``hijacked'' the session of our user. Anything that the user can do legitimately can now be done by the attacker, illegitimately. As far as G knows, nothing has happened.

This can be solved by replacing standard telnet-type applications with encrypted versions of the same thing. In this case, the attacker can still take over the session, but he'll see only ``gibberish'' because the session is encrypted. The attacker will not have the needed cryptographic key(s) to decrypt the data stream from G, and will, therefore, be unable to do anything with the session.

1.6.3. TCP

TCP is a transport-layer protocol. It needs to sit on top of a network-layer protocol, and was designed to ride atop IP. (Just as IP was designed to carry, among other things, TCP packets.) Because TCP and IP were designed together and wherever you have one, you typically have the other, the entire suites of Internet protocols are known collectively as ``TCP/IP." TCP itself has a number of important features that we'll cover briefly.

1.6.3.1. Guaranteed Packet Delivery

Probably the most important is guaranteed packet delivery. Host A sending packets to host B expects to get acknowledgments back for each packet. If B does not send an acknowledgment within a specified amount of time, A will resend the packet. Applications on host B will expect a data stream from a TCP session to be complete, and in order. As noted, if a packet is missing, it will be resent by A, and if packets arrive out of order, B will arrange them in proper order before passing the data to the requesting application. This is suited well toward a number of applications, such as a telnet session. A user wants to be sure every keystroke is received by the remote host, and that it gets every packet sent back, even if this means occasional slight delays in responsiveness while a lost packet is resent, or while out-of-order packets are rearranged. It is not suited well toward other applications, such as streaming audio or video, however. In these, it doesn't really matter if a packet is lost (a lost packet in a stream of 100 won't be distinguishable) but it does matter if they arrive late (i.e., because of a host resending a packet presumed lost), since the data stream will be paused while the lost packet is being resent. Once the lost packet is received, it will be put in the proper slot in the data stream, and then passed up to the application.

1.6.4. UDP

UDP (User Datagram Protocol) is a simple transport-layer protocol. It does not provide the same features as TCP, and is thus considered ``unreliable." Again, although this is unsuitable for some applications, it does have much more applicability in other applications than the more reliable and robust TCP.

1.6.4.1. Lower Overhead than TCP

One of the things that make UDP nice is its simplicity. Because it doesn't need to keep track of the sequence of packets, whether they ever made it to their destination, etc., it has lower overhead than TCP. This is another reason why it's more suited to streaming-data applications: there's less screwing around that needs to be done with making sure all the packets are there, in the right order, and that sort of thing.

1.7. Types And Sources Of Network Threats

Now, we've covered enough background information on networking that we can actually get into the security aspects of all of this. First of all, we'll get into the types of threats there are against networked computers, and then some things that can be done to protect you against various threats.

1.7.1. Denial-of-Service

DoS (Denial-of-Service) attacks are probably the nastiest, and most difficult to address. These are the nastiest, because they're very easy to launch, difficult (sometimes impossible) to track, and it isn't easy to refuse the requests of the attacker, without also refusing legitimate requests for service.

The premise of a DoS attack is simple: send more requests to the machine than it can handle. There are toolkits available in the underground community that make this a simple matter of running a program and telling it which host to blast with requests. The attacker's program simply makes a connection on some service port, perhaps forging the packet's header information that says where the packet came from, and then dropping the connection. If the host is able to answer 20 requests per second, and the attacker is sending 50 per second, obviously the host will be unable to service all of the attacker's requests, much less any legitimate requests (hits on the web site running there, for example). Such attacks were fairly common in late 1996 and early 1997, but are now becoming less popular. Some things that can be done to reduce the risk of being stung by a denial of service attack include

- Not running your visible-to-the-world servers at a level too close to capacity

- Using packet filtering to prevent obviously forged packets from entering into your network address space.
- Keeping up-to-date on security-related patches for your hosts' operating systems.

1.7.2. Unauthorized Access

Unauthorized access is a very high-level term that can refer to a number of different sorts of attacks. The goal of these attacks is to access some resource that your machine should not provide the attacker. For example, a host might be a web server, and should provide anyone with requested web pages. However, that host should not provide command shell access without being sure that the person making such a request is someone who should get it, such as a local administrator.

1.7.3. Executing Commands Illicitly

It's obviously undesirable for an unknown and untrusted person to be able to execute commands on your server machines. There are two main classifications of the severity of this problem: normal user access, and administrator access. A normal user can do a number of things on a system (such as read files, mail them to other people, etc.) that an attacker should not be able to do. This might, then, be all the access that an attacker needs. On the other hand, an attacker might wish to make configuration changes to a host (perhaps changing its IP address, putting a start-up script in place to cause the machine to shut down every time it's started, or something similar). In this case, the attacker will need to gain administrator privileges on the host.

1.7.4. Confidentiality Breaches

We need to examine the threat model: what is it that you're trying to protect yourself against? There is certain information that could be quite damaging if it fell into the hands of a competitor, an enemy, or the public. In these cases, it's possible that compromise of a normal user's account on the machine can be enough to cause damage (perhaps in the form of PR, or obtaining information that can be used against the company, etc.)

While many of the perpetrators of these sorts of break-ins are merely thrill-seekers interested in nothing more than to see a shell prompt for your computer on their screen, there are those who are more malicious, as we'll consider next. (Additionally, keep in mind that it's possible that someone who is normally interested in nothing more than the thrill could be persuaded to do more: perhaps an unscrupulous competitor is willing to hire such a person to hurt you.)

1.7.5. Destructive Behavior

Among the destructive sorts of break-ins and attacks, there are two major categories:

1.7.5.1. Data Diddling

The data diddling is likely the worst sort, since the fact of a break-in might not be immediately obvious. Perhaps he's toying with the numbers in your spreadsheets, or changing the dates in your projections and plans. Maybe he's changing the account numbers for the auto-deposit of certain paychecks. In any case, rare is the case when you'll come in to work one day, and simply know that something is wrong. An accounting procedure might turn up a discrepancy in the books three or four months after the fact. Trying to track the problem down will certainly be difficult, and once that problem is discovered, how can any of your numbers from that time period be trusted? How far back do you have to go before you think that your data is safe?

1.7.5.2. Data Destruction

Some of those perpetrate attacks are simply twisted jerks who like to delete things. In these cases, the impact on your computing capability - and consequently your business - can be nothing less than if a fire or other disaster caused your computing equipment to be completely destroyed.

1.8. Secure Network Devices

It's important to remember that the firewall only one entry point to your network. Modems, if you allow them to answer incoming calls, can provide an easy means for an attacker to sneak around (rather than through) your front door (or, firewall). Just as castles weren't built with moats only in the front, your network needs to be protected at all of its entry points.

1.8.1. Secure Modems and Dial-Back Systems

If modem access is to be provided, this should be guarded carefully. The terminal server, or network device that provides dial-up access to your network needs to be actively administered, and its logs need to be examined for strange behavior. Its password need to be strong - not ones that can be guessed. Accounts that aren't actively used should be disabled. In short, it's the easiest way to get into your network from remote: guard it carefully. There are some remote access systems that have the feature of a two-part procedure to establish a connection. The first part is the remote user dialing into the system, and providing the correct userid and password. The system will then drop the connection, and call the authenticated user back at a known telephone number. Once the remote user's system answers that call, the connection is established, and the user is on the network. This works well for folks working at home, but can be problematic for users wishing to dial in from hotel rooms and such when on business trips. Other possibilities include one-time password schemes, where the user enters his userid, and is presented with a ``challenge,'' a string of between six and eight numbers. He types this challenge into a small device that he carries with him that looks like a calculator. He then presses enter, and a ``response'' is displayed on the LCD screen. The user types the response, and if all is correct, he login will proceed. These are useful devices for solving the problem of good passwords, without requiring dial-back access. However, these have their own problems, as they require the user to carry them, and they must be tracked, much like building and office keys. No doubt many other schemes exist. Take a look at your options, and find out how what the vendors have to offer will help you enforce your security policy effectively.

1.8.2. Crypto-Capable Routers

A feature that is being built into some routers is the ability to session encryption between specified routers. Because traffic traveling across the Internet can be seen by people in the middle who have the resources (and time) to snoop around, these are advantageous for providing connectivity between two sites, such that there can be secure routes.

1.9. Network Security Filters and Firewalls

This article is a general introduction to network security issues and solutions in the Internet. emphasis is placed on route filters and firewalls. It is not intended as a guide to setting up a secure network. its purpose is merely as an overview. Some knowledge of IP networking is assumed, although not crucial. In the last decade, the number of computers in use has exploded. For quite some time now, computers have been a crucial element in how we entertain and educate ourselves, and most importantly, how we do business. It seems obvious in retrospect that a natural result of the explosive growth in computer use would be an even more explosive (although delayed) growth in the desire and need for computers to talk with each other. The growth of this industry has been driven by two separate forces which until recently have had different goals and end products. The first factor has been research interests and laboratories. these groups have always needed to share files, email and other information across wide areas. The research labs developed several protocols and methods for this data transfer, most notably TCP/IP. Business interests are the second factor in network growth. For quite some time, businesses were primarily interested in sharing data within an office or campus environment, this led to the development of various protocols suited specifically to this task. Within the last five years, businesses have begun to need to share data across wide areas. This has prompted efforts to convert principally LAN-based protocols into WAN-friendly protocols. The result has spawned an entire industry of consultants who know how to manipulate routers, gateways and networks to force principally broadcast protocols across point-to-point links (two very different methods of transmitting packets across networks). Recently (within the last 2 or 3 years) more and more companies have realized that they need to settle on a common networking protocol. Frequently the protocol of choice has been TCP/IP, which is also the primary protocol run on the Internet. The emerging ubiquitousness of TCP/IP allows companies to interconnect with each other via private networks as well as through public networks.

This is a very rosy picture: businesses, governments and individuals communicating with each other across the world. While reality is rapidly approaching this utopian picture, several relatively minor issues have changed status from low priority to extreme importance. Security is probably the most well known of these problems. When businesses send private information across the net, they place a high value on it getting to its destination intact and without being intercepted by someone other than the intended recipient. Individuals sending private communications obviously desire secure communications. Finally, connecting a system to a network can open the system itself up to attacks. If a system is compromised, the risk of data loss is high.

It can be useful to break network security into two general classes:

- Methods used to secure data as it transits a network
- Methods which regulate what packets may transit the network

While both significantly effect the traffic going to and from a site, their objectives are quite different.

1.10. Transit Security

Currently, there are no systems in wide use that will keep data secure as it transits a public network. Several methods are available to encrypt traffic between a few coordinated sites. Unfortunately, none of the current solutions scale particularly well. Two general approaches dominate this area:

- **Virtual Private Networks:**

  This is the concept of creating a private network by using TCP/IP to provide the lower levels of a second TCP/IP stack. This can be a confusing concept, and is best understood by comparing it to the way TCP/IP is normally implemented. In a nutshell, IP traffic is sent across various forms of physical networks. Each system that connects to the physical network implements a standard for sending IP messages across that link. Standards for IP transmission across various types of links exist, the most common are for Ethernet and Point to Point links (PPP and SLIP). Once an IP packet is received, it is passed up to higher layers of the TCP/IP stack as appropriate (UDP, TCP and eventually the application). When a virtual private network is implemented, the lowest levels of the TCP/IP protocol are implemented using an existing TCP/IP connection. There are a number of ways to accomplish this which tradeoff between abstraction and efficiency. The advantage this gives you in terms of secure data transfer is only a single step further away. Because a VPN gives you complete control over the physical layer, it is entirely within the network designers power to encrypt the connection at the physical (virtual) layer. By doing this, all traffic of any sort over the VPN will be encrypted, whether it be at the application layer (such as Mail or News) or at the lowest layers of the stack (IP, ICMP). The primary

- **advantages of VPNs are**

they allow private address space (you can have more machines on a network), and they allow the packet encryption/translation overhead to be done on dedicated systems, decreasing the load placed on production machines.

Packet Level Encryption: Another approach is to encrypt traffic at a higher layer in the TCP/IP stack. Several methods exist for the secure authentication and encryption of telnet and rlogin sessions (Kerberos, S/Key and DESlogin) which are examples of encryption at the highest level of the stack (the application layer). The advantages to encrypting traffic at the higher layer are that the processor overhead of dealing with a VPN is eliminated, inter-operability with current applications is not affected, and it is much easier to compile a client program that supports application layer encryption than to build a VPN. It is possible to encrypt traffic at essentially any of the layers in the IP stack. Particularly promising is encryption that is done at the TCP level which provides fairly transparent encryption to most network applications. It is important to note that both of these methods can have performance impacts on the hosts that implement the protocols, and on the networks which connect those hosts. The relatively simple act of encapsulating or converting a packet into a new form requires CPU-time and uses additional network capacity. Encryption can be a very CPU-intensive process and encrypted packets may need to be padded to uniform length to guarantee the robustness of some algorithms. Further, both methods have impacts on other areas (security related and otherwise- such as address allocation, fault tolerance and load balancing) that need to be considered before any choice is made as to which is best for a particular case.

1.11. Traffic Regulation

The most common form of network security on the Internet today is to closely regulate which types of packets can move between networks. If a packet which may do something malicious to a remote host never gets there, the remote host will be unaffected. Traffic regulation provides this screen between hosts and remote sites. This typically happens at three basic areas of the network: routers, firewalls and hosts. Each provides similar service at different points in the network. In fact the line between them is somewhat ill-defined and arbitrary. In this article, I will use the following definitions:

- **Router traffic regulation:** Any traffic regulation that occurs on a router or terminal server (hosts whose primary purpose is to forward the packets of other hosts) and is based on packet characteristics. This does not include application gateways but does include address translation.
- **Firewall traffic regulation:** Traffic regulation or filtering that is performed via application gateways or proxies.
- **Host traffic regulation:** Traffic regulation that is performed at the destination of a packet. Hosts are playing a smaller and smaller role in traffic regulation with the advent of filtering routers and firewalls.

1.12. Filters and access lists

Regulating which packets can go between two sites is a fairly simple concept on the surface- it shouldn't be and isn't difficult for any router or firewall to decide simply not to forward all packets from a particular site. Unfortunately, the reason most people connect to the Internet is so that they may exchange packets with remote sites. Developing a plan that allows the right packets through at the right time and denies the malicious packets is a thorny task which is far beyond this article's scope. A few basic techniques are worth discussing, however.

- **Restricting access in, but not out:** Almost all packets (besides those at the lowest levels which deal with network reachability) are sent to destination sockets of either UDP or TCP. Typically, packets from remote hosts will attempt to reach one of what are known as the well-known ports. These ports are monitored by applications that provide services such as Mail Transfer and Delivery, Usenet News, the time, Domain Name Service, and various login protocols. It is trivial for modern routers or firewalls only to allow these types of packets through to the specific machine that provides a given service. Attempts to send any other type of packet will not be forwarded. This protects the internal hosts, but still allows all packets to get out. Unfortunately this isn't the panacea that it might seem.

- **The problem of returning packets**: Let's pretend that you don't want to let remote users log into your systems unless they use a secure, encrypting application such as S/Key. However, you are willing to allow your users to attempt to connect to remote sites with telnet or ftp. At first glance, this looks simple: you merely restrict remote connections to one type of packet and allow any type of outgoing connection. Unfortunately, due to the nature of interactive protocols, they must negotiate a unique port number to use once a connection is established. If they didn't, at any given time, there could only be one of each type of interactive session between any given two machines. This results in a dilemma: all of a sudden, a remote site is going to try to send packets destined for a seemingly random port. Normally, these packets would be dropped. However, modern routers and firewalls now support the ability to dynamically open a small window for these packets to pass through if packets have been recently transmitted from an internal host to the external host on the same port. This allows connections that are initiated internally to connect, yet still denies external connection attempts unless they are desired.
- **Dynamic route filters**: A relatively recent technique is the ability to dynamically add entire sets of route filters for a remote site when particular sets of circumstances occur. With these techniques, it is possible to have a router automatically detect suspicious activity (such as ISS or SATAN) and deny a machine or entire site access for a short time. In many cases this will thwart any sort of automated attack on a site.

Filters and access lists are typically placed on all three types of systems, although they are most common on routers.

- **Address Translation:** Another advancement has been to have a router modify outgoing packets to contain their own IP number. This prevents an external site from knowing any information about the internal network, it also allows for certain tricks to be played which provide for a tremendous number of additional internal hosts with a small allocated address space. The router maintains a table which maps an external IP number and socket with an internal number and socket. Whenever an internal packet is destined for the outside, it is simply forwarded with the routers IP number in the source field of the IP header. When an external packet arrives, it is analyzed for its destination port and re-mapped before it is sent on to the internal host. The procedure does have its pitfalls. checksums have to be recalculated because they are based in part on IP numbers, and some upper layer protocols encode/depend on the IP number. These protocols will not work through simple address translation routers.
- **Application gateways and proxies:** The primary difference between firewalls and routers is that firewalls actually run applications. These applications frequently include mail daemons, ftp servers and web servers. Firewalls also usually run what are known as application gateways or proxies. These are best described as programs which understand a protocol's syntax, but do not implement any of the functionality of the protocol. Rather, after verifying that a message from an external site is appropriate, they send the message on to the real daemon which processes the data. This provides security for those applications that are particularly susceptible to interactive attacks. One advantage of using a firewall for these services is that it makes it very easy to monitor all activity, and very easy to quickly control what gets in and out of a network.

1.13. IP Security

A secure network starts with a strong security policy that defines the freedom of access to information and dictates the deployment of security in the network. Cisco Systems offers many technology solutions for building a custom security solution for Internet, extranet, intranet, and remote access networks. These scalable solutions seamlessly interoperate to deploy enterprise-wide network security. Cisco offers comprehensive support for perimeter security, user authentication and accounting, and data privacy. Cisco's IPSec delivers a key technology component for providing this total security solution.

Cisco's IPSec offering provides privacy, integrity, and authenticity for networked commerce-crucial requirements for transmission of sensitive information over the Internet. Cisco's unique end-to-end offering allows customers to implement IPSec transparently into the network infrastructure without affecting individual workstations or PCs. Cisco IPSec technology is available across the entire range of computing infrastructure: Windows 95, Windows NT 4.0, Cisco IOS™ software, and the Cisco PIX Firewall.

IPSec is a framework of open standards for ensuring secure private communications over the Internet. Based on standards developed by the Internet Engineering Task Force (IETF), IPSec ensures confidentiality, integrity, and authenticity of data communications across a public network. IPSec provides a necessary component of a standards-based, flexible solution for deploying a network-wide security policy.

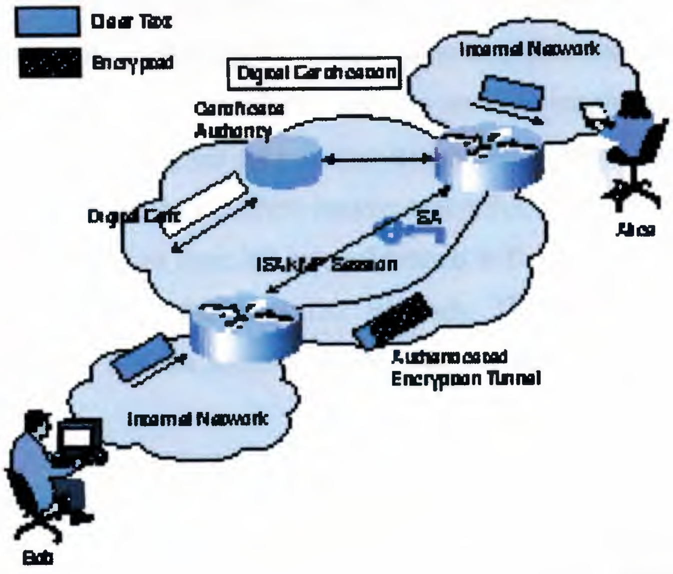


Figure 1.7 IPSec with IKE Deployed across a Public IP Network

The component technologies include:

- *Diffie-Hellman, a public-key method for key exchange*-This feature is used within IKE to establish ephemeral session keys.
- *DES*-The Data Encryption Standard (DES) is used to encrypt packet data.
- *MD5/SHA*-The Message Digest 5/SHA hash algorithms are used to authenticate packet data.

1.13.1. Benefits

IPSec is a key technology component of Cisco's end-to-end network service offerings. Working with its partners in the Enterprise Security Alliance, Cisco will ensure that IPSec is available for deployment wherever its customers need it. Cisco and its partners will offer IPSec across a wide range of platforms, including Cisco IOS software, Cisco PIX Firewall, Windows 95, Windows NT4.0, and Windows NT 5.0. Cisco is working closely with the IETF to ensure that IPSec is quickly standardized and is available on all other platforms. Customers who use Cisco's IPSec will be able to secure their network infrastructure without costly changes to every computer. Customers who deploy IPSec in their network applications gain privacy, integrity, and authenticity controls without affecting individual users or applications. Application modifications are not required, so there is no need to deploy and coordinate security on a per-application, per-computer basis. This scenario provides great cost savings because only the infrastructure needs to be changed. IPSec provides an excellent remote user solution. Remote workers will be able to use an IPSec client on their PC in combination with Layer 2 Tunneling Protocol (L2TP) to connect back to the enterprise network. The cost of remote access is decreased dramatically, and the security of the connection actually improves over that of dialup lines.

1.13.2. Applications

The Internet is rapidly changing the way we do business. While the speed of communications is increasing, the costs are decreasing. This unprecedented potential for increased productivity will reward those who take advantage of it. The Internet enables such things as:

- ***Extranets:*** Companies can easily create links with their suppliers and business partners. Today, this linkage must be accomplished with dedicated leased lines or slow-speed dial lines. The Internet enables instant, on-demand, high-speed communications.
- ***Intranets:*** Most large enterprises maintain unwieldy and costly wide-area networks. While the cost of dedicated lines has been greatly reduced, there is no question that the Internet offers a drastic cost savings.
- ***Remote users:*** The Internet provides a low-cost alternative for enabling remote users to access the corporate network. Rather than maintaining large modem banks and large phone bills, the enterprise can enable the remote user to access the network over the Internet. With just a local phone call to the Internet service provider, the user can have access to the corporate network.

These and other Internet applications are changing the way businesses communicate. The Internet provides the public communications infrastructure necessary to make all this possible. Unfortunately, the Internet is missing some key components, such as security, quality of service, reliability, and manageability. IPSec is one of the key technologies for providing security as a foundation network service.

1.14. IPSec Network Security

IPSec is a framework of open standards developed by the Internet Engineering Task Force (IETF) that provides security for transmission of sensitive information over unprotected networks such as the Internet. It acts at the network level and implements the following standards:

- IP Sec
- Internet Key Exchange (IKE)
- Data Encryption Standard (DES)
- MD5 (HMAC variant)
- SHA (HMAC variant)
- Authentication Header (AH)
- Encapsulating Security Payload (ESP)

IPSec services provide a robust security solution that is standards-based. IPSec also provides data authentication and anti-replay services in addition to data confidentiality services, while CET provides only data confidentiality services.

1.15. IPSec Encryption Technology

IPSec shares the same benefitsc as CET both technologies protect sensitive data that travels across unprotected networks, and, like CET, IPSec security services are provided at the network layer, so you do not have to configure individual workstations, PCs, or applications. This benefit can provide a great cost savings. Instead of providing the security services you do not need to deploy and coordinate security on a per-application, per-computer basis, you can simply change the network infrastructure to provide the needed security services.

IPSec encryption offers a number of additional benefits not present in CET:

- Because IPSec is standards-based, enables Cisco devices to interoperate with other IPSec-compliant networking devices to provide the IPSec security services. IPSec-compliant devices could include both Cisco devices and non-Cisco devices such as PCs, servers, and other computing systems.

  Cisco and its partners, including Microsoft, are planning to offer IPSec across a wide range of platforms, including Cisco IOS software, the Cisco PIX Firewall, and Windows 2000.

- Enables a mobile user to establish a secure connection back to the office. For example, the user can establish an IPSec "tunnel" with a corporate firewall—requesting authentication services—in order to gain access to the corporate network. all of the traffic between the user and the firewall will then be authenticated. The user can then establish an additional IPSec tunnel—requesting data privacy services—with an internal router or end system.

- Provides support for the Internet Key Exchange (IKE) protocol and for digital certificates. IKE provides negotiation services and key derivation services for IPSec. Digital certificates allow devices to be automatically authenticated to each other without the manual key exchanges required by Cisco Encryption Technology.

This support allows IPSec solutions to scale better than CET solutions, making IPSec preferable in many cases for use with medium-sized, large-sized, and growing networks, where secure connections between many devices is required.

1.21. Summary

A basic understanding of computer networks is requisite in order to understand the principles of network security. We have covered some of the foundations of computer networking, then move on to an overview of some popular networks. Following that, we'll take a more in-depth look at TCP/IP, the network protocol suite that is used to run the Internet and many intranets. Once we've covered this, we'll go back and discuss some of the threats that managers and administrators of computer networks need to confront, and then some tools that can be used to reduce the exposure to the risks of network computing. This article is a general introduction to network security issues and solutions in the Internet. emphasis is placed on route filters and firewalls. It is not intended as a guide to setting up a secure network.

2. FIREWALLS AND PACKET FILTERING

2.1. Overview

Firewalls are a very effective type of network security. This chapter briefly describes what Internet firewalls can do for your overall site security. describes the various types of firewalls in use today. In building construction, a firewall is designed to keep a fire from spreading from one part of the building to another. In theory, an Internet firewall serves a similar purpose: it prevents the dangers of the Internet from spreading to your internal network. In practice, an Internet firewall is more like a moat of a medieval castle than a firewall in a modern building. It serves multiple purposes:

- It restricts people to entering at a carefully controlled point.
- It prevents attackers from getting close to your other defenses.
- It restricts people to leaving at a carefully controlled point.

An Internet firewall is most often installed at the point where your protected internal network connects to the Internet, as shown in figure 2.1

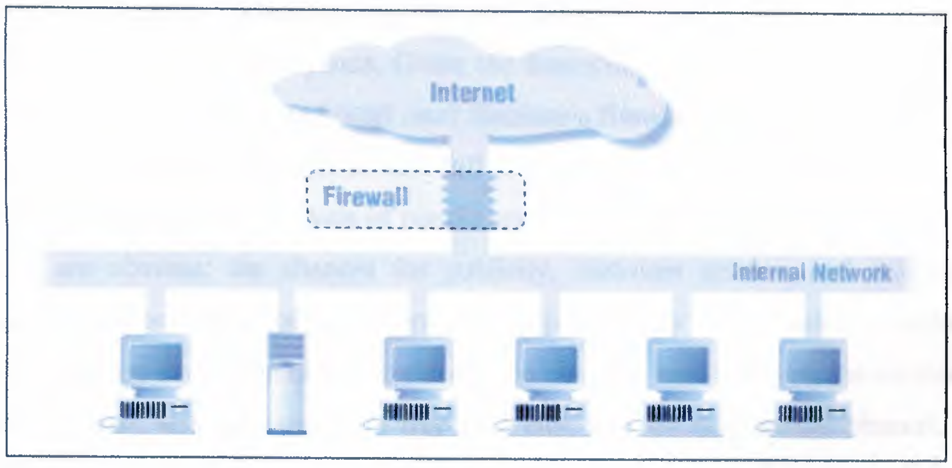


Figure 2.1 A firewall usually separates an internal network from the Internet

All traffic coming from the Internet or going out from your internal network passes through the firewall. Because it does, the firewall has the opportunity to make sure that this traffic is acceptable. What does "acceptable" mean to the firewall? It means that whatever is being done - email, file transfers, remote logins, or any kinds of specific interactions between specific systems - conforms to the security policy of the site. Security policies are different for every site; some are highly restrictive and others fairly open. Logically, a firewall is a separator, a restricter, an analyzer. The physical implementation of the firewall varies from site to site. Most often, a firewall is a set of hardware components - a router, a host computer, or some combination of routers, computers, and networks with appropriate software. There are various ways to configure this equipment; the configuration will depend upon a site's particular security policy, budget, and overall operations. A firewall is very rarely a single physical object, although some of the newest commercial products attempt to put everything into the same box. Usually, a firewall has multiple parts, and some of these parts may do other tasks besides function as part of the firewall. Your Internet connection is almost always part of your firewall. Even if you have a firewall in a box, it isn't going to be neatly separable from the rest of your site; it's not something you can just drop in. We've compared a firewall to the moat of a medieval castle, and like a moat, a firewall is not invulnerable. It doesn't protect against people who are already inside; it works best if coupled with internal defenses; and, even if you stock it with alligators, people sometimes manage to swim across. A firewall is also not without its drawbacks; building one requires significant expense and effort, and the restrictions it places on insiders can be a major annoyance. Given the limitations and drawbacks of firewalls, why would anybody bother to install one? Because a firewall is the most effective way to connect a network to the Internet and still protect that network. The Internet presents marvelous opportunities. Millions of people are out there exchanging information. The benefits are obvious: the chances for publicity, customer service, and information gathering. The popularity of the information superhighway is increasing everybody's desire to get out there. The risks should also be obvious: any time you get millions of people together, you get crime; it's true in a city, and it's true on the Internet. Any superhighway is fun only while you're in a car. If you have to live or work by the ighway, it's loud, smelly, and dangerous. How can you benefit from the good parts of the Internet without being overwhelmed by the bad? Just as you'd like to drive on a highway without suffering the nasty effects of putting a freeway off-ramp into your living room, you need to carefully control the contact that your network has to the

Internet. A firewall is a tool for doing that, and in most situations, it's the single most effective tool for doing that. There are other uses of firewalls. For example, they can be used as firewalls in a building that divide parts of a site from each other when these parts have distinct security needs (and we'll discuss these uses in passing, as appropriate).

2.1.1. What Can a Firewall Do?

Firewalls can do a lot for your site's security. In fact, some advantages of using firewalls extend even beyond security.

2.1.2. What Can't a Firewall Do?

Firewalls offer excellent protection against network threats, but they aren't a complete security solution. Certain threats are outside the control of the firewall. we need to figure out other ways to protect against these threats by incorporating physical security, host security, and user education into your overall security plan. Some of the weaknesses of firewalls are discussed below.

2.2. Firewall Politics

Firewalls are used for two purposes:

1. to keep people (worms / crackers) out.
2. to keep people (employees / children) in.

How do we distinguish between the good and the bad network packets?

1. First of all, we know that the adversary is able to attach a new host to the internal network, where a trusted host is a host controlled by the firewall administrator. The firewall should never let a network packet through from an untrusted host.
2. We know that any user can start any (potentially hostile) applications. So we need the firewall to distinguish between trusted and untrusted applications, where a trusted application is an application installed by the firewall administrator on a trusted host. The firewall should never let a network packet through from an untrusted application.

3. Lastly, we might consider some users to be more trust worthy than others. It would, in fact, be a nice feature if we can allow trusted users to run potentially hostile applications, while other users cannot. For instance, the system administrator might be allowed to run some VPN application (Virtual Private Network), which interconnects one network to another. VPN software can certainly be used as a backdoor to the system, other users must not be allowed to communicate with this software. If the firewall can some how retrieve information about the identity of the host, the application and the user, responsible for sending each network packet it receives, it has a chance of determining whether the network packet should be allowed to proceed to the outside or not.

2.2.1. How it create a security policy

we have seen some realy high folutin documentation on how to create a security policy. After many years of experence we know now to say, don't believe a word of them. Create a security policy is simple.

1. describe what you need to service
2. describe the group of people you need to service
3. describe which service each group needs access to
4. for each service group describe how the service should be keep secure
5. write a statment making all other forms of access a vialation

Your policy will become more complicated with time but don't try to cover to much ground now. Make it simple and clear.

2.3. Types of Firewalls

There are two types of firewalls :

1. Filtering Firewalls - that block selected network packets.
2. Proxy Servers (sometimes called firewalls) - that make network connections for you.

2.3.1. Packet Filtering Firewalls

Packet Filtering is the type of firewall built into the Linux kernel.

A filtering firewall works at the network level. Data is only allowed to leave the system if the firewall rules allow it. As packets arrive they are filtered by their type, source address, destination address, and port information contained in each packet.

Many network routers have the ability to perform some firewall services. Filtering firewalls can be thought of as a type of router. Because of this you need a deep understanding of IP packet structure to work with one. Because very little data is analyzed and logged, filtering firewalls take less CPU and create less latency in your network.

Filtering firewalls do not provide for password controls. User can not identify themselves. The only identity a user has is the IP number assigned to their workstation. This can be a problem if you are going to use DHCP (Dynamic IP assignments). This is because rules are based on IP numbers you will have to adjust the rules as new IP numbers are assigned. I don't know how to automate this process. Filtering firewalls are more transparent to the user. The user does not have to setup rules in their applications to use the Internet. With most proxy servers this is not true.

2.3.2. Proxy Servers

Proxies are mostly used to control, or monitor, outbound traffic. Some application proxies cache the requested data. This lowers bandwidth requirements and decreases the access the same data for the next user. It also gives unquestionable evidence of what was transferred.

There are two types of proxy servers:

1. Application Proxies - that do the work for you.
2. SOCKS Proxies - that cross wire ports.

2.3.2.1. Application Proxy

The best example is a person telneting to another computer and then telneting from there to the outside world. With a application proxy server the process is automated. As you telnet to the outside world the client send you to the proxy first. The proxy then connects to the server you requested (the outside world) and returns the data to you.

Because proxy servers are handling all the communications, they can log everything they (you) do. For HTTP (web) proxies this includes very URL they you see. For FTP proxies this includes every file you download. They can even filter out "inappropriate" words from the sites you visit or scan for viruses. Application proxy servers can authenticate users. Before a connection to the outside is made, the server can ask the user to login first To a web user this would make every site look like it required a login.

2.3.2.2. SOCKS Proxy

A SOCKS server is a lot like an old switch board. It simply cross wires your connection through the system to another outside connection. Most SOCKS server only work with TCP type connections. And like filtering firewalls they don't provide for user authentication. They can however record where each user connected to.

2.4. Firewall Technologies

2.4.1. Packet Filtering

This is the most basic means of implementing a firewall. It is a basic feature of routers and computers that are set up to route. Basic packet filtering is based on the information contained in the Layer 3 (IP) header of a packet. Decisions on security are based on the following criteria:

- Source address
- Destination address
- Protocol being used (TCP, UDP, etc.)
- Port number (21, 23, 80, etc.)

By configuring a router with a table of filters, this information can be used to either pass or block packets that flow through the router. If a connection request meets the criteria of the filter, then it is allowed to pass, otherwise it is blocked. In this way, packet filters look at the source and destination addresses and leave certain ports either open or closed. The advantage of this is that packet filtering is very fast. However, because the filters are based on static entries, this form of firewall offers the least flexibility and security. There is no way that the validity of a request can be verified in the context of the connection, and various ports are left open to trojan horses or attacks that use IP spoofing. So although this was the original and most inexpensive form of implementing a firewall, most of today's modern networks cannot rely on packet filtering alone to protect their networks. In order to provide the most flexible, secure protection against modern network attacks, application level proxies and stateful inspection systems have come into existence.

2.4.2. Application-level Proxies

According to Webster's New World College Dictionary, proxy is defined as "the authority to act for another." This is the principle behind firewalls that use application-level proxies, or gateways. Instead of operating at a lower level of the protocol stack, these operate at the application layer of the OSI model. Services that are made possible by the application layer include e-mail, file transferring, web browsing, etc. A suite of proxies, including a proxy for each of the protocols (such as POP, FTP, HTTP, etc.) that support these services, fully understands the various protocols and acts in behalf of the services. In order for a connection to be authorized, the whole data stream is inspected. Not only are things like IP addresses in the packet heading verified, but the payload is also checked to see if it contains valid data and that the data belongs to the application. Rules are applied at the application layer, and if the data or the request does not meet the criteria of these rules or if the data just doesn't make sense, then the connection is denied. Every packet is processed, validated, and re-generated by the proxy. This means that applications are not allowed to talk directly to each other, and as a result, this form of firewall is extremely secure and effective, even against application-level attacks. This means that a hacker would not be able to construct data packets that appear to have valid info at the other layers and that pass other inspections, but that actually contain harmful commands in the payload that could harm your network. Application-level proxies, or gateways, also make use of NAT (Network Address Translation), which hides your internal network's IP addresses from those on the outside. Instead, all of the traffic that originates inside your network is given the IP address of your firewall (in some cases, the packets may receive a public IP address that is different from your firewall's, but that are used only for that purpose). The firewall keeps track of which packets belong to the different computers on your internal network, and forwards these accordingly. This provides additional protection, since it keeps the IP addresses of your "trusted" network hidden from the eyes of anyone who might have malicious intent.

2.5. Firewall Architecture

There are lots of ways to structure your network to protect your systems using a firewall. If you have a dedicated connections to the Internet through a router, you could plug the router directly into your firewall system. Or, you could go through a hub to provide for full access servers outside your firewall. [8]

2.5.1. Dial-up Architecture

You may be using a dialup service like an ISDN line. In this case you might use a third network card to provide provide a filtered DMZ. This gives you full control over your Internet services and still separates them from your regular network.

```
                      ___________
    _/\__/\_         |           |           ______________
   |        |        | Firewall  |   (LAN)  |              |
  / Internet \-------|  System   |--(HUB)---| Workstation/s |
  \_  _  _  _/       |___________|          |______________|
    \/ \/ \/               |
                      (DMZ)  (HUB)
```

Figure 2.2 Dial-up Architecture

2.5.2. Single Router Architecture

If there is a router or cable modem between you and the Internet. If you own the router you could setup some hard filter rules in the router. If this router is owned by your ISP so you may not the have the needed controls. You can ask your ISP to put in filters.

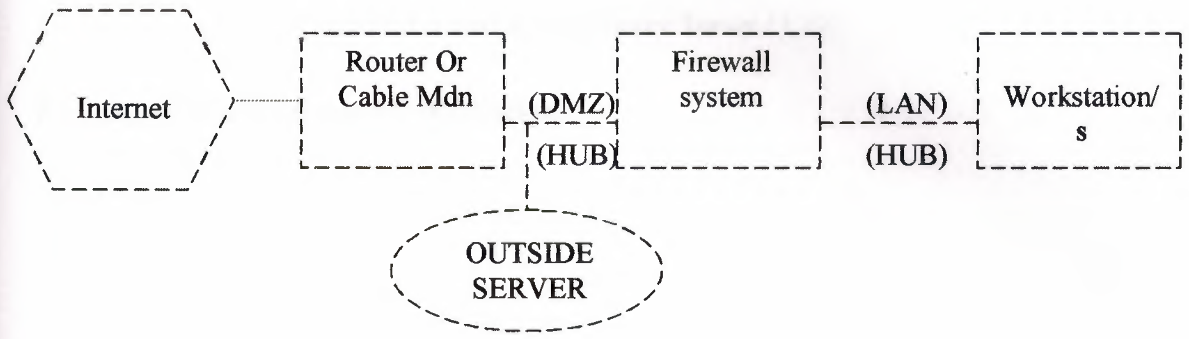


Figure 2.3 Single Router Architecture

2.5.3. Firewall with Proxy Server

If you need to monitor where users of your network are going and your network is small, you can intergrate a proxy server into your firewall. ISP's some times do this to create interest list of their users to resell to marketing agencies.

```
                       __________
     _/\__/\_         | Proxy /  |
                                           ________________
   |          |       | Firewall |   (LAN)  |                |
  / Internet \----|  System   |--(HUB)--| Workstation/s  |
  \_  _  _  _/        |__________|          |________________|
    \/ \/ \/
```

Figure2.4 Firewall with Proxy Server

You can put the proxy server on your LAN as will. In this case the firewall should have rules to only allow the proxy server to connect to the Internet for the services it is providing. This way the users can get to the Internet only through the proxy.

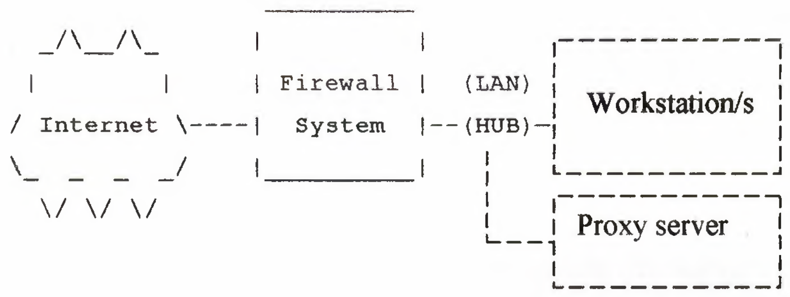


Figure2.5 Firewall with Proxy Server / LAN

2.5.4. Redundent Internet Configuration

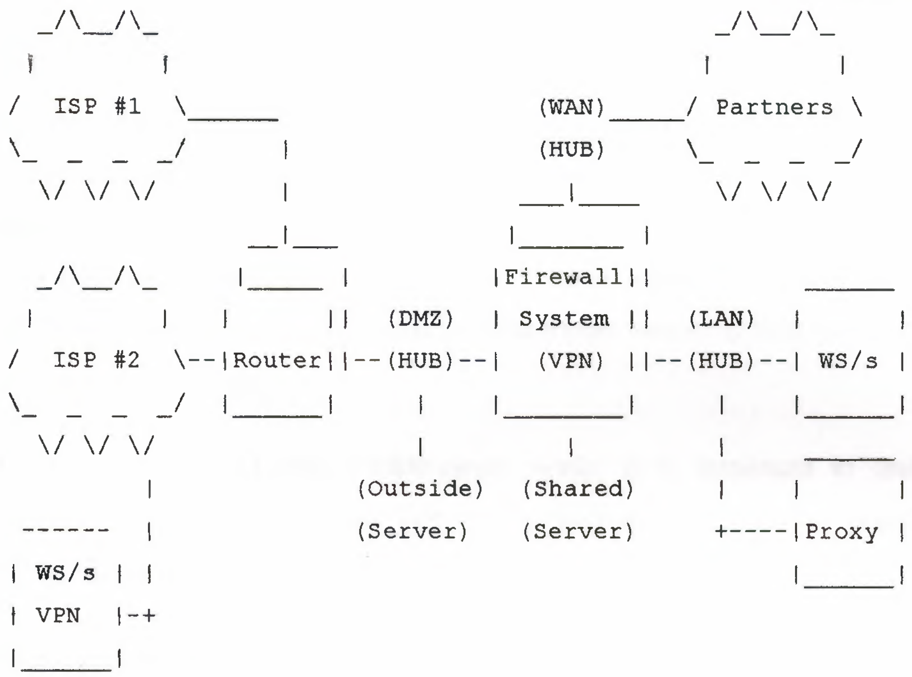


Figure 2.6 round-robin DNS techniques to provide access to multipule web servers from one URL and multiple ISP's.

If you are going to run a service like YAHOO or maybe SlashDot you may want to make your system by using redundant routers and firewalls. (Check out the High Availability HowTo.) By using a round-robin DNS techniques to provide access to multipule web servers from one URL and multiple ISP's, routers and firewalls using High Avaibility technics you can create a 100% uptime service. It is easy to let your network get out of hand. Keep control of every connection. It only takes a user with a modem to compromise your LAN.

2.6. Web Proxy Architecture

2.6.1. How the Web Proxy Service Works

2.6.1.1. The CERN-Proxy Protocol

The CERN-proxy protocol is widely recognized throughout the World Wide Web (WWW) community as the standard for implementing proxy services in TCP/IP-based networks. It has its origins within the standards for Hypertext Transfer Protocol (HTTP) as a UNIX-based service first developed by members of Switzerland's Conseil Europeen pour la Recherche Nucleair (European Laboratory for Particle Physics, or CERN) in the early 1990s.

As the CERN staff added application-aware proxy support for Hypertext Transfer Protocol daemon (HTTPd) servers commonly known as Web servers to their libraries, the WWW community built on these additions. In the time since it was first introduced, the CERN-proxy protocol has become an accepted industry standard for implementing HTTP proxy service. The Proxy Server Web Proxy service is fully compatible with the CERN-proxy standard.

To better understand how CERN-proxy works, it is important to understand the differences between how most Internet applications, such as File Transfer Protocol (FTP) and Gopher, work and how HTTP applications work.

Standard TCP/IP applications such as FTP use TCP (or in some cases, User Datagram Protocol, or UDP) as the transport-level protocol for supporting client/server communications. For HTTP-based applications, a set of commands (called methods) are defined that are used in Web-based client/server communications. While CERN-compatible proxy services support WWW (HTTP), FTP, and Gopher requests, it is important to keep in mind that a CERN-based Web proxy server uses HTTP for all communications with its Web Proxy clients.

2.6.2. How HTTP Works

HTTP defines a set of commands (called *methods*) that a client can send to a server. The two most common methods are:

- **GET** GET is used to forward a Uniform Resource Locator (URL) to a server requesting the resource to which the URL refers.
- **POST** POST is used to forward a request that contains a URL and data; typically, a user provides this data by completing a Hypertext Markup Language (HTML) form.

2.6.3. How the GET Method Works

In a simple HTTP request, a browser that is not configured for proxy service sends an HTTP URL directly to a Web server. The browser sends the server a GET method, which includes the path and resource name requested. In the process, the browser will remove the protocol and site name from the URL (http://*domainname*) before forwarding the GET method request on to the site named in the URL.

For example, if the following URL is typed on the command line of a browser not configured for proxy service

```
http://host.com/sales/report.htm
```

the browser parses the URL and sends the following command to Host.com as:

```
GET /Sales/Report.htm
```

This type of processing simplifies communications because HTTP is the protocol for all message requests that browser clients communicate to the server, but it limits browser requests to sending and receiving by use of HTTP only. Browser requests for FTP URLs can not be handled directly in this manner, although proxy service can support these types of requests for browser clients.[5]

2.6.4. Examples of GET Usage with Proxy Service

When a browser is configured for use with a proxy server, GET methods issued by the browser are created with more detail about the resource included. The browser client will issue the full URL without parsing the named protocol first. The named protocol indicates the type of service being requested and whether the GET request is for a WWW, FTP, or Gopher resource. The fully detailed GET method is then forwarded to the proxy server.

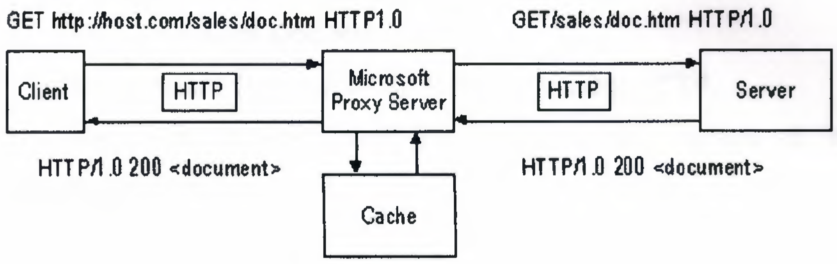


Figure 2.7 following is an example of a WWW (HTTP) request that shows proxy-based service for a document entitled Doc.htm in the Sales directory on the server Host.com

The process by which a proxy request is serviced is as follows:

1. The Web browser sends the full URL request as a GET statement to the proxy server.
2. The proxy server:
   - Receives the request.
   - Parses the URL.

   If the URL is in the cache, the request is serviced to the Web browser from the cache.

   - Identifies the type of resource the request is for (such as HTTP or FTP).
   - Resolves the domain name to an IP address.
   - Requests Doc.htm from the Internet server by using the appropriate protocol, such as HTTP or FTP.

In this case, because the URL specified HTTP, HTTP is the appropriate protocol. Note that a request sent to a Web server is formatted by the proxy service as a standard request, that is, a request not forwarded by the proxy service.

3. The Web server:
   - Receives the request.
   - Responds by sending Doc.htm to the proxy by using HTTP.
4. The proxy server:
   - Receives Doc.htm.
   - Sends Doc.htm to the browser by using HTTP.
5. The Web browser receives Doc.htm and displays it on-screen, completing the process.

Another example of this process will demonstrate how an FTP resource is requested and handled under proxy service.

The following diagram shows how an FTP request is forwarded using proxy service. The browser sends a request to the proxy service for an FTP-published document named Q296.doc. The GET method that is used contains the full FTP URL entered at the browser command line, ftp://host.com/sales/Q296.doc. (Note that the HTTP protocol is used by the browser to send the GET method to the proxy.)

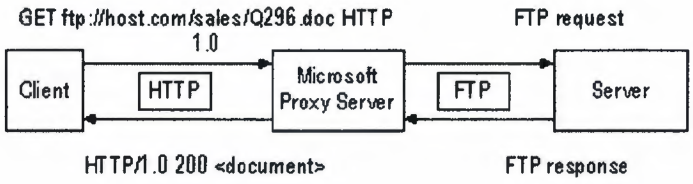


Figure 2.8 how an FTP request is forwarded using proxy service

In this case, the proxy service identifies the request as the FTP protocol, and requests Q296.doc from host.com by using the FTP protocol. Host.com then returns Q296.doc to the proxy by using the FTP protocol, and the proxy uses the HTTP protocol to send the document to the client.

Browsers that are not configured for the proxy service issue FTP and Gopher requests by using the FTP and Gopher protocols, respectively. A browser that is not configured for the proxy service cannot issue an FTP or Gopher requests by using HTTP.

Configuring a browser to communicate with a proxy server actually simplifies the work that the browser needs to do, because all requests are processed with HTTP and have complete URLs.[5]

2.7. Firewall placement

Security is not good if it isn't in the right place. Think about a modern office building, where are the doors with locks? The lobby doors can always be locked, and usually the doors on each floor have locks as well. If only the office doors had locks then it would take a lot more effort to secure the building (i.e. many more locks would be needed) as well as security guards to make sure no-one is trying to force a door. In a perfect world every door would have a strong lock and a guard sitting next to it with a shotgun, but since cost is a factor this won't happen. The same applies to networks, while there are many free firewalls available the cost of installing and maintaining a firewall on each machine would be high, it is much more efficient to have firewalls where your network connects to untrusted networks. As well you should consider the type of traffic a firewall will need to handle, the more traffic it must handle the more complex the rules will need to be, and the greater a chance an attacker can slip in.

2.7.1. Before or after?

It goes without saying that your internal LAN should have a firewall between it and the Internet. But what about servers that won't work properly unless machines on the Internet can connect to them? An Internet connection with no capacity for incoming email, would be very Spartan indeed (web servers and DNS are also usually hosted on-site in most cases). If you place the email server behind the firewall, on your corporate LAN then you must "poke" holes in the firewall to let email in. If an attacker manages to compromise the mail server then there is nothing between it and your internal LAN to slow down further attacks. If you place the Email server in front of the firewall, you can easily set up firewall rules to allow internal LAN hosts to connect to it and send/retrieve email. If an attacker compromises the mail server they will not be able to access the internal LAN, since there is a firewall.[6]

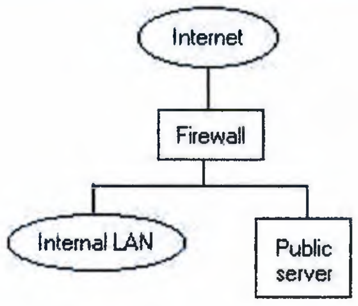
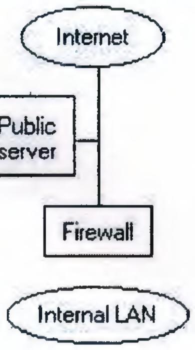
	

Figure2.9 One the attacker has gained access to the Public server they can easily attack the internal LAN	**Figure2.10** Attacker will have less problems attacking the public server, but cannot use it to attack internal LAN as easily

2.7.2. In-between

For the best of both worlds you need two firewalls, on in front of your public servers, and one in-between the public servers and the internal LAN. This setup is usually refereed to as the "DMZ", because the zone in-between the two firewalls is untrusted and heavily restricted. Additionally you can setup application proxies (such as www and ftp proxies), then block outbound access from the internal LAN on the exterior firewall and force all clients to go through your application proxies (which can have anti-virus capabilities for example). The DMZ should be a relatively "quiet" zone, any hostile packets trying to enter it (from the Internet, or the Internal LAN) should be blocked at either firewall, increasing the effectiveness of any intrusion detection systems (there will be less false positives). But the DMZ concept is far from perfect too. It is common to have the public servers and the internal LAN served off the same link and firewalls, increasing the amount and types of traffic, and thus the complexity of the firewall rules. Ideally you would have a path from the internal LAN to a DMZ with the public servers that is only used for internal LAN to public servers traffic (i.e. website updates, retrieving email, etc.). There would be a second path from the internal LAN to another DMZ with proxy servers for Internet services (i.e. WWW, FTP)

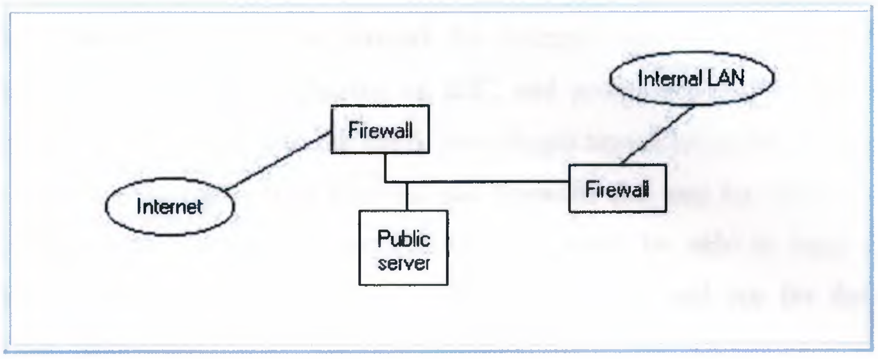


Figure 2.11 An attacker would need to get through the first firewall, and the second one to penetrate the internal LAN

that is used for Internet traffic only. This would greatly reduce the complexity, on the "public" DMZ the only traffic would be services you provide, and on the "internet access" DMZ there would only be outgoing traffic with few or no incoming connections being required (the notable exception being FTP). This of course would require 4 firewalls, and two completely separate configurations, however the reduction in complexity (especially for large sites) is probably worth it. Also since the "public" servers are more likely to be attacked then the proxy servers surviving attacks (especially if you had separate Internet links for each) would be much easier.

2.7.3. Types of traffic

Generally speaking there are two types of traffic, inbound and outbound (if you see traffic going sideways call me). Usually incoming traffic will generate a response in the form of outgoing traffic, and vice versa. Ideally you should place limits on what ports foreign machines are allowed to connect to (i.e. locally ports 25, 53 and 80 are ok), and limits should be placed on which ports internal hosts can connect out to. This is very important with the growth of trojan horses and other "backdoors", attackers can send infected email and in a matter of minutes an entire corporation can be infected with a Melissa or "I love you" type virus. Imagine if it carried a payload such as Back Orifice 2000 (a popular Windows trojan) and was set to connect out to a certain host, from which it would take commands (such as "send me the contents of C:\ My Documents" or "log all keystrokes and email them to me"). The cost of cleaning up from such a problem, let alone any cost associated with lost or stolen data, passwords and other sensitive information is very high. By blocking (and logging) outgoing connections you can detect improper usage of the network for example outgoing connections to port 6667 would indicate a user is chatting on IRC, and potential security breaches, or it might be Back Orifice 2000 with the speak easy plugin broadcasting the machine it just infected. Splitting the traffic load between two firewalls (i.e. one for inbound and one for outbound) doesn't make any sense since you won't be able to keep state for connections. Splitting types of traffic (i.e. one for servers, and one for the internal clients) does make sense.[7]

2.7.4. What to block

Oddly enough this is something many people don't think about a whole lot, in some cases you can simply deny everything and have a few specific allow rules which results in a pretty tight configuration. However it is more likely that you have specific blocking rules and allow most other things. Also this is usually based on port numbers (i.e. service) and destination, however source is also very important. Even if you only allow a few trusted IP addresses to say connect to your "secret" web server, an attacker can still spoof packets, and so on. You can reduce the risk by blocking IP addresses that are in "high risk" environments, such as Universities, foreign countries and so on (assuming of course you are not terribly interested in talking to them via the Internet).

2.7.5. Foreign sites

If your business is only concerned about North America for example then it might make sense to heavily restrict access from other countries such as Russia, and China. If you are securing network sites that are not providing public network services (such as WWW sites) then you should probably restrict access from network blocks like 24.* (cablemodem providers, a favorite jumping point for attackers). You may also wish to restrict access to external sites, for reasons including content or employee distractions.

For example by blocking access to login.oscar.aol.com (152.163.242.24, 152.163.242.28, 152.163.241.120, 152.163.241.128) you can effectively cripple AOL

Instant Messenger (AIM) from working. If you wanted to block access to Freedom network you would block outgoing connections to ports: 51102/tcp and 51112/tcp.[7]

If you have specific software or services you wish to block access to the best thing to do is install it on a test machine (preferably behind a firewall). You can then run the software and use "netstat" to find out who the machine is talking to, and if there is a firewall in-between you can closely monitor what it is connecting to and so on (add a logging rule for all packets to or from that host).

2.7.6. Internal sites

Chances are not all of your internal machines require access to the Internet, and by blocking them you can head off problems. Machines without Internet access cannot connect to "naughty" sites, and trojan horse software running on them cannot contact outside sites to report it is installed, or send stolen passwords. Any machines providing services to Internet users (such as DNS, WWW, Email) should b allowed access to the Internet (otherwise they will not work), although you may wish to restrict it, for example the mail server should only need to reply to clients that initiate connections, and only establish connections to other mail servers (port 25), there is no need for the mail server to establish connections to machines on any other port then 25. Thus if someone were to break into your mailserver they would only be able to attack other machines on a single port (port 25), instead of all 65,536 ports (and of course you would be able to quickly detect this type of behavior). Web servers should only need to answer queries to ports 80 (and 443 for secure web), and generally speaking do not need to establish outbound connections. By heavily restricting the outbound access of machines you can significantly reduce your exposure, and increase the chance of detecting a security incident.

2.7.7. Services

There are many services, thousands in fact. However some of these services are so common, and so dangerous that they warrant special attention. The most common problems are in what I call network infrastructure protocols. Things that almost all networks use such as DHCP, DNS, SNMP, LPR, NFS, SMB, which provide basic network management, or services such as file and print sharing. Generally speaking these do not need to be shared out across the Internet, and if remote users do need access to them (such as file and print sharing) they should be tunneled through a VPN (such as IPSec) and not allowed to go out in the clear.

23/tcp - telnet, cleartext authentication and sessions, should not be used (replace with SSH).
37/tcp and udp - time, use ntp (Network Time Protocol) instead
67/tcp and udp - bootp server, should only be used locally
68/tcp and udp - bootp client, should only be used locally
69/tcp and udp - tftp (Trivial FTP), should only be used locally
79/tcp - finger, should only be used locally
110 and 111/tcp and udp - POP2 and POP3, if remote users need access use SSL wrapped POP or VPN
143/tcp and udp - IMAP, if remote users need access use SSL wrapped POP or VPN
161/udp - SNMP, attackers love this protocol
162/udp - SNMP-trap, attackers love this protocol
177/tcp and udp - xdmcp (X Display Manager Control Protocol, restrict access or VPN
389/tcp - LDAP, restrict access or VPN
512, 513 and 514/udp and tcp - various remote services and logging, restrict access or VPN
1812/tcp and udp - Radius, restrict access or VPN

And the list goes on and on. The decision tree should look like this:

Can we firewall it completely? If yes do so.
Can we restrict access to it to people via a VPN only (i.e. IPSec)? If yes do so.
Can we firewall it restrictively (i.e. to "trusted hosts")? If yes do so.
Can we restrict access to it from "risky" sources (i.e. Canadian colleges, China, the USA)? If yes do so.

Basically what is the most restrictive thing you can do, and the service is still useful.

2.8. Common configuration problems

There are many common configuration problems with firewalls, ranging in severity and scope. By far the most common problems relate to what should be blocked or allowed. This is often problematic because needs change, you may need to allow video streaming for example, and unless done properly the addition of new firewall rules can seriously undermine the security provided by a firewall. Before any changes are made to a firewall you should sit down with whoever is responsible, and ensure they will not have unintended side effects. I find the best way to do this is to print out the rules and make sure the new rules fit logically into the existing structure. For example my rules typically start with rules to block private and non routed network (like 10.*, 127.*, and so on), followed by ICMP related rules, then I have rules that allow traffic in (like SSH, Email, WWW and so on), and then depending on the security required I block the first 1024 ports (which are usually the most interesting ones), or I have a default deny policy. If I need to add a new rule to let in secure WWW traffic for example I would not add it to the front of the list, I would add it in the chapter with the specific allowals. Adding it to the front of the list would result in attackers being able to send packets to port 443 (secure WWW) from non routed addresses. If your firewall has multiple administrators this can be especially dangerous. You should sit down together and write out a document describing how and why rules are in place, ideally this document should be online so it can be referred to, and modified as needed (always document changes!).

Another common problem is people not blocking broadcast traffic on firewalls. Broadcast traffic almost never needs to be passed through firewalls, and generally speaking traffic sent to the network address should not be passed either (in general this means anything ending in a 0 or 255 should be blocked, your netmasks may be different so make sure it's correct). Attackers will often try to use broadcast ICMP traffic to attack other sites, this is called "smurfing". By pinging your broadcast address instead of one host responding, many hosts respond, all they need to do is forge the return address, and some unwitting victim gets blasted by your network. Windows machines are also in the habit of sending out a lot of broadcast traffic, the default installs of pretty much every Microsoft OS results in it advertising it's name and any services it is running, this is especially dangerous for NT servers. On a modern network there should be very little broadcast traffic.

Firewalls should block certain network management protocols, especially bootp/bootpc, SNMP and RPC based services. These are services attackers will attempt to compromise because they are almost installed, and in many cases they will not be configured securely (i.e. SNMP community names of "public" or "private" are far to common). Windows file sharing should generally not be used across the Internet, blocking ports 135 to 139 (udp and tcp) at the firewall is often a good idea, if people do need access to their files this should be done through a virtual private network. Ideally the only things let into a corporate network should be public network services like DNS, Email and WWW (and even then they should only be allowed to a few secured servers), and returning packets from client connections. VPN technology has matured to the point where there is not excuse not to use it to connect sites securely that need to transfer sensitive information such as corporate files.

Management of firewalls is also often handled poorly. Telnet and R services should never be used. Web based admin tools should not be used unless they provide an encrypted link (i.e. https://), or through a VPN. Telnet, R services and unencrypted web sessions mean that the passwords and so on can be gathered by an attacker, or if they are sufficiently skilled they can hijack the connection and insert a few commands before it is dropped (like flush all rulesets). Any management should be done from a handful of highly secured workstations that are NOT equipped with mail readers, web browsers or any unneeded software. There are far to many security problems in today's mail readers, web browsers and other application software. If an attacker is able to seize control of the administration workstation they can run a key logger, network sniffer or simply seize control of it with programs like Back Orifice 2000. For day to day administration of the firewall (i.e. service packs, updates, and so on) you should install OpenSSH or the commercial version of SSH if it is UNIX, and if it is NT based you should administer it locally or use a remote administration tools that support the GUI (i.e. PCAnywhere and VNC) in conjunction with a VPN package (not PPTP!). Another secure method is to manage the firewall "out of band", i.e. use the serial console on a sparc/Cisco/etc., or PPP (or similar) on an NT box, this allows you to completely disable all network services on the firewall, including remote logins, making it very difficult for an attacker to get in using them (since they do not exist). Additionally when someone attacks your firewall through a denial of service there is a better chance of being able to login and correct it. If you decide to use a serial console/PPP and a modem then make sure the modem is secured, the best way is to enable callback support, this way an attacker would need to go to great lengths to connect to it.

Firewalls are an extremely useful component in any network security plan, they have gone from "only the government needs firewalls" to "what is the best firewall for my home computer?". Unless they are setup correctly, and managed correctly however they are worse then useless (you will think you are secure when in fact you are not).

2.9. Summary

Firewalls can do much more than just block unwanted traffic from entering your network. Most can perform a myriad of tricks, including filtering, mangling, and NATing of traffic. More concisely, they can provide control over not only who comes in and goes out, but help prioritize traffic (mangling), and masquerade internal hosts behind one or more public IPs. Firewalls can also allow you to protect your Internet servers (web, mail, FTP, etc.) while still allowing public access to those services through port forwarding - that is, forwarding an attempted connection to the firewall through to an internal host.

3. PROXY SERVERS SECURITY

3.1. Overview

The growth and increased popularity of the World Wide Web has created a corresponding growth in network traffic. With this growth have come delays, slower response times, and security concerns. The network traffic problems are partly due to the repeated retrieving of objects from remote Web servers on the Internet. Proxy Services can help improve performance by locally caching frequently requested Internet information. In general, Proxy Services stores copies of frequently requested Web information closer to the user, thereby reducing the number of times the same information is accessed over an Internet connection, the download time, and the load on the remote server.

3.2. Web Proxy Servers

3.2.1. What is Web Proxy Server?

A Web proxy server is a specialized HTTP server. The primary use of a proxy server is to allow internal clients access to the Internet from behind a firewall. Anyone behind a firewall can now have full Web access past the firewall host with minimum effort and without compromising security.

The proxy server listens for requests from clients within the firewall and forwards these requests to remote internet servers outside the firewall. The proxy server reads responses from the external servers and then sends them to internal client clients. In the usual case, all the clients within a given subnet use the same proxy server. This makes it possible for the proxy to cache documents efficiently that are requested by a number of clients. People using a proxy server should feel as if they are getting responses directly from remote servers.

Clients without Domain Name Services (DNS) can still use the Web. The proxy IP address is the only information they need. Organizations using private network address spaces such as the class A net 10.*.*.* can still use the Internet as long as the proxy is visible to both the private internal net and the Internet.[9]

Most proxy servers are implemented on a per-access method basis. Proxy servers can allow or deny internet requests according to the protocol of the requests. For instance a proxy server can allow calls to FTP servers while denying calls to HTTP servers.

3.2.2. When Web Proxy Servers are Useful

You can use a proxy server in a number of ways including:

- Permitting and restricting client access to the Internet based on the client IP address.
- Caching documents for internal documents.
- Selectively controlling access to the Internet and subnets based on the submitted URL.
- Providing Internet access for companies using private networks.
- Converting data to HTML format so it is readable by a browser.

3.3. Browser Access to the Internet

Some machines on your local network might not be able to directly access Internet resources. For instance, some browsers might not be able to directly access Internet resources because they run on systems behind a protective firewall. In these cases, a proxy server can retrieve the desired files for them.[10]

In Figure 3.1, the Proxy server is running on a firewall host and making connections to the outside world using firewall software. You could also run the proxy server on another internal host that has full internet access, or on a machine inside the firewall.

The proxy server receives the request from the browser in the form of a URL. The proxy server retrieves the requested information, converts it to HTML format and sends it on to the browser behind the firewall. The proxy server can handle all network requests if it is the only machine directly connected to the Internet.

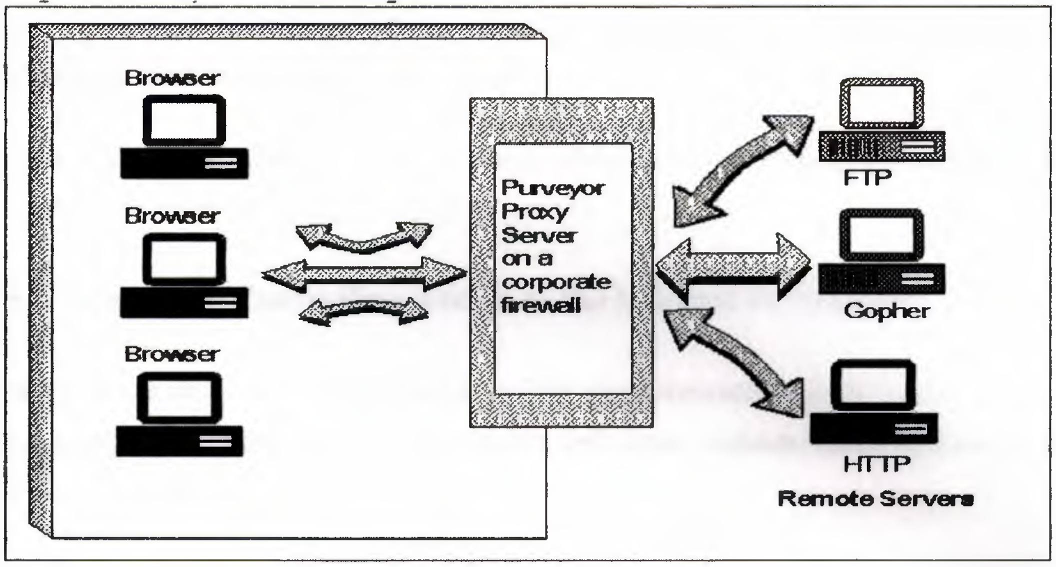


Figure 3.1 Proxy Server Running on a Firewall

3.4. Caching Documents

Usually, the clients within a subnet access the same Web proxy server. Some proxy servers let you cache internet documents for clients within the local area network. Caching documents means keeping a local copy of internet documents, so that the server doesn't need to request them over and over again.

Caching is more effective on the proxy server than on each client system. This saves disk space because only a single copy is cached. Caching on the proxy server means more documents that are often referenced by multiple browsers can be cached more efficiently. The system administrator can predict which documents are worth caching for a long time and which are not.

It is easy to configure an entire workgroup to use the proxy server's cache of documents. This reduces the load on the server by allowing it to get information from the cache when responding to subsequent client requests for the same data.

Caching also makes it possible to browse the Web even if a Web server, or even the external network, is down, as long as one can connect to the proxy server. This improves service to remote network resources, such as busy FTP sites and transient Gopher servers that are often unavailable remotely, but may be cached locally.

You can also cache a presentation you plan to present elsewhere when you are unsure of the location's Internet capabilities.[10]

3.5. Selectively Controlling Access to the Internet and Subnets

When using a proxy server it is possible to filter client transactions at the protocol level. The proxy can control access to services for individual methods, hosts, and domains. Some proxy servers let you:

- Decide which requests to honor and which requests to turn down.
- Specify the URLs or URL masks of locations that you do not want the proxy server to serve.
- Specify which protocols clients can use based on their IP addresses. For example, you could let certain clients make HTTP requests but not let them use FTP.

3.5.1. Configuring Browsers to Use the Proxy Server

For a browser to use a proxy server they must channel their internet requests through the proxy server. Most browsers allow you to configure them so that they direct their requests through a proxy server. Depending on the browser, you can identify a proxy server by identifying the server's domain name or IP address. However, unless you configure the browsers individually on your subnet to look for the proxy server, they won't send their requests to it.

Providing Internet Access for Companies Using Private Networks:

Organizations that use one or more private network address spaces, such as class A 10.*.*.*, can still use the Internet. To access the Internet they need to have a proxy server that is visible to the Internet and to the private internal network(s).

An Ordinary Web Transaction Via a Server

Many clients have their own IP address and a direct connection to servers on the Internet. When a normal HTTP request is made by the browser, the HTTP server gets only the path and keyword portion of the requested URL. Other parts of the URL, such as the protocol specifier "http:" and the host name, are clear to the remote HTTP server. The remote server knows that it is an HTTP server, and it knows the host machine that on which it is running (see Figure 26). The requested path specifies the document or a CGI program on the local filesystem of the server, or some other resource available from that server.[11]

When a user enters:

http://mycompany.com/information/ProxyDetails.html

The browser converts it to:

GET /information/ProxyDetails.html

The browser connects to the server running on mycompany.com and issues the command and waits for a response. In this example, the browser makes a request to the HTTP server and specifies the requested resource relative to that server. there is no protocol nor host name specifier in the URL.

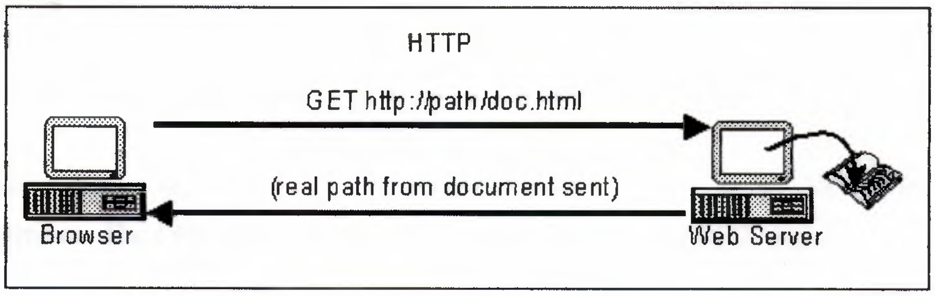


Figure 3.2 A Normal Web Transaction

The request specified the path (Data Directory) of information and the ProxyDetails.html document located in the Data Directory. The response is a document or an error message. The user in this example could just as easily use FTP:// ----- in which case the client sends the request to the specified FTP server.

- **Communication Via a Proxy Server**

The proxy server acts as both a server system and a client system. It is a server when accepting HTTP requests from browsers, and acts as a client system when its browser software connects to remote servers to retrieve documents.

The proxy server uses the header fields passed to it by the browser without modification when it connects to the remote server. This means the browser does not lose any functionality when going through a proxy.

A complete proxy server should be able to communicate all the Web protocols, the most important ones being HTTP, FTP, Gopher, and WAIS. Proxies that handle only a single Internet protocol, such as HTTP, are possible, but a Web browser would then require access to other proxy servers to handle the remaining protocols.

When a browser sends a request through a proxy server, the browser always uses HTTP for the transactions with the proxy server. This is true even when the user wants to access a remote server that uses another protocol. for example, FTP.

Instead of specifying only the pathname and search keywords to the proxy server, the browser specifies the full URL. This way the proxy server has all the information necessary to make the actual request to the remote server specified in the request URL, using the protocol specified in the URL.

3.5.2. HTTP Browser Request to Remote HTTP Transaction

When you use a proxy server as a client system, it acts as a browser to receive documents. The following is a typical example of a proxied HTTP request:

When you enter a full URL, for example:
http://mycompany.com/information/ProxyDetails.html

The browser converts the URL to:

    GET http://mycompany.com/information/ProxyDetails.html

The browser then connects to the server, and then the proxy server provides the connection to the Internet.

The proxy server converts this request to:

GET /information/ProxyDetails.html

The proxy server connects to the server running on mycompany.com. The server then issues the command and waits for a response, returns the response to the proxy server, which then returns the response to the client.[11]

Figure 3.2 shows a browser making a request to the proxy server using HTTP and specifying a full URL. The figure shows that the URL passed between the proxy server and the remote server specifies neither the remote host name nor the HTTP protocol.

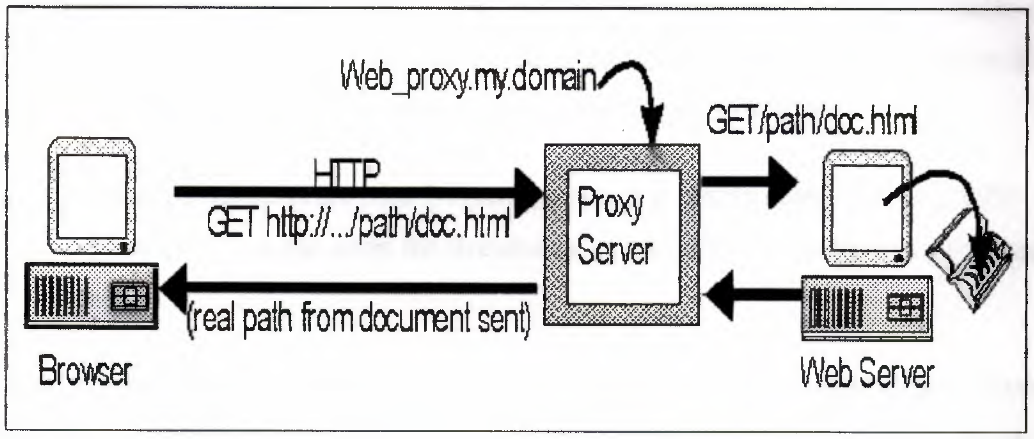


Figure 3.3 An HTTP Transaction via a Proxy Server

- **HTTP Browser Request to Remote FTP Transaction**

Figuer 3.4 shows a browser request via a proxy server using HTTP even though the request specifies a document on an FTP server on the Internet. The proxy server sees from the full URL that it should make an FTP connection. The proxy server makes the connection and retrieves the file from the remote FTP server and sends it to the browser using HTTP. In this case, the proxy server returns an FTP directory listing as an HTML document.

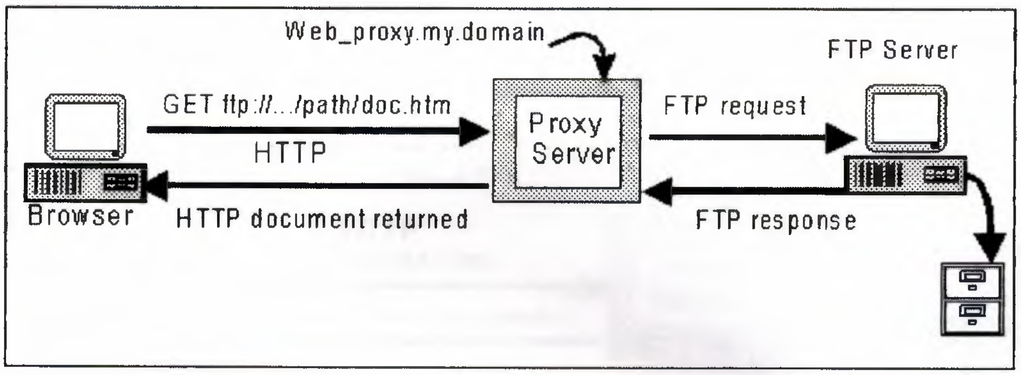


Figure 3.4 An FTP Transaction via a Proxy Server

3.5.3. Advantages and Disadvantages of Caching Documents

Caching documents means storing documents locally so users do not have to connect to a remote server to get files. When a local browser requests a file, the server checks its cache to see if it has the document. If the file exists in the cache, the server serves the local copy to the browser. If you cache documents you need to decide:

- Which documents are used frequently enough to justify keeping them locally
- How long you can keep the documents in cache before fetching more recent copies.

Figure 3.5 shows a proxy server caching a document retrieved from a remote server. The client (or other clients) can request and receive this locally stored document at a later time.

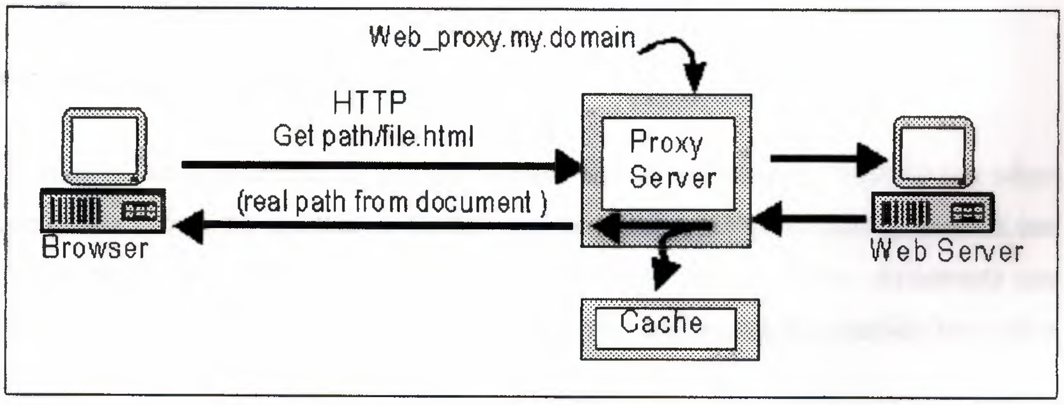


Figure 3.5 Caching Documents on a Proxy Server

If an up-to-date version of the requested document is found in the cache of the proxy server no connection to the remote server is necessary as shown in Figure 3.6

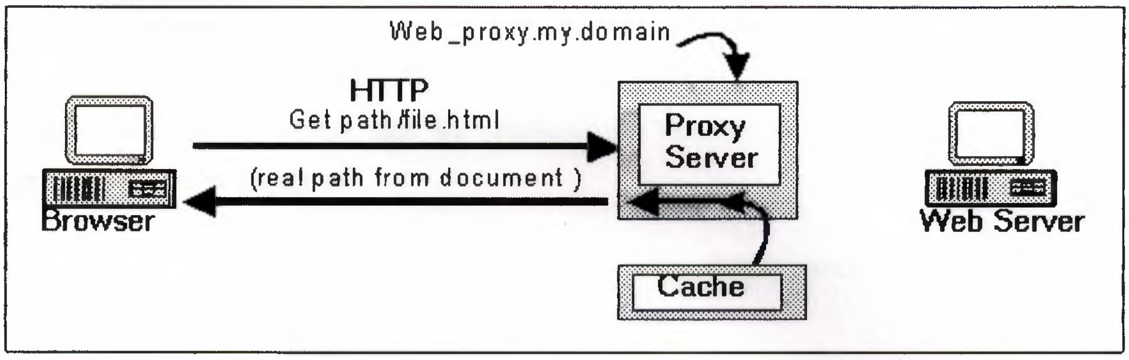


Figure 3.6 Retrieving Cached Documents

3.5.4. Advantages of Caching on a Proxy Server

Caching documents can save users considerable time when they request documents normally located out on the Internet. A proxy server can serve these documents much more quickly than remote servers. In addition, caching a document that many users need can save considerable network cost and connection time. Caching can also reduce the amount of disk space browsers use because many local browsers can use a single copy of a cached document.

Caching is disk based. when you restart the server, documents that you cache are still available. If you want, you can also configure the proxy server to use only the local cache. For instance, you can provide Internet documents to local browsers that do not have an internet connection.

3.5.5. Managing Cached Documents

Many documents available on the Internet are "living" documents. Determining when documents should be updated or deleted can be a difficult task. Some documents can remain stable for a very long time and then suddenly change. Other documents can change on a weekly or a daily basis. This means you need to decide carefully how often to refresh or delete the documents held in cache.

3.5.6. Proxy Server-to-Proxy Server Linking

Chaining proxy servers lets you run a proxy server as a local cache on behalf of a department within an organization. The individual departments have control over the server and cache. These departmental proxy servers can connect to a proxy server on a firewall between the Internet and the organization. This proxy server talks to the Internet as shown in Figure 3.7.

Any restrictions for access set for the organization proxy server take precedence over access restrictions set for the departmental proxy servers.

For example, departmental proxy server 1 might be set to allow all URL requests. The organizational proxy server, as corporate policy, might be set to deny all URL requests for certain online publications. A request for one of these publications coming into proxy server 1 would be forwarded to the organizational proxy server. The organizational proxy server would then deny the request.

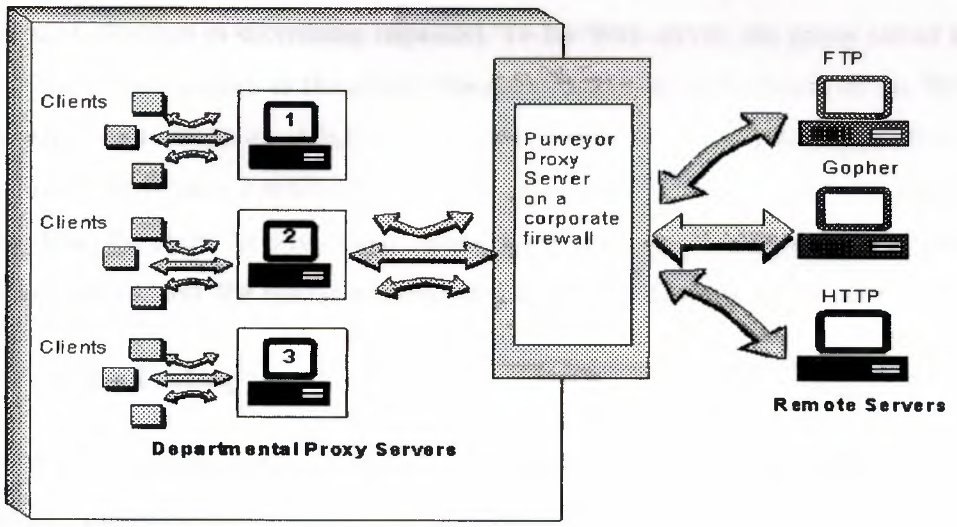

Clients
1
Clients
2
Clients
3
Departmental Proxy Servers
Purveyor Proxy Server on a corporate firewall
FTP
Gopher
HTTP
Remote Servers

Figure 3.7 Proxy Linking

Conversely, proxy server 1 could be configured to deny URLs going to a designated FTP site while proxy server 2 and 3 and the organizational server are all allowed access to the site.

3.6. Types of Caching

There are four types of caching:

3.6.1. Passive Caching

With passive caching (also called basic or on-demand caching), the client (browser) sends a request directly to a proxy server, an HTTP server that usually runs on a firewall server. The proxy server locates the object in its cache and returns the object to the client. If the object is not in the cache, the proxy retrieves a copy from the origin Web server on the Internet, stores it in the cache on the proxy server, and returns a copy of the object to the client. The object is cached for a preset period of time or until the cache is full. If the cache disk space is low, older objects are removed from the cache. Subsequent browser requests for the cached object are made to the proxy server at local intranet speeds. This reduces Internet traffic and the request load on the source Web server, thereby reducing the delays in returning information to the client.

To the client, the proxy server has the same basic functionality as the Web server (with a subtle difference in submitting requests). To the Web server, the proxy server has the same basic functionality as the client. The proxy builds its cache based on the Web sites that users visit. When an object is retrieved from the Web and put in a cache, a Time-To-Live (TTL) value is associated with the object. Before the TTL expires, requests are filled from the cache for that object. When the TTL expires, the Web server is contacted for a newer version, the update is stored in the cache, and a new TTL is calculated.[12]

3.6.2. Active Caching

Active caching is an add-on to passive caching that improves performance. With active caching, the proxy automatically sends a request to the origin server to retrieve an object. The server updates objects that are more frequently accessed or requested, have longer TTLs, and are actively cached during periods of low server load.

3.6.3. Negative Caching

Negative caching occurs when a proxy attempts to resolve a request for a URL that does not exist or cannot be located or accessed. In this case, the proxy caches the negative result so that future requests for that URL are resolved quickly. The proxy continues to check in the background and refreshes the cache when the pages become available. Negative caching occurs for HTTP error conditions such as 403 (forbidden request) and 404 (URL not found).

3.6.4. Hierarchical Caching

Hierarchical caching allows information to be retrieved from the nearby or closest proxy servers instead of from the originating Web server. HTTP and FTP acceleration (reverse proxy cache acceleration) also allows static information to be cached by and retrieved from the border proxy servers instead of the origin Web servers to reduce the Web server load. The proxy cache uses cache aging information that Web servers provide to browsers to determine how long pages should be cached.[12]

3.7. Interaction with Other BorderManager Services

Access control is issued by the Proxy Services software applications to forward and filter connections for such services as HTTP, Gopher, and FTP. The host running Proxy Services is known as the gateway. In general, Proxy Services allows services only for which there are proxies. For example, if a gateway has proxies for FTP, then only FTP can be requested. requests for all other services are ignored.

With gateways, you can hide the names and addresses of internal systems---the gateway is the only hostname known outside the system. Also, traffic can be logged before it reaches the internal hosts. Proxy Services improves security by hiding private network domain names and addresses and sending all requests through a single gateway. For more information about gateways.[12]

3.8. Proxy Technology

Proxy Services is based on both the first-generation CERN proxy technology and the newer, second-generation Harvest/Squid hierarchical proxy cache technology. The Harvest/Squid technology enhances standard CERN proxy cache services with negative URL caching and negative Domain Name System (DNS) caching, and introduces hierarchical caching through the Internet Cache Protocol (ICP). The Harvest project, an Internet Resource Discovery Project contract performed by the University of Colorado, introduced ICP hierarchical caching to improve Internet Web performance and scalability. The project was transferred to the National Laboratory for Applied Network Research (NLANR) in early 1996 as the basis for the Squid project. The goal of the Squid project is to facilitate the evolution of an efficient national architecture for handling highly popular information.[13]

3.9. Supported Protocols

BorderManager Proxy Services supports the following protocols and applications:

- HTTP (0.9, 1.0, and 1.1), including HTTPS support and Secure Sockets Layer (SSL)
- FTP
- Domain Name System (DNS)
- Gopher
- Simple Mail Transfer Protocol/Post Office Protocol 3 (SMTP/POP3)
- Network News Transfer Protocol (NNTP)
- RealAudio and RealVideo*
- SOCKS 4 and 5
- Generic TCP/UDP
- Transparent proxy

The passive mode (PASV) is supported for FTP to allow the firewall administrator to deny incoming connections above port 1023, if necessary. Otherwise, normal (PORT) FTP mode is used. Proxy Services also supports the HTTP protocol over the Internetwork Packet Exchange^{TM} (IPX^{TM}) software. Novell IPX/IP and IP/IP gateway clients, as well as other clients, can directly access the proxy server using the gateway client transparent proxy feature. For more information about the Novell IP Gateway.

3.10. Proxy Services Benefits

BorderManager Proxy Services combines an Internet proxy, a Web caching facility, and the NDS^{TM} software to provide World Wide Web access from within a firewall. Proxy Services has the following benefits:

- Reduces WAN traffic to the Internet and on the primary Web server by providing local LAN access to cached information. Proxy Services also reduces the load on Web Internet servers and increases Internet and intranet performance.
- Uses a single protocol on the LAN (for HTTP proxy only). Users do not need to have separate clients-HTTP is used to communicate with a proxy server. The proxy server uses the appropriate protocol-FTP, Gopher, and so on-for HTTP requests to access documents from the network.
- Improves intranet security by hiding the local network from the Internet. Private network domain names and addresses are hidden and all requests are sent through a single gateway. This applies to forward proxy only. Reverse proxy is used to hide the origin host from the client or local network.
- Enhances intranet security with access control and content filtering.
- Distributes LAN client requests across multiple proxy servers, for example, FTP requests on one server and HTTP requests on another server.
- Reduces the disk space requirements for retrieved information on client workstations and reduces the load on Web Internet servers.
- Enables document access even when the Internet or intranet Web server is down or inaccessible, if the document is already cached by the proxy server and Time-To-Live is not expired.
- Undeletes and serves if the origin server is down.
- Provides a single point of NDS-based administration.
- Logs and filters client transactions.

These benefits apply to both Internet and intranet Web sites. Because Proxy Services supports open Internet standards, it can be used with Novell's intranet and Internet products, as well as with other vendors' browsers and Web servers.[13]

3.11. Application Proxies

This section describes in detail the following supported application proxies:

3.11.1. HTTP Proxy

There are two types of HTTP proxy:

- HTTP proxy (forward proxy)
- HTTP accelerator (reverse proxy)

3.11.1.1. HTTP or Forward Proxy

HTTP proxy resolves URL requests on behalf of Web clients on your network. This is also known as forward proxy. These requests are cached, if possible, on the proxy server to increase the speed of delivering the same content the next time the same information is requested.

HTTP itself is an application-level protocol used for distributed, collaborative, hypermedia information systems. It is generic and allows systems to be created independently of the data being sent. It is also an object-oriented protocol that can be used for name servers, distributed object management systems, and so on. HTTP servers use HTTP as the primary application protocol, allowing users to access and exchange Web files. The HTTP protocol can also be used for communication between users, proxies, gateways, and other Internet protocols, such as SMTP, NNTP, FTP, and Gopher.

HTTP communication is usually over TCP/IP connections, on default port 80, although other ports can be used.

3.11.1.2. HTTP Accelerator or Reverse Proxy

The proxy server can be configured as an HTTP accelerator to protect an intranet server from the Internet and reduce the load on the public Web servers maintained on the intranet. HTTP acceleration, also known as reverse proxy cache acceleration or Web server acceleration, creates a front-end processor to a Web server. An HTTP accelerator server lies between one or more Web servers and the Internet and represents the Web servers to any clients accessing them. An HTTP accelerator can also be used to create a local mirror site of a remote server.

When the Internet user queries DNS for the Web server address, it returns the address of the requested Web server. The HTTP accelerator listens for HTTP requests on port 80 (or another configured port) and processes all incoming Web requests. Requests for objects that can be cached-static information that does not change often, such as HTML pages and GIF images-are processed by the proxy. Requests for objects that cannot be cached---dynamic information that changes frequently-are processed by the origin Web server on port 80. In general, approximately 90 percent of a typical Web server content is static and 10 percent is dynamic.

You can set up an HTTP accelerator server to retrieve information or references to cachable objects from a Web server and cache the information on a BorderManager server. This reduces loading on the Web server. The HTTP accelerator server forwards only requests and references that are not in the cache to the Web server.

If your site receives requests for a high percentage of objects that can be cached, the HTTP accelerator reduces the Web load. For even greater performance, you can cache objects of a more volatile nature, such as stock quotes, and specify an accuracy delay time to users.

BorderManager reverse proxy can handle more TCP connections than an origin Web server (typically UNIX or Windows NT).[14]

HTTP acceleration has the following benefits:

- Provides caching for Web servers

- Reduces the load on the Web servers and speeds them up
- Protects Web servers
- Protects IP networks in conjunction with the other BorderManager services

3.11.2. FTP Proxy

There are two types of FTP proxy:

3.11.2.1. Benefits of FTP Proxy

FTP is the standard Internet protocol used for file transfer. FTP proxy is used to proxy FTP requests when users use pure FTP clients, for example, the LAN WorkPlace® software, UNIX, Macintosh, and so on.

FTP proxy has the following benefits:

- Centralized access control
- Data caching for FTP data files
- Anonymous users allowed
- URL representation of FTP data

Standard FTP requires a user account on the server being accessed. Anonymous FTP does not require a user account and provides access to specific files on the Internet. The username is "anonymous" or "ftp". You can use proxy servers to control access to authenticated FTP sites. When an FTP proxy server is placed on a firewall, all FTP client requests in the intranet must pass through the FTP proxy server. This helps enforce centralized control over Internet access and scans data that is being sent or retrieved by users within an organization. The FTP intranet client (or user) must first connect to the FTP proxy server by entering the IP address or name of the proxy server, for example, ftp://novell.com. The user must then enter the following to identify the origin host and connect to the FTP proxy:

USER ProxyUserName$DestFTPUserName$
DestFTPHostName

PASS UserNDSPassword$DestFTPPassword

where ProxyUserName is the NDS username, DestFTPUserName is the FTP username on the destination server, DestFTPHostName is the hostname or IP address of the destination FTP server, UserNDSPassword is the user's NDS password, and DestFTPPassword is the user password on the destination server. Only the FTP hostname DestFTPHostName is required. If the DestFTPUserName is missing, it is assumed to be anonymous, and no password is required. The ProxyUserName is required only if FTP authentication is enabled. The proxy makes the final connection to the origin host or server.[14]

Both active and passive FTP modes are supported, and can be enabled or disabled. Active mode (PORT) posts a listener on the intranet and allows clients to make a connection to the intranet machine, a less secure method. Passive mode (PASV) for FTP allows the client to initiate the connection to a remote FTP server. PASV mode is supported to allow the firewall administrator to deny incoming connections above port 1023, if necessary.

3.11.2.2. FTP Reverse Proxy

FTP reverse proxy, or FTP accelerator, is an application that is placed in front of the FTP server. The FTP accelerator acts as an FTP server to Internet users and protects the FTP servers behind the firewall from outside break-ins. The FTP accelerator scans inbound and outbound data, and with third-party support, can trap any viruses being sent through the system.

The FTP accelerator also caches frequently requested data and FTP files for anonymous users and helps accelerate FTP requests. This process is useful because most FTP requests from the Internet are from anonymous FTP users. Caching shifts the load from FTP servers to the reverse FTP proxy.

3.12. Mail (SMTP/POP3) Proxy

Electronic mail is the most fundamental and useful of Internet services. It is also the most vulnerable. To create a secure environment, you must be able to restrict access to outside mail to only a few machines, screen messages for hostile applets or scripts, and avoid other malicious e-mail schemes.

SMTP handles electronic mail exchange between mail servers, accepting mail and sending it directly to the destination mail domains or delivering it to an intermediate relay agent. Post Office Protocol 3 (POP3) is used to handle the user electronic mailboxes on servers.

The Mail proxy server provides secure SMTP mail services for incoming and outgoing mail. SMTP allows intranet users to send mail to the Internet in a secure manner. Similarly, Internet users can send mail through SMTP to intranet users in a secure manner.

SMTP proxy can perform the following access control and filtering for outgoing and incoming mail:

- Enforce access control based on usernames and mail domain names for incoming and outgoing mail.
- Hide internal mail domain names and usernames. The Mail proxy can be configured to overwrite the "From" address so that only the primary mail domain name for an organization is exposed to the Internet.
- Filter for Multimedia Internet Mail Extensions (MIME). Mail is scanned and filtered for attachments, including non-ASCII character sets, nontext data, rich text messages (with formatted text), and multipart messages.
- Scan and filter incoming e-mail, with third party support, for viruses and junk mail. Unwanted junk mail is scanned using access control lists that combine mail domain filtering and content filtering.

Mail proxy can be used in an organization between the existing intranet mail server and the Internet, or between the intranet and the Internet without an existing intranet mail server. The following e-mail commands are allowed by the Mail proxy: HELO, MAIL, RCPT, DATA, RSET, HELP, NOOP, and QUIT. For more information about configuring mail filters and access control,

3.13. News (NNTP) Proxy

The News, or NNTP, proxy is used for accessing and using Usenet news, an Internet bulletin-board-like feature that contains articles on many subjects. Articles are grouped into subjects or news groups. More than 10,000 public news groups exist on the Internet. The News proxy provides secure NNTP news services for transferring news postings or articles in both directions between the intranet and the Internet. The News proxy is a TCP-based service that uses a store-and-forward type of protocol.

Internal or private news servers can use the proxy to exchange articles with outside or public news servers in a secure manner. For public news servers, the News proxy acts as a corporate news server and feeds all configured private news servers, if any. For private news servers and news readers (for example, Netscape* Communicator*), the News proxy acts as a public news server and feeds all configured public news servers. The following news commands are allowed by the News proxy: POST, IHAVE, NEWNEWS, NEWGROUPS.

If an intranet has no private news servers, such as in a small company, the News proxy acts as a news server. All user requests for listings of groups, articles, and retrieval and posting of articles are sent by the browsers or news readers to the News proxy. The News proxy then sends the requests and information to the configured public news servers and forwards all responses back to the users. The news reader utilities sort the articles or groups and display the information to the users. No articles are cached in this version of the News proxy.

Users can retrieve news articles by either specifying the article ID or selecting a group and the article number. All commands for retrieving news articles are supported. You should dedicate a server to the News proxy and services because they tend to consume disk space quickly. News proxy performance is optimal when internal news servers are available. This reduces the request load on the News proxy.

You can apply access control rules to the News proxy by specifying the following:

- A list of public or private news groups that can be allowed or denied. For public groups, the allowed groups can be seen by the news readers and private news servers. For private groups, the allowed groups can be seen by the public news servers only.

- An article that can be allowed or denied posting to a news group by a user. The user can be specified as ANY to apply the rule to all users or a list of news groups.

3.14. DNS Proxy

DNS is a distributed data system that translates hostnames to IP addresses and vice versa. DNS also stores and accesses other information about hosts.

When enabled, the DNS proxy acts as a DNS server for clients on the intranet. A listener is posted on the DNS port. When a DNS request is received from a client, the DNS proxy checks its local DNS cache and returns a response, if available. If the address is not in the cache, the DNS proxy forwards the request to the configured DNS name servers. The proxy caches only the responses of Internet class and Internet address queries.

The client must have the private IP address of the DNS proxy configured as the address of its DNS server.

On the server, you can set up the IP addresses of the DNS name servers and the domain name in the SYS:\ETC\RESOLV.CFG file.

3.15. HTTPS Proxy

HTTPS proxy provides the ability to access secure sites using SSL over a persistent IP connection. The browser sends an HTTPS request as an SSL request through the proxy, which then tunnels the request to the origin Web server.

3.16. SOCKS Client

This features enables a proxy to authenticate through a SOCKS 5 firewall. This release also supports the forwarding of HTTP traffic only.

SOCKS is a circuit-level gateway protocol. With SOCKS, hosts behind a firewall can gain full access to the Internet without full IP support. When SOCKS support is enabled, all requests sent to the Internet are forwarded to a SOCKS 5 server when the proxy is used for caching only.

When the proxy receives a request, it checks its cache. If the requested object is not in the cache, the proxy makes a TCP connection to the SOCKS server and redirects the request from the intranet to the SOCKS server, allowing for more secure Internet access. The SOCKS server then connects to the origin server and retrieves the object. The proxy simply acts as a SOCKS client to the SOCKS server and is used for caching only. Null (no username or password) and username/password authentication are supported. The Novell IP Gateway can also support the proxy as a SOCKS client.

This release requires that the proxy server and the SOCKS server are both on the same intranet. The reason is that in the username/password combination, SOCKS authentication uses clear text to send the password.

3.17. Generic Proxy

Generic proxy is a circuit-level, pass-through proxy used to serve multiple protocols when an application proxy is not available. A mapping is created between the address and ports, creating a tunnel to the destination host. When the generic proxy server receives a connection request from the intranet, it forwards the request to the mapped address, connects to it, and transfers data between the two connections.

For example, to establish a Telnet connection to an internal host from home, a generic TCP proxy should be set up at the proxy server. You can also define a generic UDP proxy. When connecting to the proxy, the user is connected to the internal host. Authentication is available for generic TCP proxy---a user must be authenticated using access control list rules before connecting to a remote host. Authentication is not available for generic UDP proxy.

You can apply access control rules to generic TCP proxy. Access can be allowed or denied based on the following:

- The IP address or hostname of the original host
- The port number associated with the origin host

- The IP address of the source host in the intranet

3.18. HTTP Transparent Proxy

Transparent proxy can be implemented using either of the following features:

3.18.1. Gateway Client Transparent Proxy

If the client is not configured to use a specific proxy or is not set up to use the HTTP Transparent proxy feature of Proxy Services, the Novell IP Gateway client will enforce the use of a proxy by capturing the browser request and redirecting it to an active proxy, which the client finds through NDS. During initialization, if the gateway client transparent proxy is enabled, the gateway client uses NDS to find active HTTP proxy servers and sends the request to the first proxy server found that the user has permission to access.[14]

3.18.2. HTTP Transparent Proxy

HTTP Transparent proxy enables users to use their Web browsers without having to specifically reconfigure each browser to point to a proxy. This feature is useful if you have limited time and cannot immediately reconfigure the browsers for all your users. It is also useful when you want to enforce network security and ensure that all client requests pass through a proxy.

The HTTP Transparent proxy intercepts traffic between the client and the origin Web server, and funnels it to a proxy server. Relative URLs are translated to absolute URLs. For HTTP Transparent proxy only, traffic from a configurable list of ports or IP addresses is intercepted. Only the ports or addresses on the list participate in forwarding traffic to the proxy.

To use HTTP Transparent proxy, you must ensure that all HTTP requests are sent through the proxy server. Therefore, the proxy server must be the default router or provide the only access to the Internet. The clients must use the proxy's private IP address as the TCP/IP gateway address. IP forwarding must be enabled on the server.

3.19. Client servers

3.19.1. Client/Server Networking

Ages ago (in Internet time), when mainframe dinosaurs roamed the Earth, a new approach to computer networking called "client/server" emerged. Client/server proved to be a more cost-effective way to build many types of networks, particularly PC-based LANs running end-user database applications. Many types of client/server systems remain popular today.

3.19.2. What Is Client/Server?

The most basic definition of client/server comes from the corresponding Usenet FAQ

Client/server is a computational architecture that involves client processes requesting service from server processes.

In general, client/server maintains a distinction between processes and network devices. Usually a client computer and a server computer are two separate devices, each customized for their designed purpose. For example, a Web server will often contain large amounts of memory and disk space, whereas Web clients often include features to support the graphic user interface of the browser such as high-end video cards and large-screen displays. Client/server networking, however, focuses primarily on the applications rather than the hardware. The same device may function as both client and server. for example, Web server hardware functions as both client and server when local browser sessions are run there. Likewise, a device that is a server at one moment can reverse roles and become a client to a different server (either for the same application or for a different application).[16]

3.19.3. Client/Server Applications

Some of the most popular applications on the Internet follow the client/server design:

- Email clients
- FTP (File transfer) clients
- Web browsers

Each of these programs presents a user interface (either graphic- or text-based) in a client process, that allows the user to connect to servers. In the case of email and FTP, the user enters a computer name (or sometimes an IP address) into the interface to set up future connections to the server process.

For example, an Earthlink subscriber enters the name smtp.earthlink.net into the configuration settings of their email client to allow them to send messages over the Internet. In the case of email, a person generally enters the server information only one time, as the server side of the connection rarely changes. In the case of FTP, however, one typically enters a different server name each time they use the program. One day a person may visit ftp.earthlink.net to download tools, the next day they may visit ftp.microsoft.com to find a software patch, and so on. When using a Web browser, the name or address of the server appears in the URL of each request. Although a person may start a Web surfing session by entering a particular server name (such as www.about.com), the name regularly changes as they click links on the pages. In the Web model, server information is provided by the HTML content developer encoded in the anchor tags.

3.19.4. Client/Server at Home

Many home networkers use client/server systems without even realizing it. Microsoft's Internet Connection Sharing (ICS), for example, relies on DHCP server and client functionality built into the operating system. Cable modem and DSL routers like those from Linksys also include a DHCP server with the hardware unit. Many home LAN gaming applications also use a single-server/multiple-client configuration.

3.19.5. Pros and Cons of Client/Server

Client/server was originally developed to allow more users to share access to database applications. Compared to the mainframe approach, client/server offers improved scalability because connections can be made as needed rather than being hard-wired. The client/server model also supports modular applications. In the so-called "two-tier" and "three-tier" types of client/server systems, a software application is separated into modular pieces, and each piece is installed on hardware specialized for that subsystem.

One area of special concern in client/server networking is system management. With applications distributed across the network, it can be challenging to keep configuration information up-to-date and consistent among all of the devices. Likewise, upgrades to a newer version of a client/server application can be difficult to synchronize or stage appropriately. Finally, client/server systems rely heavily on the network's reliability. redundancy or fail-over features can be expensive to implement.[16]

3.20. Summary

The primary use of a proxy server is to allow internal clients access to the Internet from behind a firewall. The proxy technology, which has developed greatly in the last years, offers the perfect solution for organizations sitting on a closed subnet behind a firewall, who are interested in giving their employees a controlled access to the Internet. A proxy is in fact, an http server that sits on a firewall machine and usually has a caching ability making surfing much faster.

This ability makes it attractive also in case there is no firewall. It also allows one to read documents "unplugged" to the Internet. This caching proxy functions as a server when connected by a client, and as a client when contacting the original server. Today's proxies are very sophisticated, so that the security "holes" are minimal.

4. PUBLIC-KEY CRYPTOGRAPHY AND SECURE SOCKETS LAYER

4.1. Overview

Public-key cryptography and related standards and techniques underlie security features of many Netscape products, including signed and encrypted email, form signing, object signing, single sign-on, and the Secure Sockets Layer (SSL) protocol. This chapter introduces the basic concepts of public-key cryptography.

4.2. Internet Security Issues

All communication over the Internet uses the Transmission Control Protocol/Internet Protocol (TCP/IP). TCP/IP allows information to be sent from one computer to another through a variety of intermediate computers and separate networks before it reaches its destination.[20]

The great flexibility of TCP/IP has led to its worldwide acceptance as the basic Internet and intranet communications protocol. At the same time, the fact that TCP/IP allows information to pass through intermediate computers makes it possible for a third party to interfere with communications in the following ways:

- **Eavesdropping.** Information remains intact, but its privacy is compromised. For example, someone could learn your credit card number, record a sensitive conversation, or intercept classified information.
- **Tampering.** Information in transit is changed or replaced and then sent on to the recipient. For example, someone could alter an order for goods or change a person's resume.
- **Impersonation.** Information passes to a person who poses as the intended recipient. Impersonation can take two forms:
    - **Spoofing.** A person can pretend to be someone else. For example, a person can pretend to have the email address jdoe@mozilla.com, or a computer can identify itself as a site called www.mozilla.com when it is not. This type of impersonation is known as spoofing.

- **Misrepresentation.** A person or organization can misrepresent itself. For example, suppose the site www.mozilla.com pretends to be a furniture store when it is really just a site that takes credit-card payments but never sends any goods.

Normally, users of the many cooperating computers that make up the Internet or other networks don't monitor or interfere with the network traffic that continuously passes through their machines. However, many sensitive personal and business communications over the Internet require precautions that address the threats listed above. Fortunately, a set of well-established techniques and standards known as **public-key cryptography** make it relatively easy to take such precautions.
Public-key cryptography facilitates the following tasks:

- **Encryption and decryption** allow two communicating parties to disguise information they send to each other. The sender encrypts, or scrambles, information before sending it. The receiver decrypts, or unscrambles, the information after receiving it. While in transit, the encrypted information is unintelligible to an intruder.
- **Tamper detection** allows the recipient of information to verify that it has not been modified in transit. Any attempt to modify data or substitute a false message for a legitimate one will be detected.
- **Authentication** allows the recipient of information to determine its origin--that is, to confirm the sender's identity.
- **Nonrepudiation** prevents the sender of information from claiming at a later date that the information was never sent.

The chapters that follow introduce the concepts of public-key cryptography that underlie these capabilities.

4.3. Encryption and Decryption

Encryption is the process of transforming information so it is unintelligible to anyone but the intended recipient. Decryption is the process of transforming encrypted information so that it is intelligible again. A cryptographic algorithm, also called a

cipher, is a mathematical function used for encryption or decryption. In most cases, two related functions are employed, one for encryption and the other for decryption.[21]
With most modern cryptography, the ability to keep encrypted information secret is based not on the cryptographic algorithm, which is widely known, but on a number called a key that must be used with the algorithm to produce an encrypted result or to decrypt previously encrypted information. Decryption with the correct key is simple. Decryption without the correct key is very difficult, and in some cases impossible for all practical purposes.
The chapters that follow introduce the use of keys for encryption and decryption.

4.3.1. Symmetric-Key Encryption

With symmetric-key encryption, the encryption key can be calculated from the decryption key and vice versa. With most symmetric algorithms, the same key is used for both encryption and decryption, as shown in Figure 1.

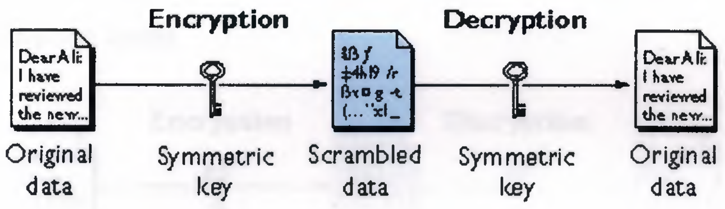


Figure 4.1 Symmetric-key encryption

Implementations of symmetric-key encryption can be highly efficient, so that users do not experience any significant time delay as a result of the encryption and decryption. Symmetric-key encryption also provides a degree of authentication, since information encrypted with one symmetric key cannot be decrypted with any other symmetric key. Thus, as long as the symmetric key is kept secret by the two parties using it to encrypt communications, each party can be sure that it is communicating with the other as long as the decrypted messages continue to make sense.
Symmetric-key encryption is effective only if the symmetric key is kept secret by the two parties involved. If anyone else discovers the key, it affects both confidentiality and authentication. A person with an unauthorized symmetric key not only can decrypt

messages sent with that key, but can encrypt new messages and send them as if they came from one of the two parties who were originally using the key.
Symmetric-key encryption plays an important role in the SSL protocol, which is widely used for authentication, tamper detection, and encryption over TCP/IP networks. SSL also uses techniques of public-key encryption, which is described in the next chapter.

4.3.2. Public-Key Encryption

The most commonly used implementations of public-key encryption are based on algorithms patented by RSA Data Security. Therefore, this chapter describes the RSA approach to public-key encryption.
Public-key encryption (also called asymmetric encryption) involves a pair of keys--a public key and a private key--associated with an entity that needs to authenticate its identity electronically or to sign or encrypt data. Each public key is published, and the corresponding private key is kept secret. Data encrypted with your public key can be decrypted only with your private key. Figure 2 shows a simplified view of the way public-key encryption works.

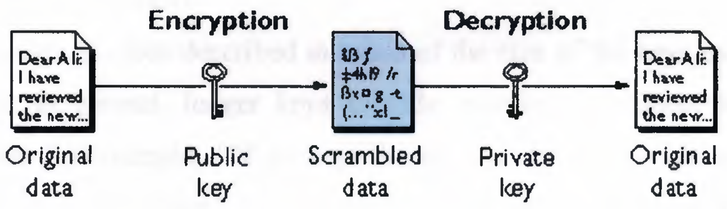


Figure 4.2 Public-key encryption

The scheme shown in Figure 4.2 lets you freely distribute a public key, and only you will be able to read data encrypted using this key. In general, to send encrypted data to someone, you encrypt the data with that person's public key, and the person receiving the encrypted data decrypts it with the corresponding private key.
Compared with symmetric-key encryption, public-key encryption requires more computation and is therefore not always appropriate for large amounts of data. However, it's possible to use public-key encryption to send a symmetric key, which can then be used to encrypt additional data. This is the approach used by the SSL protocol.

As it happens, the reverse of the scheme shown in Figure 4.2 also works: data encrypted with your private key can be decrypted only with your public key. This would not be a desirable way to encrypt sensitive data, however, because it means that anyone with your public key, which is by definition published, could decrypt the data. Nevertheless, private-key encryption is useful, because it means you can use your private key to sign data with your digital signature--an important requirement for electronic commerce and other commercial applications of cryptography. Client software such as Communicator can then use your public key to confirm that the message was signed with your private key and that it hasn't been tampered with since being signed. Digital Signatures and subsequent chapters describe how this confirmation process works.

4.4. Key Length and Encryption Strength

In general, the strength of encryption is related to the difficulty of discovering the key, which in turn depends on both the cipher used and the length of the key. For example, the difficulty of discovering the key for the RSA cipher most commonly used for public-key encryption depends on the difficulty of factoring large numbers, a well-known mathematical problem.

Encryption strength is often described in terms of the size of the keys used to perform the encryption: in general, longer keys provide stronger encryption. Key length is measured in bits. For example, 128-bit keys for use with the RC4 symmetric-key cipher supported by SSL provide significantly better cryptographic protection than 40-bit keys for use with the same cipher. Roughly speaking, 128-bit RC4 encryption is 3×10^{26} times stronger than 40-bit RC4 encryption.

Different ciphers may require different key lengths to achieve the same level of encryption strength. The RSA cipher used for public-key encryption, for example, can use only a subset of all possible values for a key of a given length, due to the nature of the mathematical problem on which it is based. Other ciphers, such as those used for symmetric key encryption, can use all possible values for a key of a given length, rather than a subset of those values. Thus a 128-bit key for use with a symmetric-key encryption cipher would provide stronger encryption than a 128-bit key for use with the RSA public-key encryption cipher.

This difference explains why the RSA public-key encryption cipher must use a 512-bit key (or longer) to be considered cryptographically strong, whereas symmetric key ciphers can achieve approximately the same level of strength with a 64-bit key. Even this level of strength may be vulnerable to attacks in the near future.[22]

Because the ability to surreptitiously intercept and decrypt encrypted information has historically been a significant military asset, the U.S. Government restricts export of cryptographic software, including most software that permits use of symmetric encryption keys longer than 40 bits. For detailed information about these restrictions as they apply to Netscape products, see Export Restrictions on International Sales.

4.5. Digital Signatures

Encryption and decryption address the problem of eavesdropping, one of the three Internet security issues mentioned at the beginning of this chapter. But encryption and decryption, by themselves, do not address the other two problems mentioned in Internet Security Issues: tampering and impersonation.

This chapter describes how public-key cryptography addresses the problem of tampering. The chapters that follow describe how it addresses the problem of impersonation.

Tamper detection and related authentication techniques rely on a mathematical function called a one-way hash (also called a message digest). A one-way hash is a number of fixed length with the following characteristics:

- The value of the hash is unique for the hashed data. Any change in the data, even deleting or altering a single character, results in a different value.
- The content of the hashed data cannot, for all practical purposes, be deduced from the hash--which is why it is called "one-way."

As mentioned in Public-Key Encryption, it's possible to use your private key for encryption and your public key for decryption. Although this is not desirable when you are encrypting sensitive information, it is a crucial part of digitally signing any data. Instead of encrypting the data itself, the signing software creates a one-way hash of the data, then uses your private key to encrypt the hash. The encrypted hash, along with other information, such as the hashing algorithm, is known as a digital signature.

Figure 4.3 shows a simplified view of the way a digital signature can be used to validate the integrity of signed data.

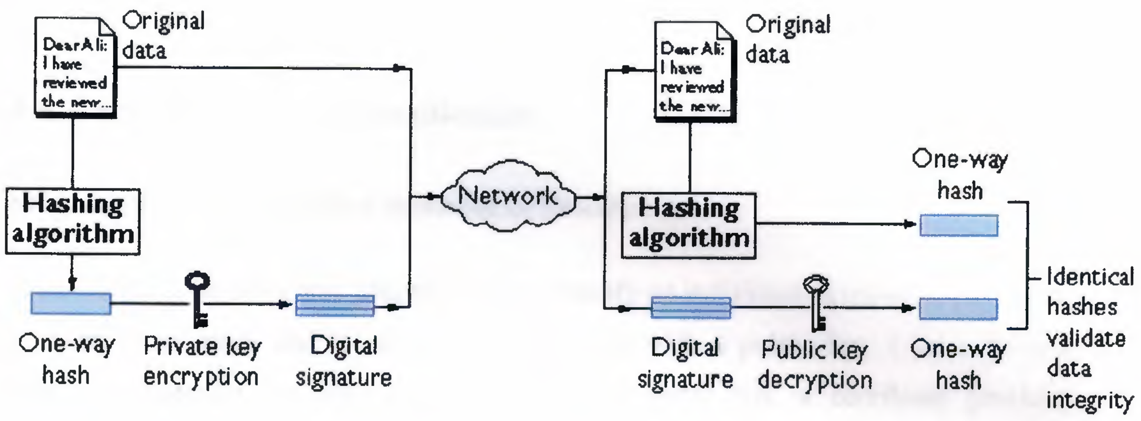


Figure 4.3 Using a digital signature to validate data integrity

Figure 4.3 shows two items transferred to the recipient of some signed data: the original data and the digital signature, which is basically a one-way hash (of the original data) that has been encrypted with the signer's private key. To validate the integrity of the data, the receiving software first uses the signer's public key to decrypt the hash. It then uses the same hashing algorithm that generated the original hash to generate a new one-way hash of the same data , the receiving software compares the new hash against the original hash. If the two hashes match, the data has not changed since it was signed. If they don't match, the data may have been tampered with since it was signed, or the signature may have been created with a private key that doesn't correspond to the public key presented by the signer.[22]

If the two hashes match, the recipient can be certain that the public key used to decrypt the digital signature corresponds to the private key used to create the digital signature. Confirming the identity of the signer, however, also requires some way of confirming that the public key really belongs to a particular person or other entity. For a discussion of the way this works, see Certificates and Authentication.

The significance of a digital signature is comparable to the significance of a handwritten signature. Once you have signed some data, it is difficult to deny doing so later--assuming that the private key has not been compromised or out of the owner's control. This quality of digital signatures provides a high degree of nonrepudiation--that is, digital signatures make it difficult for the signer to deny having signed the data. In some situations, a digital signature may be as legally binding as a handwritten signature.

4.6. Certificates and Authentication

4.6.1. A Certificate Identifies Someone or Something

A certificate is an electronic chapter used to identify an individual, a server, a company, or some other entity and to associate that identity with a public key. Like a driver's license, a passport, or other commonly used personal IDs, a certificate provides generally recognized proof of a person's identity. Public-key cryptography uses certificates to address the problem of impersonation (see Internet Security Issues).

To get a driver's license, you typically apply to a government agency, such as the Department of Motor Vehicles, which verifies your identity, your ability to drive, your address, and other information before issuing the license. To get a student ID, you apply to a school or college, which performs different checks (such as whether you have paid your tuition) before issuing the ID. To get a library card, you may need to provide only your name and a utility bill with your address on it.[23]

Certificates work much the same way as any of these familiar forms of identification. Certificate authorities (CAs) are entities that validate identities and issue certificates. They can be either independent third parties or organizations running their own certificate-issuing server software (such as Netscape Certificate Server). The methods used to validate an identity vary depending on the policies of a given CA-just as the methods to validate other forms of identification vary depending on who is issuing the ID and the purpose for which it will be used. In general, before issuing a certificate, the CA must use its published verification procedures for that type of certificate to ensure that an entity requesting a certificate is in fact who it claims to be.

The certificate issued by the CA binds a particular public key to the name of the entity the certificate identifies (such as the name of an employee or a server). Certificates help prevent the use of fake public keys for impersonation. Only the public key certified by the certificate will work with the corresponding private key possessed by the entity identified by the certificate.

In addition to a public key, a certificate always includes the name of the entity it identifies, an expiration date, the name of the CA that issued the certificate, a serial number, and other information. Most importantly, a certificate always includes the digital signature of the issuing CA. The CA's digital signature allows the certificate to function as a "letter of introduction" for users who know and trust the CA but don't know the entity identified by the certificate.

For more information about the role of CAs, see How CA Certificates Are Used to Establish Trust.

4.6.2. Authentication Confirms an Identity

Authentication is the process of confirming an identity. In the context of network interactions, authentication involves the confident identification of one party by another party. Authentication over networks can take many forms. Certificates are one way of supporting authentication.

Network interactions typically take place between a client, such as browser software running on a personal computer, and a server, such as the software and hardware used to host a Web site. Client authentication refers to the confident identification of a client by a server (that is, identification of the person assumed to be using the client software). Server authentication refers to the confident identification of a server by a client (that is, identification of the organization assumed to be responsible for the server at a particular *network address).*

Client and server authentication are not the only forms of authentication that certificates support. For example, the digital signature on an email message, combined with the certificate that identifies the sender, provide strong evidence that the person identified by that certificate did indeed send that message. Similarly, a digital signature on an HTML form, combined with a certificate that identifies the signer, can provide evidence, after the fact, that the person identified by that certificate did agree to the contents of the form. In addition to authentication, the digital signature in both cases ensures a degree of nonrepudiation--that is, a digital signature makes it difficult for the signer to claim later not to have sent the email or the form.

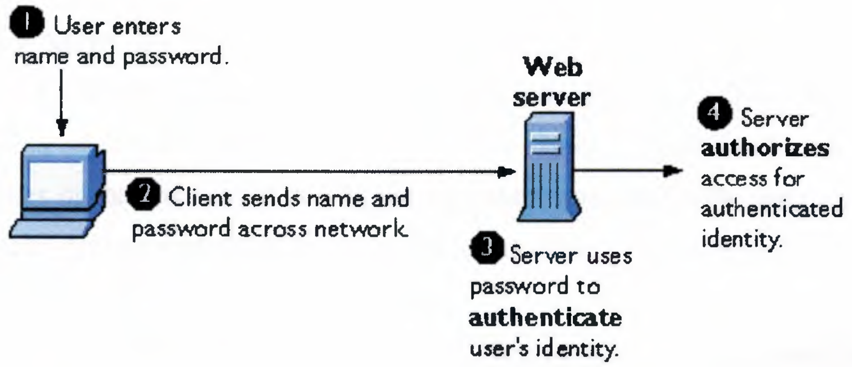
Client authentication is an essential element of network security within most intranets or extranets. The chapters that follow contrast two forms of client authentication:

- Password-Based Authentication. Almost all server software permits client authentication by means of a name and password. For example, a server might require a user to type a name and password before granting access to the server. The server maintains a list of names and passwords; if a particular name is on the list, and if the user types the correct password, the server grants access.
- Certificate-Based Authentication. Client authentication based on certificates is part of the SSL protocol. The client digitally signs a randomly generated piece of data and sends both the certificate and the signed data across the network. The server uses techniques of public-key cryptography to validate the signature and confirm the validity of the certificate.[24]

4.6.3. Password-Based Authentication

Figure 4.4 shows the basic steps involved in authenticating a client by means of a name and password. Figure 4 assumes the following:

- The user has already decided to trust the server, either without authentication or on the basis of server authentication via SSL.
- The user has requested a resource controlled by the server.
- The server requires client authentication before permitting access to the requested resource.



1 User enters name and password.
Web server
2 Client sends name and password across network.
3 Server uses password to authenticate user's identity.
4 Server authorizes access for authenticated identity.

Figure 4.4 Using a password to authenticate a client to a server

These are the steps shown in Figure 4.4:

1. In response to an authentication request from the server, the client displays a dialog box requesting the user's name and password for that server. The user must supply a name and password separately for each new server the user wishes to use during a work session.
2. The client sends the name and password across the network, either in the clear or over an encrypted SSL connection.
3. The server looks up the name and password in its local password database and, if they match, accepts them as evidence authenticating the user's identity.
4. The server determines whether the identified user is permitted to access the requested resource, and if so allows the client to access it.

With this arrangement, the user must supply a new password for each server, and the administrator must keep track of the name and password for each user, typically on separate servers.

As shown in the next chapter, one of the advantages of certificate-based authentication is that it can be used to replace the first three steps in Figure 4.4 with a mechanism that allows the user to supply just one password (which is not sent across the network) and allows the administrator to control user authentication centrally.

4.6.4. Certificate-Based Authentication

Figure 4.5 shows how client authentication works using certificates and the SSL Protocol. To authenticate a user to a server, a client digitally signs a randomly generated piece of data and sends both the certificate and the signed data across the network. For the purposes of this discussion, the digital signature associated with some data can be thought of as evidence provided by the client to the server. The server authenticates the user's identity on the strength of this evidence.

Like Figure 4.4, Figure 4.5 assumes that the user has already decided to trust the server and has requested a resource, and that the server has requested client authentication in the process of evaluating whether to grant access to the requested resource. Unlike the process shown in Figure 4.4, the process shown in Figure 4.5 requires the use of SSL. Figure 4.5 also assumes that the client has a valid certificate that can be used to identify the client to the server. Certificate-based authentication is generally considered preferable to password-based authentication because it is based on what the user has (the private key) as well as what the user knows (the password that protects the private key).

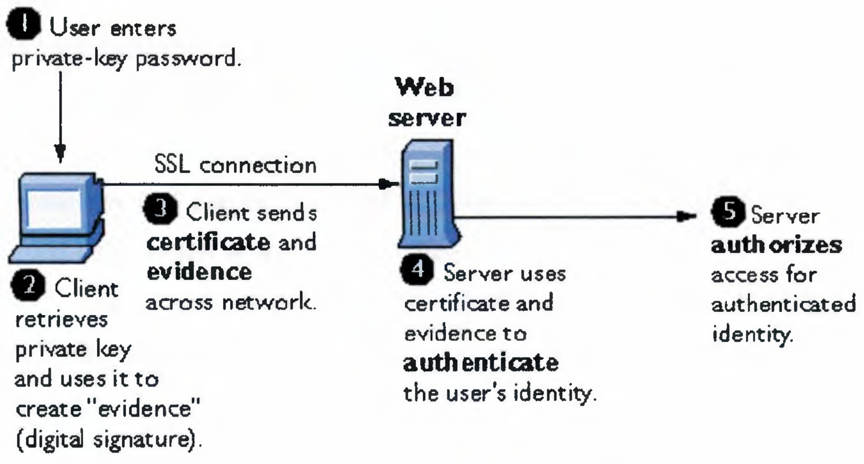

1 User enters private-key password.
Web server
SSL connection
3 Client sends certificate and evidence across network.
2 Client retrieves private key and uses it to create "evidence" (digital signature).
4 Server uses certificate and evidence to authenticate the user's identity.
5 Server authorizes access for authenticated identity.

Figure 4.5 Using a certificate to authenticate a client to a server

However, it's important to note that these two assumptions are true only if unauthorized personnel have not gained access to the user's machine or password, the password for the client software's private key database has been set, and the software is set up to request the password at reasonably frequent intervals.

- Important Neither password-based authentication nor certificate-based authentication address security issues related to physical access to individual machines or passwords. Public- key cryptography can only verify that a private key used to sign some data corresponds to the public key in a certificate. It is the user's responsibility to protect a machine's physical security and to keep the private-key password secret.

These are the steps shown in Figure 4.3:

1. The client software, such as Communicator, maintains a database of the private keys that correspond to the public keys published in any certificates issued for that client. The client asks for the password to this database the first time the client needs to access it during a given session--for example, the first time the user attempts to access an SSL-enabled server that requires certificate-based client authentication. After entering this password once, the user doesn't

need to enter it again for the rest of the session, even when accessing other SSL-enabled servers.

2. The client unlocks the private-key database, retrieves the private key for the user's certificate, and uses that private key to digitally sign some data that has been randomly generated for this purpose on the basis of input from both the client and the server. This data and the digital signature constitute "evidence" of the private key's validity. The digital signature can be created only with that private key and can be validated with the corresponding public key against the signed data, which is unique to the SSL session.

3. The client sends both the user's certificate and the evidence (the randomly generated piece of data that has been digitally signed) across the network.

4. The server uses the certificate and the evidence to authenticate the user's identity. (For a detailed discussion of the way this works, see Introduction to SSL.)

5. At this point the server may optionally perform other authentication tasks, such as checking that the certificate presented by the client is stored in the user's entry in an LDAP directory. The server then continues to evaluate whether the identified user is permitted to access the requested resource. This evaluation process can employ a variety of standard authorization mechanisms, potentially using additional information in an LDAP directory, company databases, and so on. If the result of the evaluation is positive, the server allows the client to access the requested resource.

As you can see by comparing Figure 4.5 to Figure 4.4, certificates replace the authentication portion of the interaction between the client and the server. Instead of requiring a user to send passwords across the network throughout the day, single sign-on requires the user to enter the private-key database password just once, without sending it across the network.

For the rest of the session, the client presents the user's certificate to authenticate the user to each new server it encounters. Existing authorization mechanisms based on the authenticated user identity are not affected.

4.7. How Certificates Are Used

4.7.1. Types of Certificates

Five kinds of certificates are commonly used with Netscape products:

- **Client SSL certificates:** Used to identify clients to servers via SSL (client authentication). Typically, the identity of the client is assumed to be the same as the identity of a human being, such as an employee in an enterprise. See Certificate-Based Authentication for a description of the way client SSL certificates are used for client authentication. Client SSL certificates can also be used for Form Signing and as part of a Single Sign-On solution.

  Examples: A bank gives a customer a client SSL certificate that allows the bank's servers to identify that customer and authorize access to the customer's accounts. A company might give a new employee a client SSL certificate that allows the company's servers to identify that employee and authorize access to the company's servers.

- **Server SSL certificates:** Used to identify servers to clients via SSL (server authentication). Server authentication may be used with or without client authentication. Server authentication is a requirement for an encrypted SSL session. See SSL Protocol.

  Example: Internet sites that engage in electronic commerce (commonly known as **e-commerce**) usually support certificate-based server authentication, at a minimum, to establish an encrypted SSL session and to assure customers that they are dealing with a web site identified with a particular company. The encrypted SSL session ensures that personal information sent over the network, such as credit card numbers, cannot easily be intercepted.

- **S/MIME certificates:** Used for signed and encrypted email. As with client SSL certificates, the identity of the client is typically assumed to be the same as the identity of a human being, such as an employee in an enterprise. A single certificate may be used as both an S/MIME certificate and an SSL certificate. See Signed and Encrypted Email. S/MIME certificates can also be used for Form Signing and as part of a Single Sign-On solution.

Examples: A company deploys combined S/MIME and SSL certificates solely for the purpose of authenticating employee identities, thus permitting signed email and client SSL authentication but not encrypted email. Another company issues S/MIME certificates solely for the purpose of both signing and encrypting email that deals with sensitive financial or legal matters.

- **Object-signing certificates:** Used to identify signers of Java code, JavaScript scripts, or other signed files. See Object Signing.

Example: A software company signs software distributed over the Internet to provide users with some assurance that the software is a legitimate product of that company. Using certificates and digital signatures in this manner can also make it possible for users to identify and control the kind of access downloaded software has to their computers.

- **CA certificates:** Used to identify CAs. Client and server software use CA certificates to determine what other certificates can be trusted. See How CA Certificates Are Used to Establish Trust.

Example: The CA certificates stored in Communicator determine what other certificates that copy of Communicator can authenticate. An administrator can implement some aspects of corporate security policies by controlling the CA certificates stored in each user's copy of Communicator.

4.8. SSL Protocol

The Secure Sockets Layer (SSL) protocol, which was originally developed by Netscape, is a set of rules governing server authentication, client authentication, and encrypted communication between servers and clients. SSL is widely used on the Internet, especially for interactions that involve exchanging confidential information such as credit card numbers.

SSL requires a server SSL certificate, at a minimum. As part of the initial "handshake" process, the server presents its certificate to the client to authenticate the server's identity. The authentication process uses Public-Key Encryption and Digital Signatures to confirm that the server is in fact the server it claims to be. Once the server has been authenticated, the client and server use techniques of Symmetric-Key Encryption, which is very fast, to encrypt all the information they exchange for the remainder of the session and to detect any tampering that may have occurred.

Servers may optionally be configured to require client authentication as well as server authentication. In this case, after server authentication is successfully completed, the client must also present its certificate to the server to authenticate the client's identity before the encrypted SSL session can be established.

4.9. Signed and Encrypted Email

Some email programs (including Messenger, which is part of Communicator) support digitally signed and encrypted email using a widely accepted protocol known as Secure Multipurpose Internet Mail Extension (S/MIME). Using S/MIME to sign or encrypt email messages requires the sender of the message to have an S/MIME certificate.

An email message that includes a digital signature provides some assurance that it was in fact sent by the person whose name appears in the message header, thus providing authentication of the sender. If the digital signature cannot be validated by the email software on the receiving end, the user will be alerted.

The digital signature is unique to the message it accompanies. If the message received differs in any way from the message that was sent--even by the addition or deletion of a comma--the digital signature cannot be validated. Therefore, signed email also provides some assurance that the email has not been tampered with. As discussed at the beginning of this chapter, this kind of assurance is known as nonrepudiation. In other words, signed email makes it very difficult for the sender to deny having sent the message. This is important for many forms of business communication. S/MIME also makes it possible to encrypt email messages. This is also important for some business users. However, using encryption for email requires careful planning. If the recipient of encrypted email messages loses his or her private key and does not have access to a backup copy of the key, for example, the encrypted messages can never be decrypted.

4.9.1. Single Sign-On

Network users are frequently required to remember multiple passwords for the various services they use. For example, a user might have to type a different password to log into the network, collect email, use directory services, use the corporate calendar program, and access various servers. Multiple passwords are an ongoing headache for both users and system administrators. Users have difficulty keeping track of different passwords, tend to choose poor ones, and tend to write them down in obvious places. Administrators must keep track of a separate password database on each server and deal with potential security problems related to the fact that passwords are sent over the network routinely and frequently.

Solving this problem requires some way for a user to log in once, using a single password, and get authenticated access to all network resources that user is authorized to use--without sending any passwords over the network. This capability is known as

4.9.2. single sign-on

Both client SSL certificates and S/MIME certificates can play a significant role in a comprehensive single sign-on solution. For example, one form of single sign-on supported by Netscape products relies on SSL client authentication (see Certificate-Based Authentication). A user can log in once, using a single password to the local client's private-key database, and get authenticated access to all SSL-enabled servers that user is authorized to use--without sending any passwords over the network. This approach simplifies access for users, because they don't need to enter passwords for each new server. It also simplifies network management, since administrators can control access by controlling lists of certificate authorities (CAs) rather than much longer lists of users and passwords.

In addition to using certificates, a complete single-sign on solution must address the need to interoperate with enterprise systems, such as the underlying operating system, that rely on passwords or other forms of authentication.

For information about the single sign-on support currently provided by Netscape products, see Single Sign-On Deployment Guide.

4.9.3. Form Signing

Many kinds of e-commerce require the ability to provide persistent proof that someone has authorized a transaction. Although SSL provides transient client authentication for the duration of an SSL connection, it does not provide persistent authentication for transactions that may occur during that connection. S/MIME provides persistent authentication for email, but e-commerce often involves filling in a form on a web page rather than sending an email.
The Netscape technology known as form signing addresses the need for persistent authentication of financial transactions. Form signing allows a user to associate a digital signature with web-based data generated as the result of a transaction, such as a purchase order or other financial chapter. The private key associated with either a client SSL certificate or an S/MIME certificate may be used for this purpose.
When a user clicks the Submit button on a web-based form that supports form signing, a dialog box appears that displays the exact text to be signed. The form designer can either specify the certificate that should be used or allow the user to select a certificate from among the client SSL and S/MIME certificates that are installed in Communicator. When the user clicks OK, the text is signed, and both the text and the digital signature are submitted to the server. The server can then use a Netscape utility called the Signature Verification Tool to validate the digital signature.

4.9.4. Object Signing

Communicator and other Netscape products support a set of tools and technologies called object signing. Object signing uses standard techniques of public-key cryptography to let users get reliable information about code they download in much the same way they can get reliable information about shrink-wrapped software.
Most importantly, object signing helps users and network administrators implement decisions about software distributed over intranets or the Internet--for example, whether

to allow Java applets signed by a given entity to use specific computer capabilities on specific users' machines.

The "objects" signed with object signing technology can be applets or other Java code, JavaScript scripts, plug-ins, or any kind of file. The "signature" is a digital signature. Signed objects and their signatures are typically stored in a special file called a JAR file. Software developers and others who wish to sign files using object-signing technology must first obtain an object-signing certificate.

4.10. Secure Sockets Layer (SSL)

This chapter introduces the Secure Sockets Layer (SSL) protocol. Originally developed by Netscape, SSL has been universally accepted on the World Wide Web for authenticated and encrypted communication between clients and servers. The new Internet Engineering Task Force (IETF) standard called Transport Layer Security (TLS) is based on SSL. This was recently published as an IETF Internet-Draft, The TLS Protocol Version 1.0. Netscape products will fully support TLS. This chapter is primarily intended for administrators of Netscape server products, but the information it contains may also be useful for developers of applications that support SSL. The chapter assumes that you are familiar with the basic concepts of public-key cryptography, as summarized in the companion chapter Introduction to Public-Key Cryptography.[25]

4.10.1. The SSL Protocol

The Transmission Control Protocol/Internet Protocol (TCP/IP) governs the transport and routing of data over the Internet. Other protocols, such as the HyperText Transport Protocol (HTTP), Lightweight Directory Access Protocol (LDAP), or Internet Messaging Access Protocol (IMAP), run "on top of" TCP/IP in the sense that they all use TCP/IP to support typical application tasks such as displaying web pages or running email servers.

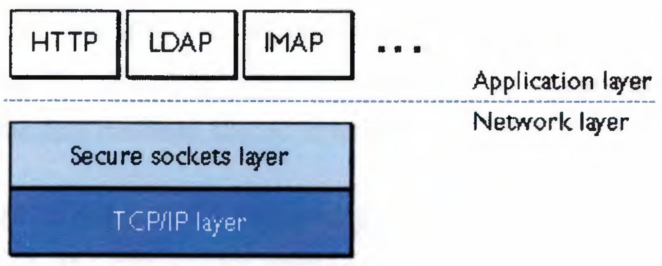


Figure 4.6 SSL runs above TCP/IP and below high-level application protocols

The SSL protocol runs above TCP/IP and below higher-level protocols such as HTTP or IMAP. It uses TCP/IP on behalf of the higher-level protocols, and in the process allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection.

These capabilities address fundamental concerns about communication over the Internet and other TCP/IP networks:

- **SSL server authentication** allows a user to confirm a server's identity. SSL-enabled client software can use standard techniques of public-key cryptography to check that a server's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the client's list of trusted CAs. This confirmation might be important if the user, for example, is sending a credit card number over the network and wants to check the receiving server's identity.
- **SSL client authentication** allows a server to confirm a user's identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check that a client's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the server's list of trusted CAs. This confirmation might be important if the server, for example, is a bank sending confidential financial information to a customer and wants to check the recipient's identity.
- **An encrypted SSL connection** requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thus providing a high degree of confidentiality. Confidentiality is important for both parties to any private transaction. In

addition, all data sent over an encrypted SSL connection is protected with a mechanism for detecting tampering--that is, for automatically determining whether the data has been altered in transit.

The SSL protocol includes two sub-protocols: the SSL record protocol and the SSL handshake protocol. The SSL record protocol defines the format used to transmit data. The SSL handshake protocol involves using the SSL record protocol to exchange a series of messages between an SSL-enabled server and an SSL-enabled client when they first establish an SSL connection. This exchange of messages is designed to facilitate the following actions:

- Authenticate the server to the client.
- Allow the client and server to select the cryptographic algorithms, or ciphers, that they both support.
- Optionally authenticate the client to the server.
- Use public-key encryption techniques to generate shared secrets.
- Establish an encrypted SSL connection.

4.10.2. Ciphers Used with SSL

The SSL protocol supports the use of a variety of different cryptographic algorithms, or ciphers, for use in operations such as authenticating the server and client to each other, transmitting certificates, and establishing session keys. Clients and servers may support different cipher suites, or sets of ciphers, depending on factors such as the version of SSL they support, company policies regarding acceptable encryption strength, and government restrictions on export of SSL-enabled software. Among its other functions, the SSL handshake protocol determines how the server and client negotiate which cipher suites they will use to authenticate each other, to transmit certificates, and to establish session keys.
The cipher suite descriptions that follow refer to these algorithms:

- **DES:** Data Encryption Standard, an encryption algorithm used by the U.S. Government.
- **DSA:** Digital Signature Algorithm, part of the digital authentication standard used by the U.S. Government.

- **KEA:** Key Exchange Algorithm, an algorithm used for key exchange by the U.S. Government.
- **MD5:** Message Digest algorithm developed by Rivest.
- **RC2 and RC4:** Rivest encryption ciphers developed for RSA Data Security.
- **RSA:** A public-key algorithm for both encryption and authentication. Developed by Rivest, Shamir, and Adleman.
- **RSA key exchange:** A key-exchange algorithm for SSL based on the RSA algorithm.
- **SHA-1:** Secure Hash Algorithm, a hash function used by the U.S. Government.
- **SKIPJACK:** A classified symmetric-key algorithm implemented in FORTEZZA-compliant hardware used by the U.S. Government. (For more information, see FORTEZZA Cipher Suites.)
- **Triple-DES:** DES applied three times.

Key-exchange algorithms like KEA and RSA key exchange govern the way in which the server and client determine the symmetric keys they will both use during an SSL session. The most commonly used SSL cipher suites use RSA key exchange.

The SSL 2.0 and SSL 3.0 protocols support overlapping sets of cipher suites. Administrators can enable or disable any of the supported cipher suites for both clients and servers. When a particular client and server exchange information during the SSL handshake, they identify the strongest enabled cipher suites they have in common and use those for the SSL session.

Decisions about which cipher suites a particular organization decides to enable depend on trade-offs among the sensitivity of the data involved, the speed of the cipher, and the applicability of export rules.

Some organizations may want to disable the weaker ciphers to prevent SSL connections with weaker encryption. However, due to U.S. government restrictions on products that support anything stronger than 40-bit encryption, disabling support for all 40-bit ciphers effectively restricts access to network browsers that are available only in the United States (unless the server involved has a special Global Server ID that permits the international client to "step up" to stronger encryption). For more information about U.S. export restrictions, see Export Restrictions on International Sales.

To serve the largest possible range of users, it's administrators may wish to enable as broad a range of SSL cipher suites as possible. That way, when a domestic client or server is dealing with another domestic server or client, respectively, it will negotiate the use of the strongest ciphers available. And when an domestic client or server is dealing with an international server or client, it will negotiate the use of those ciphers that are permitted under U.S. export regulations.

However, since 40-bit ciphers can be broken relatively quickly, administrators who are concerned about eavesdropping and whose user communities can legally use stronger ciphers should disable the 40-bit ciphers.

4.11. Server Authentication

Netscape's SSL-enabled client software always requires server authentication, or cryptographic validation by a client of the server's identity. As explained in Step 2 of The SSL Handshake, the server sends the client a certificate to authenticate itself. The client uses the certificate in Step 3 to authenticate the identity the certificate claims to represent. To authenticate the binding between a public key and the server identified by the certificate that contains the public key, an SSL-enabled client must receive a "yes" answer to the four questions shown in Figure 4.7, Although the fourth question is not technically part of the SSL protocol, it is the client's responsibility to support this requirement, which provides some assurance of the server's identity and thus helps protect against a form of security attack known as "man in the middle." [27]

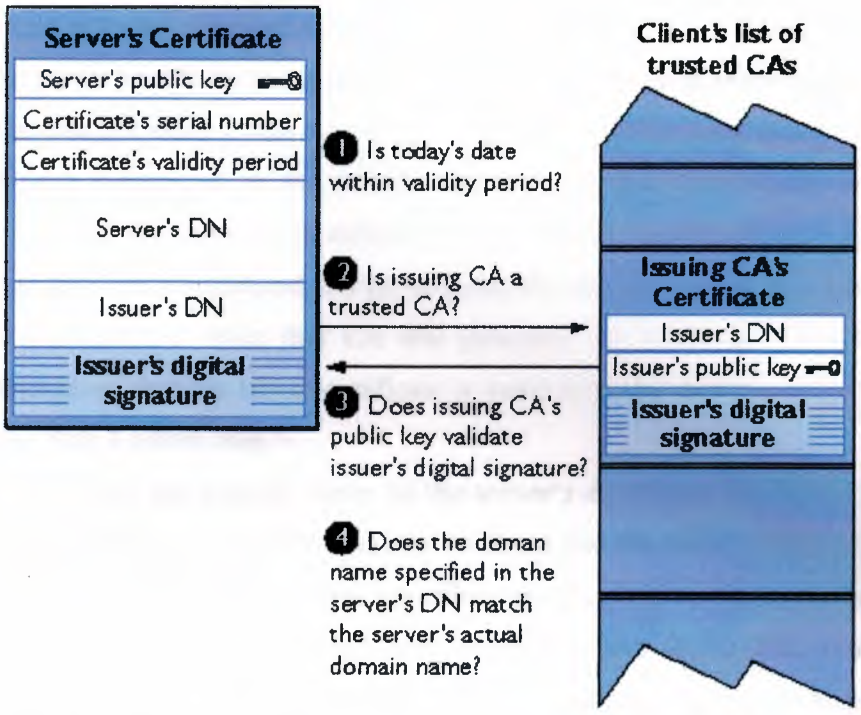

Server's Certificate
Server's public key
Certificate's serial number
Certificate's validity period
Server's DN
Issuer's DN
Issuer's digital signature
Client's list of trusted CAs
1 Is today's date within validity period?
2 Is issuing CA a trusted CA?
3 Does issuing CA's public key validate issuer's digital signature?
4 Does the doman name specified in the server's DN match the server's actual domain name?
Issuing CA's Certificate
Issuer's DN
Issuer's public key
Issuer's digital signature

Figure 4.7 How a Netscape server authenticates a client certificate

An SSL-enabled client goes through these steps to authenticate a server's identity:

1. **Is today's date within the validity period?** The client checks the server certificate's validity period. If the current date and time are outside of that range, the authentication process won't go any further. If the current date and time are within the certificate's validity period, the client goes on to Step 2.
2. **Is the issuing CA a trusted CA?** Each SSL-enabled client maintains a list of trusted CA certificates, represented by the shaded area on the right side of Figure 4.12. This list determines which server certificates the client will accept. If the distinguished name (DN) of the issuing CA matches the DN of a CA on the client's list of trusted CAs, the answer to this question is yes, and the client goes on to Step 3. If the issuing CA is not on the list, the server will not be authenticated unless the client can verify a certificate chain ending in a CA that is on the list .
3. **Does the issuing CA's public key validate the issuer's digital signature?** The client uses the public key from the CA's certificate (which it

found in its list of trusted CAs in step 2) to validate the CA's digital signature on the server certificate being presented. If the information in the server certificate has changed since it was signed by the CA or if the CA certificate's public key doesn't correspond to the private key used by the CA to sign the server certificate, the client won't authenticate the server's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the client has determined that the server certificate is valid. It is the client's responsibility to take Step 4 before Step 5.

4. **Does the domain name in the server's certificate match the domain name of the server itself?** This step confirms that the server is actually located at the same network address specified by the domain name in the server certificate. Although step 4 is not technically part of the SSL protocol, it provides the only protection against a form of security attack known as a Man-in-the-Middle Attack. Clients must perform this step and must refuse to authenticate the server or establish a connection if the domain names don't match. If the server's actual domain name matches the domain name in the server certificate, the client goes on to Step 5.

5. **The server is authenticated.** The client proceeds with the SSL handshake. If the client doesn't get to step 5 for any reason, the server identified by the certificate cannot be authenticated, and the user will be warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server requires client authentication, the server performs the steps described in Client Authentication.

After the steps described here, the server must successfully use its private key to decrypt the premaster secret the client sends in Step 4 of The SSL Handshake. Otherwise, the SSL session will be terminated. This provides additional assurance that the identity associated with the public key in the server's certificate is in fact the server with which the client is connected.

4.11.1. Man-in-the-Middle Attack

As suggested in Step 4 above, the client application must check the server domain name specified in the server certificate against the actual domain name of the server with which the client is attempting to communicate. This step is necessary to protect against a man-in the middle attack, which works as follows.
The "man in the middle" is a rogue program that intercepts all communication between the client and a server with which the client is attempting to communicate via SSL. The rogue program intercepts the legitimate keys that are passed back and forth during the SSL handshake, substitutes its own, and makes it appear to the client that it is the server, and to the server that it is the client.
The encrypted information exchanged at the beginning of the SSL handshake is actually encrypted with the rogue program's public key or private key, rather than the client's or server's real keys. The rogue program ends up establishing one set of session keys for use with the real server, and a different set of session keys for use with the client. This allows the rogue program not only to read all the data that flows between the client and the real server, but also to change the data without being detected. Therefore, it is extremely important for the client to check that the domain name in the server certificate corresponds to the domain name of the server with which a client is attempting to communicate--in addition to checking the validity of the certificate by performing the other steps described in Server Authentication.

4.12. Client Authentication

SSL-enabled servers can be configured to require client authentication, or cryptographic validation by the server of the client's identity. When a server configured this way requests client authentication (see Step 6 of The SSL Handshake), the client sends the server both a certificate and a separate piece of digitally signed data to authenticate itself. The server uses the digitally signed data to validate the public key in the certificate and to authenticate the identity the certificate claims to represent.
The SSL protocol requires the client to create a digital signature by creating a one-way hash from data generated randomly during the handshake and known only to the client and server. The hash of the data is then encrypted with the private key that corresponds to the public key in the certificate being presented to the server.

To authenticate the binding between the public key and the person or other entity identified by the certificate that contains the public key, an SSL-enabled server must receive a "yes" answer to the first four questions shown in Figure 4.8. Although the fifth question is not part of the SSL protocol, Netscape servers can be configured to support this requirement to take advantage of the user's entry in an LDAP directory as part of the authentication process. [28]

An SSL-enabled server goes through these steps to authenticate a user's identity:

1. **Does the user's public key validate the user's digital signature?** The server checks that the user's digital signature can be validated with the public key in the certificate. If so, the server has established that the public key asserted to belong to John Doe matches the private key used to create the signature and that the data has not been tampered with since it was signed. At this point, however, the binding between the public key and the DN specified in the certificate has not yet been established. The certificate might have been created by someone attempting to impersonate the user. To validate the binding between the public key and the DN, the server must also complete Step 3 and Step 4.

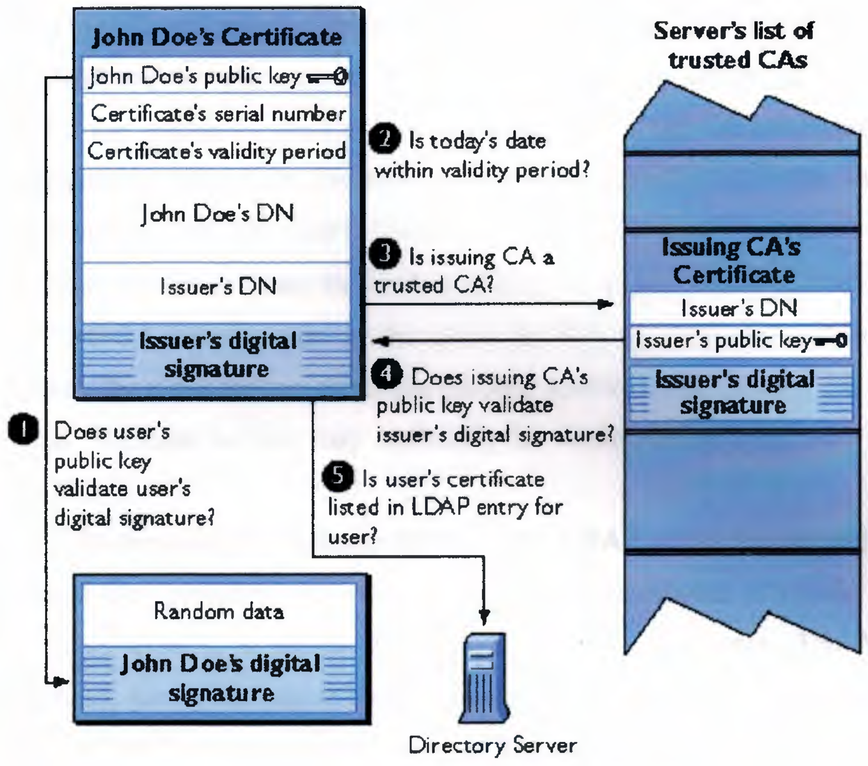

John Doe's Certificate
John Doe's public key
Certificate's serial number
Certificate's validity period
John Doe's DN
Issuer's DN
Issuer's digital signature
Server's list of trusted CAs
2 Is today's date within validity period?
3 Is issuing CA a trusted CA?
Issuing CA's Certificate
Issuer's DN
Issuer's public key
Issuer's digital signature
4 Does issuing CA's public key validate issuer's digital signature?
1 Does user's public key validate user's digital signature?
5 Is user's certificate listed in LDAP entry for user?
Random data
John Doe's digital signature
Directory Server

Figure 4.8 How a Netscape server authenticates a client certificate

2. **Is today's date within the validity period?** The server checks the certificate's validity period. If the current date and time are outside of that range, the authentication process won't go any further. If the current date and time are within the certificate's validity period, the server goes on to Step 3.

3. **Is the issuing CA a trusted CA?** Each SSL-enabled server maintains a list of trusted CA certificates, represented by the shaded area on the right side of Figure 4.8 This list determines which certificates the server will accept. If the DN of the issuing CA matches the DN of a CA on the server's list of trusted CAs, the answer to this question is yes, and the server goes on to Step 4. If the issuing CA is not on the list, the client will not be authenticated unless the server can verify a certificate chain ending in a CA that is on the list (see CA Hierarchies for details). Administrators can control which certificates are trusted or not trusted within their organizations by controlling the lists of CA certificates maintained by clients and servers.

4. **Does the issuing CA's public key validate the issuer's digital signature?** The server uses the public key from the CA's certificate (which it

found in its list of trusted CAs in Step 3) to validate the CA's digital signature on the certificate being presented. If the information in the certificate has changed since it was signed by the CA or if the public key in the CA certificate doesn't correspond to the private key used by the CA to sign the certificate, the server won't authenticate the user's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the SSL protocol allows the server to consider the client authenticated and proceed with the connection as described in Step 6. Netscape servers may optionally be configured to take Step 5 before Step 6.

5. **Is the user's certificate listed in the LDAP entry for the user?** This optional step provides one way for a system administrator to revoke a user's certificate even if it passes the tests in all the other steps. The Netscape Certificate Server can automatically remove a revoked certificate from the user's entry in the LDAP directory. All servers that are set up to perform this step will then refuse to authenticate that certificate or establish a connection. If the user's certificate in the directory is identical to the user's certificate presented in the SSL handshake, the server goes on to step 6.

6. **Is the authenticated client authorized to access the requested resources?** The server checks what resources the client is permitted to access according to the server's access control lists (ACLs) and establishes a connection with appropriate access. If the server doesn't get to step 6 for any reason, the user identified by the certificate cannot be authenticated, and the user is not allowed to access any server resources that require authentication.

4.13. Summary

This chapter introduces the basic concepts of public-key cryptography. To transfer information privately and securely across the Internet, the Secure Socket Layer (SSL) protocol was developed. This chapter examines the SSL protocol and details how to improve the performance and scalability of Web sites by off-loading SSL processing and object delivery from Web servers.

5. CACHING WEB PROXY SERVER USING JAVA APPLICATION

5.1 Overview

The designed and implemented caching web proxy server using java, which can be configured remotely over the Internet by its administrator. The proxy handles only HTTP communications. The source is written in Java, which makes the code cross-compiler and the application cross-platform. The Java technology enables the main proxy application to send Java applets to remote administrator, sitting on a different machine on a remote network, and accessed over the Internet by standard TCP\IP communication.

The proxy includes two applications. The main application starts up and serves and a proxy server which listens to clients requests on a specific port, and forwards the requests to a web server or to another web proxy (father proxy), then sending the replys back to the clients. This application will be referred to as the *proxy* application.

The proxy application also does caching. When a client requests an object, the proxy checks if the object is cached. If so, it does not forward the request, and replies to the client with the cached object (this is called a *cache hit*). If the requested object is not cached (*cache miss*), the request is forwarded, and when the reply arrives to the proxy, it both sends it back to the client and caches it on its machine, for future use. Thus, if a client requests an object which has already been requested (by the client or by another client), the proxy retrieves the object from its local machine cache and sends it back, without searching for the object out there on the Internet. This feature gives a performance boost to the proxy itself and to the client (and to all other clients who will request these cached objects in the future). Caching the objects on the local machine is done using the host machine file system, thus the cache size is dependent on the local system resources in terms of hard disk free space. [29]

The feature of forwarding requests to another web proxy **or** to a web server, along with the caching behavior, enables one to use a hierarchy of caching proxy servers. Modern client browsers do caching on the client local machine, so caching is really done in levels (hierarchy): The first level is in the RAM on the client machine (usually the browser caches a few web pages in memory). The size of this level is depended on the memory resources of the browser machine. The second level, also done by the browser, is keeping the objects in a persistent storage, using the local file system on the browser machine. The third level is the caching done by the proxy, on its machine. Then, if the proxy can forward requests to another proxy, thus creating a chain or a hierarchy of proxies, each one of them does its caching on its machine, and by this creating a new caching level. If the requested object could not be found throughout the cache chain, the request is finally forwarded to the web server, which delivers the bits back through the chain of proxies, enables each of them to cache the reply for future use.

5.2. Remote administration

The second application is a Java applet, referred to as *the applet.* It enables the proxy administrator to remotely manage the proxy, from his machine, via its web browser. When the administrator wants to, he can access the desired proxy server by typing the proxy machine's IP address (or machine name) on the URL address window in the browser plus the suffix '/admin' (for instance, 'techst02/admin'). The browser considers this to be a valid client request, and forwards it on to the proxy (or chain of proxies). When the proxy monitors this request, it compares it at run-time to the IP address (or machine name) of the host machine on which it runs. If they do not match, the proxy forwards this as a normal request on to the father proxy (or web server). But, if they do match, then the proxy assumes that an administrator is trying to attach. It responses by sending back a Java applet that handles all the remote configuration and the necessary security issues (such as password login), and of course does not forward that request. [30]

Back on the administrator machine, the browser gets a Java applet and starts it. The applet starts by requesting a password from the administrator and sending it to the proxy application. The applet and the proxy now talk full duplex. If the password is correct, the proxy sends Ack to the applet, and the applet responses by presenting all the parameters and status of the proxy, enables the administrator to alter parameters, thus change the proxy behavior (in terms of traffic management and cache activities), and sending the new parameters to the proxy. All of the operations in the administrator machine take place through its browser, by the applet, thus having full Graphical User

Interface (GUI) support which is not restricted only to HTML web forms, seen on search engines for example. It is the power of Java applets technology that gives the administrator the ability to control the proxy remotely, plus the friendly graphical environment to do so - thus, if we or anyone else in the future decide to enhance the set of configurable parameters or the user interface to the administrator, the infrastructure is there to be used and enhanced. We're talking applets here, not just a dull HTML page form.

A note on security: The applet only present GUI to the administrator and handles communications to the proxy application. When the administrator enters the login password, the applet sends it over the Internet to the proxy. The proxy checks the validity of the password and sends back to the applet Ack/Nack, based on that the applet logs the administrator in or not. So attackers can learn nothing at the password from viewing the applet operations.

5.3. Accessing multiple proxies remotely

We talked about a structure of proxy hierarchy, and how the proxy and the cache behavior support that. The remote access feature also support this design.

The administrator can access a unique proxy by specifying its IP address or machine name on the browser URL window. The requested proxy will be the only one to respond to the administrator by sending the applet. The applet and the requested proxy will talk full duplex over the Internet, and over all other chained proxies in the hierarchy. When the administrator alters the proxy parameters, only the requested proxy will be affected - all the other proxies will not change. This is important because it enables one administrator fully control the behavior of each and every one of the proxies in the structure. It also enables a group of administrators to control *their* proxies (the US administrator will control the proxies in the US, and independently the France administrator will control the proxies in France. Now, is that hot or what??).

Still, there's a potential design problem here. Let's say two or more administrators want to control a certain proxy. They both access the proxy at the same time, and each of them does not know about the other. Now, if one of them wants the proxy to behave in a certain manner, and the other administrator wants the opposite, it will lead - in the best case of implementation - to inconsistent result from one of the administrator point of view ("what the hell is going on here, I instructed the proxy to do something and it does the opposite!"). This is a design problem, and we solved it by deciding that remotely controlling a certain proxy is limited to one human administrator at a time. When an administrator controls the proxy, no other administrator can control the same proxy (this is very similar to protecting shared resources from multiple threads in a multithreaded environment). So, this solution ensures that a scenario like that could not happen. [31]

5.4. Multithreading

All the clients' requests plus the administrator remote configuration are done parallelly by using threads. A certain proxy can handle multiple client requests at the same time, plus be remotely controlled by an administrator. For instance, in a certain time frame, the proxy can handle a request from client A, a request from client B, two requests from client C (this could happen because the modern browsers use threads too...) and communications with a remote applet running on the administrator machine. Non of these users (clients and administrator) would notice the difference.

Using threads leads to special problems, concerning shared resources and synchronization issues.

5.5. Caching

The cache manager is a static object. It encapsulates all the details of caching from the other components in the application. For example, the proxy thread does not aware of the file name in which the requested object is cached - it just delivers a URL argument to the cache manager, which in its turn generate a file name out of that URL. Thus, if we will want in the future to change the mechanism of file name generation, the only code we should change is the code of the cache manager - the proxy thread's code would not be affected. This kind of encapsulation is supported by object oriented development environments, such as Java.

Caching objects is done using the host machine (onwhich the proxy runs) file system. When the cache manager is called to cache an object, it first generates a file name out of the URL of the object. The stream of bits coming from the father proxy or web server is written to a file. In addition, the cache manager holds a hash table data structure. Each entry in the hash table has two fields: a key and a value. The key is the file name, and the value is a date structure. When a new object is cached, it is stored on disk and then in the hash table a new entry created, into which the cache manager inserts the file name and the date it was created (year, month, day, hour, minute, etc.). Later, when the cache manager is asked to check if an object is cached, it first generates a file name out of the URL of the object, then enumerates the hash table to find the file name. If the file name is found, the cache manager returns it to the proxy thread; otherwise, it returns a status indicating 'not cached.[30]

If a requested object is found in the cache (a cache hit), the entry in the hash table is updated with the current date. This enables the Least Recently Used (LRU) algorithm to take place when the cache if full and a file needs to be deleted. So if the free space of the cache is going under a minimum level, the LRU algorithm enumerates the hash table and deletes the LRU file. If we would like in the future to change this policy (maybe to Least Frequently Used, for instance) we should change the value field of the hash table to be some other class, and change the code in the methods incharging of making more free space - all other components are unaware of this mechanism.

The cache manager methods are called from the multiple proxy threads. This could raise problems of synchronization and shared resources protection. We solve these problems by putting multithreading synchronization locks on all shared resources, as supported by the *synchronized()* Java native method. Methods involving the hash table are multithreaded-safe because the hash table object synchronizes all actions performed on it internally (supported by Java hash table methods). Files are multithreaded protected by the Java *File* object and by our code. For example, there is a chance that one thread will read from a file, and a second thread will try to delete this file (a scenario like that could happen, if the first thread is reading a cached object A, and the second thread is caching object B, thus making the cache size grow, and causing the cache manager to make more free space; the algorithm will detect that object A is the LRU, and try to delete it). This will cause the proxy to break in the worst case, or to failure in either the read or the delete operation, in the best case. We should not allow it, so we added code to check if write and delete operations are allowed before doing them.

5.6. Cacheable Vs. Non-Cacheable objects

Not all the replys are to be cached. For example, when a client sends a request to a search engine, typically the search engine treats that request as a query, and generates an HTML web page with the results of the query. This result page should not be cached on the proxy, because next time an identical query will be sent to the engine, different results are likely to be returned (because of the frequent changes in the engine database). These generated pages are called *dynamic* pages, while regular web pages that sit somewhere on a web server waiting to be retrieved are called *static* pages.

The problem is to identify that a certain web page is a dynamic one. We do not know a 100 percent solution to this problem, so we can only follow conventions. For example, dynamic pages can be generated by CGI scripts, and there's no way for the proxy to know that the page is a result of a CGI script, unless it gets some help from the HTTP reply headers or the URL of the page. It is a convention that CGI generated pages are taken from a URL which contains the sub-string "cgi-bin", so we added code to check the URL of each request, and if it contains this sub-string we do not cache the reply. We also check for URLs containing special characters, indicating that this URL is a query. For example, the question mark "?" is a typical character used to submit queries. The proxy also gets help from the reply return code. If it contains a return code other then OK, for example, we do not cache the reply.

Let us just note that this problem is not as serious to the browser cache mechanism as it is to the proxy cache manager. Modern browsers manage caching on the browser cache, and theoretically they could be faced with the same problem - what should not be cached. But even if the browser do cache a non cacheable object, the user can always instruct it to "refresh" or "reload" the object from the Internet - in that case, the browser will re-send the request and wait for reply. On the other hand, proxies behave very differently, and the main reason for that is that the user should not be aware of them. So, if a proxy caches a non cacheable object, and the client will ask to refresh (reload) on his machine, the browser will re-send the request to the proxy, and the proxy would treat that request as a cache hit (because it has the requested object in its cache), will not forward the request, and reply with the cached bits. So it is extremely important for proxies to try to identify which object should not be cached.

5.7. The admin thread

One of the initialization operations done at startup by the web daemon is constructing the admin thread. This thread handles communication with a remote administrator, and sets/gets parameters from other components (such as the cache manager). It first creates a socket (*admin socket*) and listens on a special (admin) port.

When the administrator accesses the proxy, the proxy thread catch the event and sends him back a web page with the admin applet. The applet starts by presenting a login dialog box, and waiting for the administrator to enter password; then it sends the password to the proxy application, to the admin port, thus the admin thread at the main application could catch that request without interfering the handling of all client requests. The admin thread in the main application processes the password and sends back to the applet, on the admin port, an answer (Ack/Nack). Again, having the applet and the main application "talking" full duplex does not reflect on the activities going on in the main application (handling client requests).

At this phase, if another administrator accesses the main application, the proxy thread will catch that and send an admin applet to him too. The applet will run on his machine, and will try to talk to the main application. However, the admin thread in the main application talks only with the first administrator and ignores the second. As soon as the first administrator finishes, the admin thread will treat the second one. This protects the main application from multiple administration accesses, preventing the scenario of two or more remote administrators trying to control the same proxy, thus causing to inconsistent behavior of the proxy or unfriendly behavior in the applets on their machines (see above for a full description of the potential problem). The second administrator do get the applet (otherwise he could think the proxy is unreachable), but sending admin requests to the proxy is blocked until the first administrator finishes. We thought this is the best design approach for the problem, and implemented the solution that way.

After the password check, the administrator gets a full GUI window with the status of the proxy and parameters that can be altered. This is not just an HTML based form, but a powerful Java applet with all the GUI that we wanted (or any developer will want in the future) - including dialogs, check boxes, etc., and getting those user interface controls out of the browser area in a windowing manner. [32]

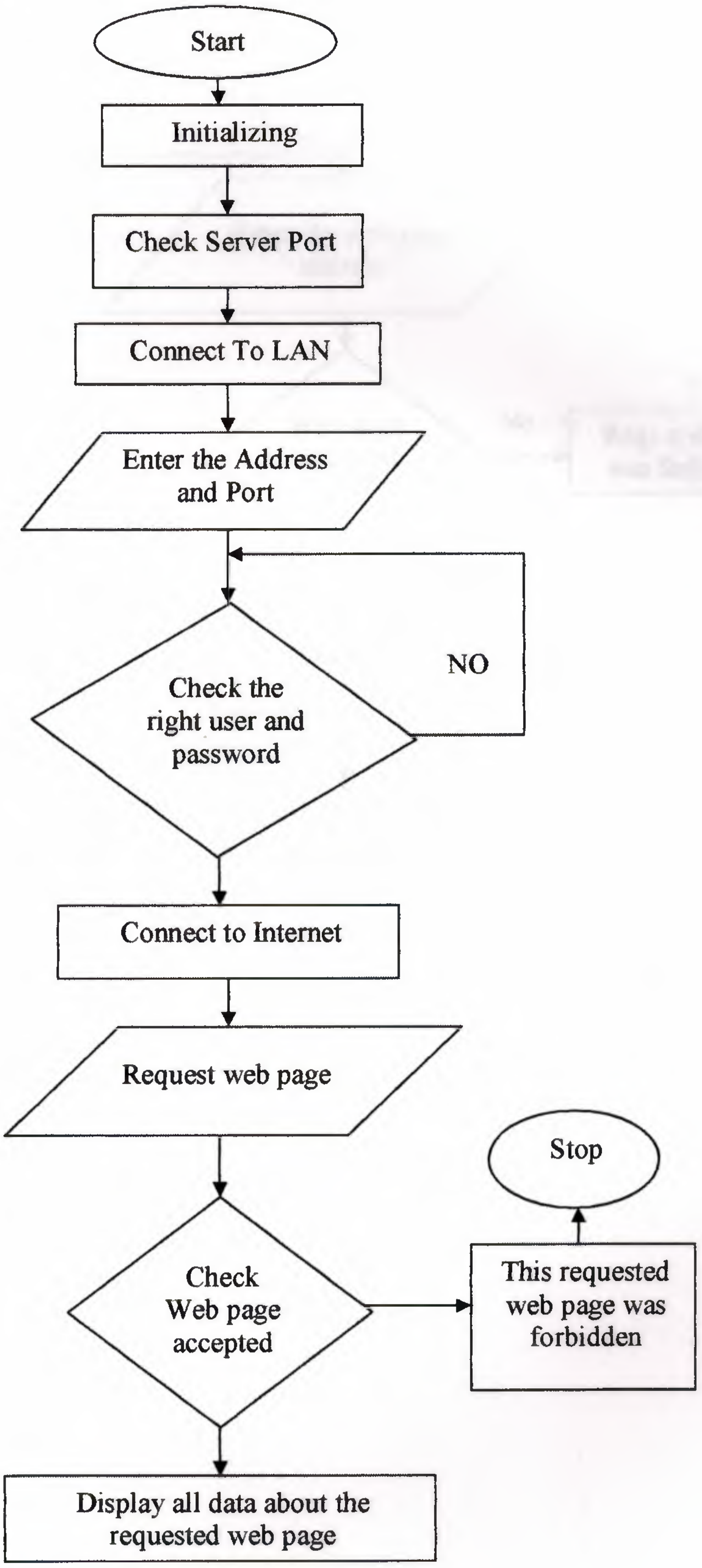
5.8. The config class

When the administrator changes parameters and chooses to send them to the main application, the applet sends the bits to the proxy, and the proxy gets them and alter its behavior. To help both the proxy and the applet with the job of getting and setting parameters, we designed a special class called *config*. This class appears both in the applet code and in the proxy code, and handles all get/set methods involved with the configurable parameters.

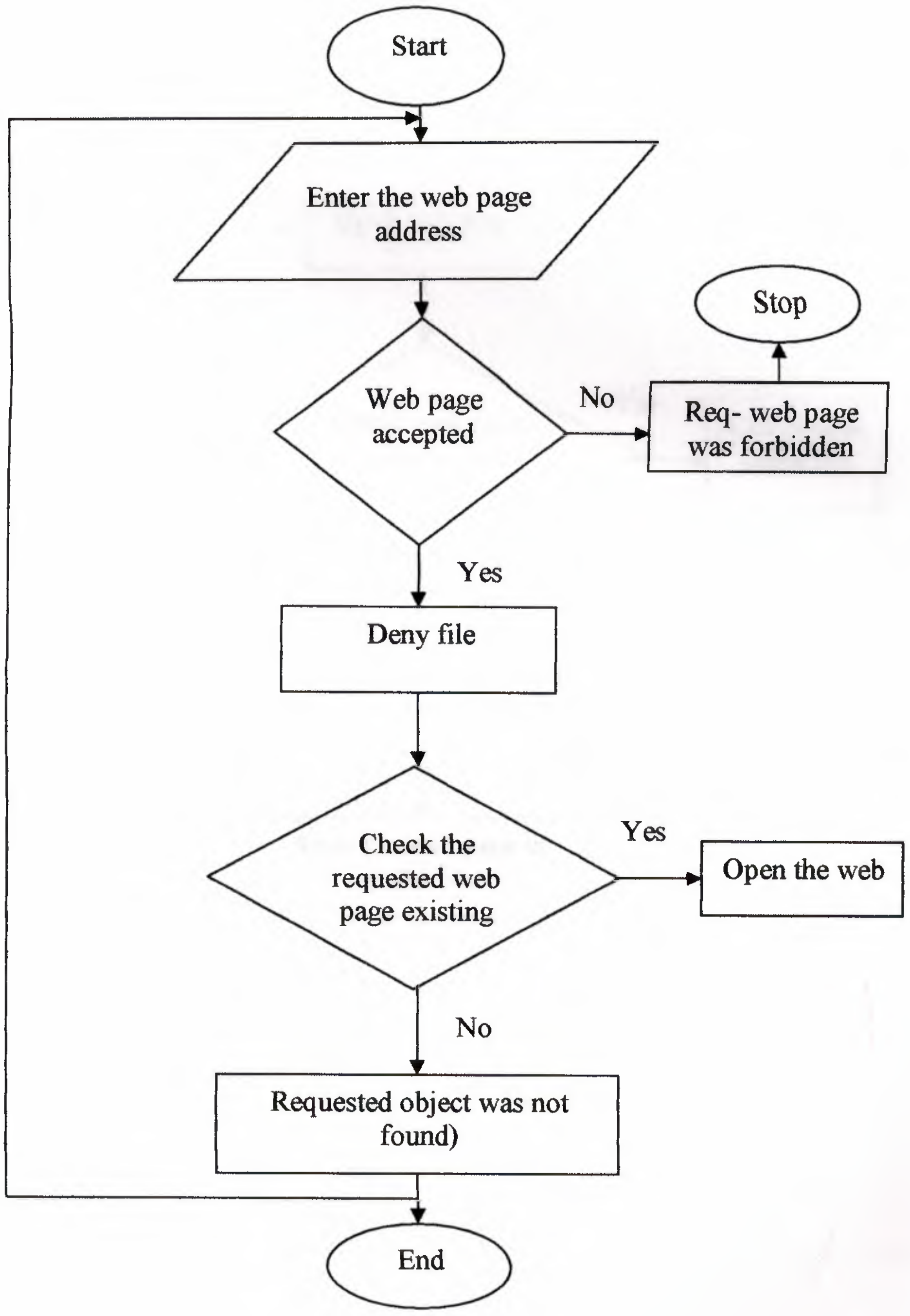
The config class calls methods on other object to retrieve status of their parameters (these are *get* methods), and calls different set of methods on the other object to change their parameters (these are *set* methods). It also does the job of packaging these parameters and sending them over sockets to a remote machine. So, the applet uses the *get* ability of the config class to learn about the status of the main application, and present this status to the administrator, providing him the interface required to change this status, while as the main application uses the *set* ability of the config class to change its parameters. Both the applet and the main application use the config class ability to send the parameters to each other.

We thought this is a good design, for a couple of reasons: First, we should write all the code incharge of managing the configurable parameters only once, and let both the proxy and the applet use it. Second, it support encapsulation and object oriented philosophy - all the details are in the config class and not spread out all over the code, so future changes can be made more easily. For example, if we decide to encrypt the parameters before sending and decrypt them on retrieving, thus making the communication between the applet and the proxy more secure, we should add code to do that only in the config class; all other objects would not be aware to the change.

5.9. The software main block diagram

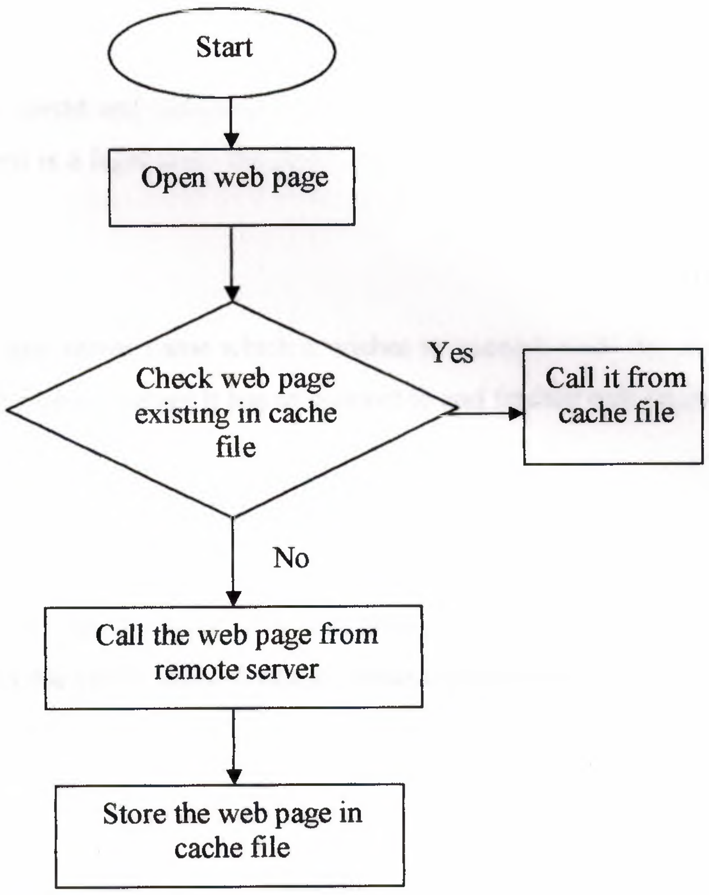

Start
Initializing
Check Server Port
Connect To LAN
Enter the Address and Port
Check the right user and password
NO
Connect to Internet
Request web page
Stop
Check Web page accepted
This requested web page was forbidden
Display all data about the requested web page

Deny Flowchart



Start
Enter the web page address
Web page accepted
No
Req- web page was forbidden
Stop
Yes
Deny file
Check the requested web page existing
Yes
Open the web
No
Requested object was not found)
End

Cache Flowchart


Start
Open web page
Check web page existing in cache file
Yes
Call it from cache file
No
Call the web page from remote server
Store the web page in cache file

5.10. Program Explanation

In this thesis, I implement the following functions:

1. Basic Authentication

Ask the client to enter its userid and username first, then the proxy sever identify if it is the legal user. If the client is a legal user, the proxy allows it connect to other server through the proxy

2. Load Balance

If the client dosn't specify any server name which it wishes to connect with, the proxy randomly chooses one of the server names it has to connect to and fetches web pages to send them to the client.

3. Fault Tolenrance

If the client give a wrong server name that the proxy can't reach, the proxy automatically choose one of the server name it has to connect and fetches web pages to send them to the client

4. Deny some websites

If the client wish to connect with a web server whose name is in the name lists which the proxy established to forbid connectting with, the proxy sends not-found reply to the client.

5. Caching

a. The proxy receives the request of the client and connects the server to fetch web pages, then sends them to the client , meantime caches some of these web pages according to their reply code.

b. The proxy receives the request of the client and finds it has cached the web pages which the client wishes to get, then the proxy read these web page files and send them to the client.

c. The cache space isn't unlimited. If there isn't enough space for the proxy to save more web pages, it deletes some old files according to the date of these files. It's better to delete these files according to the expire date which can be extract from the reply header. But I didn't implement it

6. Calculation

a. Calculate RTT from the proxy to the web server and from the proxy to the client.

b. Calculate Effective Bandwidth of the links from the proxy to the web server and from the proxy to the client.

7. Statistics

a. The total number of requests that the proxy receives from the client

b. The percentage of cached files which has been hit, which means that how many times the cached files has been sent to the client among the total requests.

c. The percentage of caching files, which means that how many files has been cached among the total requests.

8. This program supports multiple clients to send their requests.

5.11. Using the program

1. Main()

Initialize the proxy

1.1 Set the local host and local port

1.2 Set the userid and password for the proxy.

1.3 Creat directory to cache file if there isn't such a directory

1.4 Establish a hashtable of the cached files if there has been a cache directory and some cached files.

1.5 Creat a sokect to listen to the request from clients. Whenever it accepts a connection request, it starts a thread to handle it.

2. Proxy.java

Handle the requests comming from the client

2.1 Call HttpRequest class to extract related information

2.2 If a basic authentication is necessary ?

2.3 If the request has a appropriate web server name ?

2.3.1 If it doesn't specify any server, the proxy randomly choose aserver from the list of the server names which is in the proxy to connect. This is stored in a file "hostname" and it can be modified by the administrator of the proxy.

2.3.2 If it has a server name but the server is forbidden, send a "not found" reply to client. The proxy has a denyID file to record the forbbidden server name and the file can be modified by the administrator of the proxy.

2.2.3 If it has a server name which is not the case metioned above, connect to it.

2.3.4 If it has a server name but it isn't connectable, randomly choose a server from the server name list which is in the proxy.

2.4 If the web page has been cached?

2.4.1 If cached, read the cache file and send them back to the client.

2.4.2 If the web page has not been cached yet, fetch it and send it back to the client while caching it.

2.6 Record the time when proxy send request to web server, when the proxy gets response from web server, when proxy send the response back to the client and when the proxy receive next request.

2.7 Display the results of calculation of RTT and Effective Bandwidth and disply the statistics

3. HttpRequest.java

Process data received from the client

3.1 parseReq()

Receive data from the client and classified them according to HTTP protocol

3.2 rebuildHeader()

Rebuild a request needed to be sent to the web server as if the proxy is a client to the web server.

4. HttpReply.java

Build reply data

4.1 Form reply data to challenge the client for basic authentication

4.2 Form Not Found reply

4.3 General forming reply function to prepare for adding more replys

5. Cache.java

Process data received from the web server

5.1 parseReply()

Receive data from the web server and classified them according to HTTP protocol (but I didn't use it in my program. I write this function because it will be useful if I wish to use lots of fields in the reply header)

5.2 rebuildReply()

Use the information received from web server to build a reply sent to client as if the proxy is the web server which the client wish to connect with. (I don't use it in my program. I prepared it for my sniffing ads and change the address of the ads, but finally I didn't do it)

5.3 IsCached()

See if this request that comes from the client has been cached

5.4 Caching()

Cache web page according the reply code , meantime add this file name into the hashtable

5.5 CacheAll()

Cache all web pages received from th web server. I implement this function mainly for me to analyse the reply data.

5.6 CacheDelete()

Delete the oldest cached file until the proxy has space to cache newly received web pages

6. Config.java

Set and get some data related with the proxy

6.1 Set authentication data (userid and username)

6.2 Get and set wether the proxy has authenticated the client(this function has not been completely implemented because I'm still not very clear about authentication mechanism. If each client needs to be authenticated? When the authenticated state is unvalid?)

6.3 Set and get proxy host name and port number

6.4 getRandHost()

When the client does not indicate the server it wishes to connect,this function randomly choose one server from the list the proxy has prepared

6.5 denyWebsite()

See if the server that the client wish to connect with is in the list the proxy denies to connect.

6.6 isCorrect()

See if the client has sent the right userid and username.

7. Statistic.java

Set and get some data related with statistics and calculations

7.1 Set and get total number of requests that the proxy has received

7.2 Set and get times that the cached files has been hit

7.3 Set and get the number of files that ptoxy has cached

7.4 Calculate percentage of hitting cached files

7.5 Calculate percentage of caching files

7.6 Calculate RTT and Effective Bandwidth

8. Base64.java

An algorithm for basic authentication.(It comes from "http://www.gjt.org/ servlets/JCVSlet/show/ice/com/ice/convert/Base64.java/HEAD") [33]

5.12. Summary

this chapter describes how to use the program and web proxy server using java, which can be configured remotely over the Internet by its administrator. The proxy handles only HTTP communications. The source is written in Java, which makes the code cross-compiler and the application cross-platform. The Java technology enables the main proxy application to send Java applets to remote administrator, sitting on a different machine on a remote network, and accessed over the Internet by standard TCP\IP communication. Before writing code for the thesis, I should do research carefully first. Since I didn't understand this thesis well at the beginning, the structure of the code is not good for me to add some new features. So, it's very important to build an efficient and robust structure for my program.

CONCLUSION

The growth and increased popularity of the World Wide Web has created a corresponding growth in network traffic. With this growth have come delays, slower response times, and security concerns. The network traffic problems are partly due to the repeated retrieving of objects from remote Web servers on the Internet. Proxy Services can help improve performance by locally caching frequently requested Internet information. In general, Proxy Services stores copies of frequently requested Web information closer to the user, thereby reducing the number of times the same information is accessed over an Internet connection, the download time, and the load on the remote server.
The fact that proxies can provide efficient caching makes them useful even when no firewall machine is in order. Configuring a group to use a caching proxy server is easy.

Caching web proxy servers can decrease network traffic costs significantly, because once the first request was made for a certain document, the next ones are retrieved from a local cache.
The proxy handles only HTTP communications. The source is written in Java, which makes the code cross-compiler and the application cross-platform. The Java technology enables the main proxy application to send Java applets to remote administrator, sitting on a different machine on a remote network, and accessed over the Internet by standard TCP\IP communication.

The thesis is includs introduction,five chapters, conculsion and tow appendixes followed by a java application program which allow the server and client to get a complete web page.

In chapter 1 the basic principles of computer network security and solution of security problems in internet is introduced.

In chapter 2 the description of internet firewalls for security is given. The use of firewall teqnique allow to restrict people to enter at controlled points, it is also restricts people to leaving at a carefully controlled point. It allows protecting internet servers from public access.

In chapter 3 the functions and types of proxy servers and their securities have been described .The use of proxy servers allow internal clients access to internet from behind the firewall, and it's also has a caching ability that make surfing much faster.

In chapter 4 the function public key cryptography algorithms that are useful in internet security are given. This chapter examines the SSL protocol and details how to improve the performance and scalability of Web sites by off-loading SSL processing and object delivery from Web servers.

Finally, chapter 5 has described the software and analyzed the application. A java application program has configured remotely over the internet by its administrator.
All the clients' requests plus the administrator remote configuration are done parallel by using threads. A certain proxy can handle multiple client requests at the same time, plus be remotely controlled by an administrator.

REFERENCES

[1] Mechil,O. " Internet Security & Acceleration Server", May 2000 http://www.microsoft.com/isaserver/

[2] [D Pence] Stateful Inspection versus Proxy Servers by Derek Pence http://www.tersia.com. Vol. 10, No. 5 "October .1998 . 15

[3] "Security Mechanism", http://www.versign.com/wss.pdf.

[4] Adams, C. "Simple and Effective Key Scheduling for Symmetric Ciphers."

Proceeding, Workshop in Selected Areas of Cryptography, SAC" 94. 1994.

[5] "Secure Socket Layers Protocol and Application" Allan Schiffman: Tersia System.Inc{http://www.tersia.com}.

[6] Ahsan, Muninder P. Firewalls: a perpective: Dataquest Perspective. March 8,2000. http://enterprise.cnet.com/enterprise/0-9567-7- 743.html?tag=st.it.9567-7- 2481742.txt.9567-7-2481743.

[7] Ahsan, Muninder P. PIX Firewall 520: Datapro June 17, 1999. http://enterprise.cnet.com/enterprise/0-9567-707-2455823.html?tag=st.it.9567-701-2455823.navreview.9567-707-2455823-235444. Berrmingham, Eileen. Firewalls Solve Network Security Puzzles: CNET.

[8] Curtin, Matt and Marcus J. Ranum. Internet Firewalls: Frequently Asked Question. October 31, 2000. http://www.interhack.net/pubs/fwfaq.

[9] Enterprise. August, 15 2000.http://enterprise.cnet.com/enterprise/0-9567-7-2481742.html?tag=st.it.9567-7-2481743-rost.subdir.9567-7-2481742.Gallegos, Jeff. Interviewed, November 7, 2000 and December 1, 2000.

[10] Wiener, M. "Efficient DES Key Search." Proceeding, Crypto'93 , 1993.

[11] Comparison of Protocols for Secure www.SSL and S-HTTP, National Institute of Standards and Technology, U.S. Department of Commerce.DRAFT, May 1994

[12] R.T. Morris, 1985. A Weakness in the 4.2BSD Unix TCP/IP Software. Computing Science Technical Report No. 117, AT&T Bell Laboratories, Murray Hill, New Jersey.

[13] S.M. Bellovin. Security Problems in the TCP/IP Protocol Suite. Computer Communication Review, Vol. 19, No. 2, pp. 32-48, April 1989.

[14] [Y. Rekhter], R. Moskowitz, D. Karrenberg, G. de Groot, E. Lear, ``Address Allocation for Private Internets." RFC 1918. J.P. Holbrook, J.K. Reynolds. ``Site Security Handbook."

[15] RFC 1244. M. Curtin, ``Snake Oil Warning Signs: Encryption Software to Avoid."6 USENET <sci.crypt> Frequently Asked Questions File.

[16] M. Arlitt and C. Williamson.
``Trace-driven Simulation of Document Caching Strategies for Internet Web Servers" .

[17] H-W Braun and K.C. Claffy.
``Web Traffic Characterization: An Assessment of the Impact of Caching Documents from NCSA's Web Server" .
Computer Networks and ISDN Systems, 28:37-51, 1995.

[18] P. Cao and S. Irani.
``Cost-Aware WWW Proxy Caching Algotithms" .
Technical report, Dept. of Computer Sciences, University of Wisconsin-Madison, 1997.

[19] M.E. Crovella, A. Bestavros, and C.R. Cuhna.
``Characteristics of WWW Client-based Traces" .
Tr-95-010, Computer Science Dept., Boston University, 1995.

[20] http://www.cert.org/tech_tips/home_networks.html

[21] P. Lorenzetti and L. Rizzo. "Replacement Policies for a Proxy Cache" .
http://www.iet.unipi.it/ luigi/caching.ps.gz.

[22] E. Markatos. ``Main Memory Caching of Web Documents" .
In Proceedings of the 5th World Wide Web Conference '1996, Paris, France, May 1996.

[23] S. Williams, M. Abrams, C. Standridge, G. Abdulla, and E. Fox.
``Caching proxies: Limitations and potential" .
In Proceedings of the Fourth World Wide Web Conference '1995, Boston, MA, 1995.

[24] S. Williams, M. Abrams, C. Standridge, G. Abdulla, and E. Fox.
``Removal Policies in Network Caches for World-Wide Web Documents" .
In Proceedings of the ACM SIGCOMM '96, Stanford University, 1996.

[25] R. Wooster and M. Abrams.``Proxy Caching that Estimates Page Load Delays"
In Proceedings of the 6th World Wide Web Conference, Santa Clara, California, 1997.

[26] http://www.microsoft.com/isaserver/ - Internet Security & Acceleration Server

[27] HTTP http://www.w3.org/Protocols/rfc2616/rfc2616.html
http://www.ietf.org/rfc/rfc2617.txt

[28] HTML

http://www.jmarshall.com/easy/html/

[29] SSL
http://developer.netscape.com/docs/manuals/security/sslin/contents.htm

[30] CGI http://www.jmarshall.com/easy/cgi/

[31] Base64
http://www.gjt.org/servlets/JCVSlet/show/ice/com/ice/convert/Base64.java
/head.

[32] JAVA

http://java.sun.com/products/jdk/1.2/docs/api

[33] SAMPLE

http://22C178.cs.uiowa.edu/article/11.html
http://www.cs.technion.ac.il/Courses/Computer-Networks-
Lab/projects/spring97/project1.

APPENDIX A

1. Main
2. Proxy.java
3. HttpRequest.java
4. HttpReply.java
5. Cache.java
6. Config.java
7. Statistic.java
8. Base64.java
9. Deny

```
/************************************************************************
File Name: Proxy.java
************************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;


public class Proxy extends Thread
{
        static Cache cache = null;
        static Config config = null;
        static Statistic statistic = null;

        String localHost = null;
        String localIP = null;

        Socket client ;

        String message = new String();
        boolean running = true;
        URL url;

        long sendTime_ts=0, rcvTime_ts=0;
        long sendTime_tc=0, rcvTime_tc=0;
        int mesLength_ts=0,mesLength_tc=0;
        Date date;

        String websrv = new String();
        int sokid = 0;

        double k = 0.8;
        double EstimatedRTT = 0;
```

```
//main()
public static void main(String args[])
{
        int port;
        ServerSocket srv = null;
        int i = 0;
        String websrv = new String();

        if(args.length > 0)
        {
                try
                {
                        port = Integer.parseInt(args[0]);
                }
                catch (NumberFormatException e)
                {
                        System.out.println("Error: Invalid daemon port");
                        return;
                }

        }

        else
        {
                port = 8000;
        }

        //Initialize the proxy
        try
        {
                System.out.println("\nInitializing...");
                config = new Config();
                config.setLocalHost(InetAddress.getLocalHost().getHostName());
                String tmp = InetAddress.getLocalHost().toString();
                config.setLocalIP(tmp.substring(tmp.indexOf('/')+1));
                config.setProxyPort(":"+port);

                System.out.println("Running Statistic Function ...");
                statistic = new Statistic();

                System.out.println("Creating Cache Manager...");
                cache = new Cache( statistic );
                srv = new ServerSocket(port);
                System.out.println("The server : "+config.getLocalHost() +
config.getProxyPort());
                System.out.println("Proxy up and running ....\n");

                //listen to the cnnection request from the client
```

```
            while(true)
            {
                Socket connectToClient = srv.accept();
                i++;
                Proxy pro = new Proxy(connectToClient, i);
                pro.start();
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
            System.exit(1);
        }

        finally
        {
            try
            {
                srv.close();
            }
            catch(Exception e)
            { }
        }
    }

    //constructor for the Proxy class
    public Proxy(Socket sk, int id)
    {
        localHost = config.getLocalHost();
        localIP = config.getLocalIP();
        client = sk;
        sokid = id;
    }

    public void run()
    {

        HttpRequest req = new HttpRequest(config);
        HttpReply reply = new HttpReply();
        Socket sc=null;

        System.out.println("Proxy thread "+ sokid);

        while(true)
        {
            try
            {
                // Parse information from the request
                req.parseReq(client.getInputStream());
```

```
                    //record time when the client send request
                    date = new Date();
                    rcvTime_tc = date.getTime();
                    statistic.Calculate(sendTime_tc, rcvTime_tc, mesLength_tc, sokid,
"CLIENT");
                    int reqNum = statistic.getReqCount();
                    reqNum ++;
                    statistic.setReqCount(reqNum);

                    //Need authentication ?
                    if(!config.getAuthState())
                    {
                        System.out.println("Not the right client");
                        reply.sendReply(client, reply.formUnauthorized());
                        closeConnection(client);
                        break;
                    }

                    //Need not authentication
                    else
                    {
                        //Request not containing server name
                        if((req.url).compareTo("/") == 0)
                        {
                            websrv = config.getRandHost(Math.random());
                            config.setWeb(websrv);
                            req.url = config.getWeb() + req.url ;
                        }

                        else if(req.url.charAt(0) != 'h')
                        {
                            req.url = config.getWeb() + req.url ;
                        }
                        System.out.println("\nreqUrl--"+req.url + '\n');

                        url = new URL(req.url);

                        //Request containing the local server

if(url.getHost().equalsIgnoreCase(localHost)||url.getHost().equalsIgnoreCase(localIP))
                        {
                            sendLoginPage();
                            break;
                        }

                        //The server is forbidden
                        else if(config.denyWebsite(url.getHost()))
                        {
```

```
                System.out.println("This website is forbidden!");
                reply.sendReply(client, reply.formNotFound());
                closeConnection(client);
                break;
            }

            //The requested webpage has been cached
            else if(cache.IsCached(req.url))
            {
                String s=null;

                int hit = statistic.getHitCount();
                hit ++;
                statistic.setHitCount(hit);

                System.out.println("Getting from cache ...");

                DataInputStream in = new
DataInputStream(cache.getFileInputStream(req.url));
                DataOutputStream out = new
 DataOutputStream(client.getOutputStream());

                while((s=in.readLine())!=null)
                {
                    s += '\n';
                    out.writeBytes(s);
                }
                out.flush();
                in.close();
                client.close();

            }

            //The requested webpage has not been cached yet
            else
            {
                String message=null;
                String s=null;

                //Connect to the server
                try
                {
                    sc=new Socket(url.getHost(),80);
                }
                catch(IOException e)
                {
                    url = new
URL(config.getRandHost(Math.random()));
                    sc=new Socket(url.getHost(),80);
```

```
}
System.out.println("\nConnection:"+sc +'\n');

InputStream insc = sc.getInputStream();
OutputStream outsc = sc.getOutputStream();

InputStream inc = client.getInputStream();
OutputStream outc = client.getOutputStream();

//Rebuild the request from the client and send it
message=req.rebuildHeader() + '\n';

mesLength_ts = message.length();
//Record the time when the request is sent to the server
date = new Date();
sendTime_ts = date.getTime();
outsc.write(message.getBytes());
outsc.flush();

byte buf[] = new byte[4096];
boolean first = true;
String storeMes = new String();
int dataLen = 0;

//Receive reply form the server and cache it
while(true)
{
        int num = insc.read(buf);
        date = new Date();
        if(num>0)
        {
                if(first)
                {
                        //Record the time when the proxy gets the
reply from the server
                        rcvTime_ts = date.getTime();
                        statistic.Calculate(sendTime_ts,
rcvTime_ts, mesLength_ts, sokid, "SERVER");
                        date = new Date();
                        //Record the time when the proxy sends the
reply to the client
                        sendTime_tc = date.getTime();
                        first = false;
                }
                outc.write(buf,0,num);
                outc.flush();

                storeMes += new String(buf,0,num);
                dataLen += num;
```

```
                              }
                              else
                              {
                                   mesLength_tc = dataLen;
                                   cache.Caching(req.url, storeMes);
                                   break;
                              }
                         }
                         if(!(req.connection).equalsIgnoreCase("KEEP-ALIVE"))
                         {
                              closeConnection(client);
                              break;
                         }
                    }
               }//end of else
          }//end of try
          catch (Exception e)
          {
               //System.out.println("Catch an exception in proxy");
               closeConnection(client);
               if(sc != null)
                    closeConnection(sc);
               break;
          }
     }//end of while

     //Display the results of statistics
     System.out.println("\n\tTotal Requests:     " + statistic.getReqCount());

     System.out.println("\tCache Percentage:     " + statistic.calPerCache() + '%');
     System.out.println("\tCache Hit Percentage: " + statistic.calPerHit(
) + '%' + '\n');

}


//Send the login page
public void sendLoginPage()
{
     String page=null;
     String s=null;

     System.out.println("Sending the Login Page ...");

     try
     {
          File loginPage=new File("login");
          String path= loginPage.getAbsolutePath();
```

```
                    System.out.println(""+path);
                    loginPage = new File(path + File.separator + "login");

                    DataInputStream in = new DataInputStream( new
FileInputStream(loginPage));

                    DataOutputStream out = new
DataOutputStream(client.getOutputStream());
                    while((s=in.readLine())!=null)
                          page += s;

                    in.close();

                    out.writeBytes(page);
                    out.flush();
                    out.close();
                    System.out.println("End of the Login Page !");
               }
               catch(Exception e)
               {
                    System.out.println("Error: can't open login html page");
               }

         }

         //Close connection
         private void closeConnection(Socket s)
         {
               try
               {
                    System.out.println("Deconnection: "+s);
                    s.close();
               }
               catch(Exception e)
               {
                    System.err.println(e);
                    System.exit(1);
               }
         }


}
```

```
/***********************************************************************
File: HttpRequest.java
Function: Handle information received from the client
***********************************************************************/

import java.io.*;
import java.util.*;
import java.net.*;

public class HttpRequest
{
      public String method;
      public String url;
      public String file;
      public String version;
      public String referer;
      public String connection;
      public String userAgent;
      public String accept;
      public String headRemainder;
      public String authData;
      Config config;

      public HttpRequest(Config c)
      {
            config = c;
      }

      //Parse information from the header of request.
      public void parseReq ( InputStream in )
      {

            //Header information of the HTTP request
            method = new String();
            url = new String();
            file = new String();
            version = new String();
            referer = new String();
```

```
connection = new String();
userAgent = new String();
accept = new String();
headRemainder = new String();
authData = new String();

DataInputStream din ;
StringTokenizer tz;
String token = new String();

try
{
	din = new DataInputStream(in);
	tz = new StringTokenizer ( din.readLine());
}
catch(Exception e)
{
	return;
}

method = tz.nextToken();
url = tz.nextToken();
version = tz.nextToken();

while(true)
{
	try
	{
		tz = new StringTokenizer (din.readLine());

	}
	catch(Exception e)
	{
		return ;
	}

	try
	{
		token = tz.nextToken();
	}
	catch(NoSuchElementException e)
	{
		break;
	}

	/*
	if(token.equalsIgnoreCase("REFERER:"))
		referer = getRemainder(tz);
```

```
            else if(token.equalsIgnoreCase("CONNECTION:"))
                  connection = getRemainder(tz);

            else if(token.equalsIgnoreCase("USER-AGENT"))
                  userAgent = getRemainder(tz);

            else if(token.equalsIgnoreCase("ACCEPT"))
                  accept = getRemainder(tz);
            */
            if(token.equalsIgnoreCase("CONNECTION:"))
                  connection = getRemainder(tz);
            else if(token.equalsIgnoreCase("AUTHORIZATION:"))
            {
                  authData = tz.nextToken();
                  authData = getRemainder(tz);
            }
            else headRemainder += token + " " + getRemainder(tz) + '\n';
      }

      //If the tag of authentication needs to be set or reset?
      if(authData.length() != 0)
      {
            if(config.isCorrect(authData))
            {
                  config.setAuthState();
            }
      }
      else
            config.resetAuthState();
}

//rebuild the request header after extract the URL
public String rebuildHeader ( )
{
      String header = new String();
      URL tmp;

      if(url.charAt(0) == 'h')
      {
            try
            {
                  tmp = new URL(url);
                  header = method + ' ' + tmp.getFile() + ' ' + version + '\n';
            }
            catch(Exception e)
            {}

      }
      else
```

```
            header = method + ' ' + url + ' ' + version + '\n';

        if(connection.length() > 0)
            header += "Connection: " + connection + '\n';
        header += headRemainder;

        return header;
    }

    private String getRemainder(StringTokenizer tk)
    {
        String str = new String();
        while (tk.hasMoreTokens())
            str += tk.nextToken();
        return str;
    }

}


/*************************************************************************
File: HttpReply.java
Function: Build reply messages for the proxy and send the message
*************************************************************************/


import java.net.*;
import java.io.*;
import java.util.*;

public class HttpReply
{
    static String CR="\r\n";
    static String HTTP_PROTOCOL="HTTP/1.1";
    static String HTTP_SERVER="Java Proxy Server";

    //Form unauthorized reply
    public String formUnauthorized()
    {
        String header = new String();
        File file = new File("auth.html");
        FileInputStream in ;

        header += "HTTP/1.1 401 Authorization Required" + CR;
        header += "Date: " + new Date() + CR;
        header += "Server: " + HTTP_SERVER +CR;
        header += "WWW-Authenticate: Basic realm=\"Java Proxy\"" + CR;
        header += "Connection: close" + CR;
        header += "Content-Type: text/html" + CR;
        header +=  CR;
        header +=  CR;

        try
        {
            in = new FileInputStream(file);
            byte[] buf = new byte[4096];
            int num = in.read(buf);
            header += new String(buf,0,num);
            header += CR;
            in.close();
        }
        catch(Exception e)
        {}

        return header;
    }

    //Form "not found" reply
    public String formNotFound()
    {
        return formError("404 Not_found","Requested object was not found");
    }

    //Send reply message
    public void sendReply(Socket s, String mes)
    {
        OutputStream out;

        try
        {
            out = s.getOutputStream();
            out.write(mes.getBytes());
            out.flush();
        }
        catch(Exception e)
        {}
    }

    //Form general reply message
    private String formError(String Error,String Description)
    {
        String out = new String();

        out += HTTP_PROTOCOL +" " + Error + CR;
        out +="Server: " + HTTP_SERVER   + CR;
        out +="Content-type: text/html"  + CR;
        out +="Last-Modified: " + new Date() +CR;
        out += CR;
        //Generate Error Body
        out  ="<HTML><HEAD>" + CR;
        out += "<TITLE>" + Error + "</TITLE>" + CR;
        out +="</HEAD><BODY>" + CR;
        out +="<H2>" + Error +"</H2>" + CR;
        out += Description + "<P>" + CR;
        out +="</BODY></HTML>" + CR;

        return out;
    } }
/*************************************************************************
File: Cache.java
Function: Handle information received from the server
*************************************************************************/

import java.util.*;
import java.io.*;
import java.net.*;

public class Cache
{

    public Hashtable fileTable = new Hashtable();

    public String version;
    public String ret;
    public String statu;
    public String date;
    public String connection;
    public String contLength;
    public String replyRemainder;

    private File[] fileList = null;

    public String basepath;
    private File cacheDir = new File("cache");

    private long MaxDirSpace = 180000;
    private long CurFreeSpace = MaxDirSpace;

    Statistic statistic;

    //Constructor
    public Cache( Statistic st)
    {

        statistic = st;
        cacheDir.mkdirs();
        basepath = cacheDir.getAbsolutePath();

        fileList = cacheDir.listFiles();

        for(int i=0; i<fileList.length; i++)
        {
            long modiDate = fileList[i].lastModified();
            fileTable.put(fileList[i].getName(), new Date(modiDate) );
            CurFreeSpace -= fileList[i].length();
        }
        System.out.println("Current Free Space :"+CurFreeSpace);
    }

    //Parse information of the reply got from the server
    public void parseReply(InputStream in)
    {
        version = new String();
        ret = new String();
        date = new String() ;
        connection = new String();
        contLength = new String();
        replyRemainder = new String();

        DataInputStream din ;
        StringTokenizer tz;
        String token = new String();

        din = new DataInputStream(in);
        while(true)
        {
            try
            {
                tz = new StringTokenizer ( din.readLine());
            }
            catch(Exception e)
            {
                return;
            }

            version = tz.nextToken();
            if((version.substring(0,3)).equalsIgnoreCase("HTTP"))
            {
                ret = tz.nextToken();
                statu = tz.nextToken();
                break;
            }

        }
        while(true)
        {
            try
            {
                tz = new StringTokenizer(din.readLine());
            }
            catch(Exception e)
            {
                return;
            }

            try
            {
                token = tz.nextToken();
            }
            catch(NoSuchElementException e)
            {
                break;
            }

            if(token.equalsIgnoreCase("DATE:"))
                date = getRemainder(tz);
            else if(token.equalsIgnoreCase("CONTENT-LENGTH:"))
                contLength = getRemainder(tz);
            else if(token.equalsIgnoreCase("CONNECTION:"))
                connection = getRemainder(tz);
            else replyRemainder += token + " " + getRemainder(tz);

        }
    }

    //Rebuild the reply
    public String rebuildReply()
    {
        String reply = new String();

        reply = version + " " + ret + " " + statu + '\n';

        reply += "Date: " + date + '\n';
        reply += "Connection: " + connection + '\n';
        reply += "Content-Length: " + contLength + '\n';
        reply += replyRemainder;

        return reply;
    }

    //If the webpage has cached?
    public boolean IsCached(String rawurl)
    {
        String fileName = getFileName(rawurl);

        if(fileName == null)
            return false;

        return(fileTable.get(fileName) != null);
    }

    public FileInputStream getFileInputStream(String rawurl)
    {
        FileInputStream in = null;
        try
        {
            String filename = basepath + "/" + getFileName(rawurl);

            in = new FileInputStream(filename);
        }
        catch (FileNotFoundException fnf)
        {
            try

            {
                System.out.println("File Not Found:"+getFileName(rawurl)+"
"+fnf);
            }
            catch (Exception e)
            {}
        }
        finally
        {
            return in;
        }
    }


    //Cache the webpage if it's cachable
    public synchronized void Caching(String rawurl, String content)
    {

        FileOutputStream out = null;
        byte[] line ;
        URL url = null;
        String fileName = new String();
        boolean cachable = false;

        while(CurFreeSpace < content.length())
            CacheDelete();

        fileName = getFileName(rawurl);
```

```
if(fileName != null)
        cachable = true;
try
{
        url = new URL(rawurl);

        StringTokenizer tz;
        tz = new StringTokenizer(content.substring(0,
                        content.indexOf("\n")));
        String retCode = tz.nextToken();
        retCode = tz.nextToken();

        if ((retCode.equals("200") || retCode.equals("302") ||
                retCode.equals("304")) && ((url.getFile()).compareTo("/") == 0))
                cachable = true;
}
catch(MalformedURLException e)
{
        cachable = false;
}

if(cachable)
{
        System.out.println("Caching this reply ...");
        try
        {
                out = new FileOutputStream (basepath+"/"+
                                        fileName);

                line = new byte[content.length()];
                line = content.getBytes();
                out.write(line);

                fileTable.put(fileName,new Date());

                int cacheNum = statistic.getCacheCount();
                cacheNum++;
                statistic.setCacheCount(cacheNum);
        }
        catch(Exception e)
        {}
        finally
        {
                try
                {
                        out.close();
                }
                catch(Exception e)
                {}
                }
            }
        }

        //Cache all replys got from the server
        public synchronized void CacheAll(String rawurl, String content, int Id)
        {

            FileOutputStream out = null;
            byte[] line ;
            URL url = null;
            String fileName = new String();
            boolean cachable = false;

            while(CurFreeSpace < content.length())
                CacheDelete();

            System.out.println("Caching this reply ...");
            try
            {
                url = new URL(rawurl);
                fileName = url.getHost() + Id;

                out = new FileOutputStream(basepath+"/"+ fileName);

                line = new byte[content.length()];
                line = content.getBytes();
                out.write(line);

                fileTable.put(fileName,new Date());
            }
            catch(Exception e)
            {}
            finally
            {
                try
                {
                    out.close();
                }
                catch(Exception e)
                {}
            }
        }

        //Parse file name from the String
        private String getFileName(String urlName)
        {
            String filename = urlName.substring(7).replace('/' , '_') ;
        if (filename.indexOf('?') != -1 || filename.indexOf("cgi-bin") != -1)
        {
            return null;
        }

        return filename;
    }

    //delete the earliest cache file
    private synchronized void CacheDelete()
    {
        String fileDelete = null;
        long lastTime = 0;
        long fileLen = 0;

        System.out.println("There isn't enough space, deleting file ...");
        fileList = cacheDir.listFiles();
        for(int i=0; i<fileList.length; i++)
        {
            long modiDate = fileList[i].lastModified();
            if(i==0)
            {
                lastTime = modiDate;
                fileDelete = fileList[i].getName();
                fileLen = fileList[i].length();
            }
            else if (modiDate < lastTime)
            {
                lastTime = modiDate;
                fileDelete = fileList[i].getName();
                fileLen = fileList[i].length();

            }
        }
        System.out.println("Deleting  " + fileDelete + " ...");

        File file = new File(basepath + File.separatorChar + fileDelete);
        if(file.delete())
        {
            CurFreeSpace += fileLen;
            fileTable.remove(fileDelete);
            System.out.println("One file has been deleted!");

        }
    }

    private String getRemainder(StringTokenizer tk)
    {
        String str = new String();
```

```
            while (tk.hasMoreTokens())
                  str += tk.nextToken();

            return str;
      }


}


/*********************************************************************
File: Config.java
Function: Config the proxy, set or get data or status for the proxy
*********************************************************************/

import java.util.*;
import java.net.*;
import java.io.*;
import java.math.*;

public class Config
{
      private String localHost;
      private String localIP;
      private String proxyPort;
      private String websrv;
      private String authData;
      private boolean authState;

      //Initialiszee userid and password, authentication status and default server name
      public Config()
      {
            websrv = "http://www.yahoo.com";
            authState = false;
            authData = "Aladdin:open sesame";
      }

      //Set new userid and password
      public void setAuthData( String s)
      {
            authData = s;
      }

      //Set authentication status as "has authenticated"
      public void setAuthState()
      {
            authState = true;
      }
```

```
//Clear authentication status as "need authentication
public void resetAuthState()
{
      authState = false;
}

//Get authentication status
public boolean getAuthState()
{
      return authState;
}

//Initialize the proxy address
public void setLocalHost(String host)
{
localHost = host;
}

public String getLocalHost()
{
return localHost;
}

public void setLocalIP(String ip)
{
localIP = ip;
}

public String getLocalIP()
{
return localIP;
}

public void setProxyPort(String port)
{
      proxyPort = port;
}

public String getProxyPort()
{
      return proxyPort;
}

public void setWeb(String webname)
{
      websrv = webname;
}
```

```
public String getWeb()
{
      return websrv;
}

//Get a random server name
public String getRandHost(double randnum)
{
      String line = new String();
      String temp = null;
      int count = 0;
      int nameNum = 0;
      String[] nameList = new String[0] ;
      StringTokenizer s;
      System.out.println("randnum:"+randnum);
      try
      {
            File hostFile = new File ("hostname");
            DataInputStream fileIn = new DataInputStream( new
FileInputStream(hostFile));
            do
            {
                  temp = fileIn.readLine() ;
                  line += temp + " ";
                  count++;
            }while(temp != null);

            s = new StringTokenizer (line);
            nameList = new String[count] ;

            for(int i=0;i<count;i++)
                  nameList[i]=s.nextToken();
            fileIn.close();
      }
      catch(Exception e)
      {
      }

      nameNum = (int) ((count-2) * randnum);
      return nameList[nameNum];

}

//If the server name is in denyID file?
public boolean denyWebsite(String hostName)
{
      File dr = new File("deny");
      String path = dr.getAbsolutePath();
      File file = new File(path + '/' + "denyID");
```

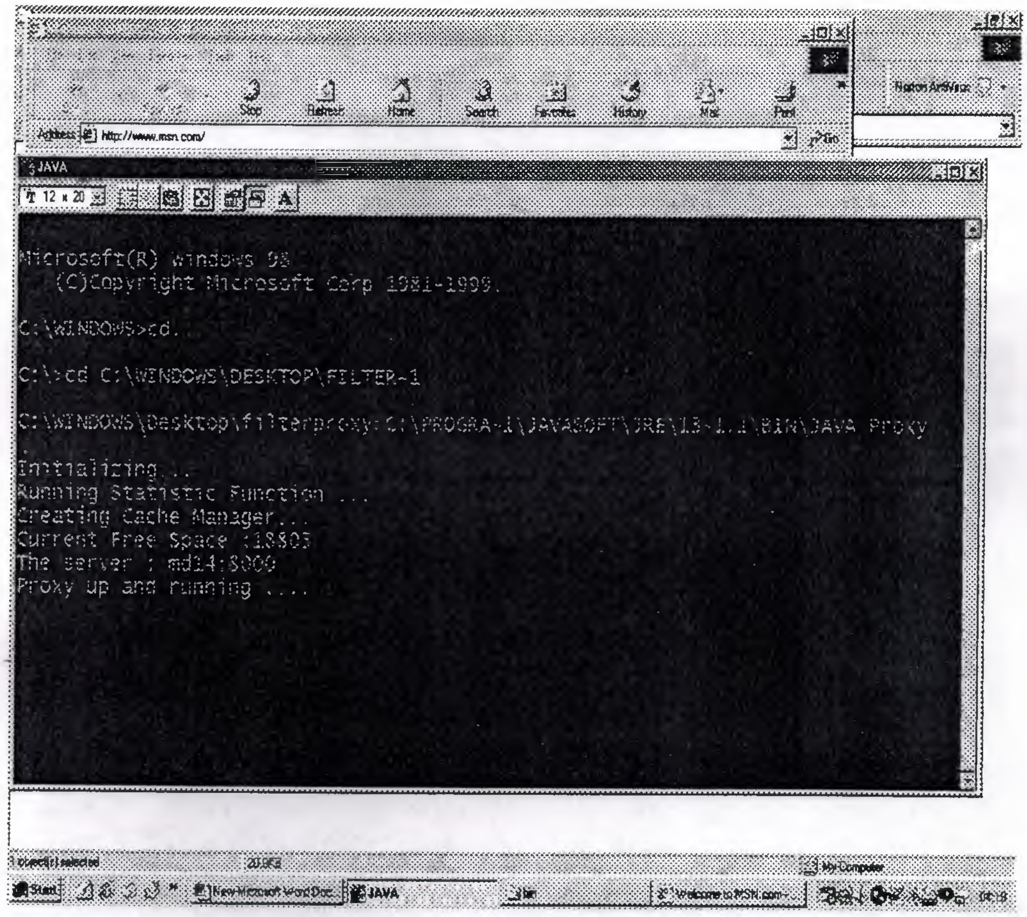
```
        DataInputStream fileIn;
        String line = new String();
        String temp = null;
        int count = 0;
        int nameNum = 0;
        String[] nameList = new String[0] ;
        StringTokenizer s;

        try
        {
            fileIn = new DataInputStream(new FileInputStream(file));

            do
            {
                temp = fileIn.readLine() ;
                line += temp + " ";
                count++;
            }while(temp != null);

            s = new StringTokenizer (line);
            nameList = new String[count] ;

            for(int i=0;i<(count-2);i++)
            {
                nameList[i]=s.nextToken();
            }
            fileIn.close();
        }
        catch(Exception e)
        {
            return true;
        }

        for(int i=0;i<(count-2); i++)
        {
            if(hostName.compareTo(nameList[i]) == 0)
                return true;
        }

        return false;
    }

    //If the userid and password are correct?
    public boolean isCorrect(String s)
    {
        StringBufferInputStream rawStr;
        StringBuffer desStr;
        Base64 base = new Base64();
        try
        {
            rawStr = new StringBufferInputStream(s);
            desStr = new StringBuffer();
            base.decode(rawStr, desStr);
            String tmpStr = desStr.substring(0);
            if(tmpStr.compareTo(authData) == 0)
                return true;
            return false;
        }
        catch(Exception e)
        {
            return false;   }}}

/************************************************************************
File: Statistic.java
Function: Handle some calculations
************************************************************************/

import java.util.*;

public class Statistic
{
    private int hitCount;
    private int reqCount;
    private int cacheCount;
    private int SampleRTT;
    private int EstimatedRTT;
    private long effBandwidth;

    //Constructor
    public Statistic()
    {
        cacheCount = 0;
        reqCount = 0;
        hitCount = 0;
        SampleRTT = 0;
        EstimatedRTT = 0;
        effBandwidth = 0;
    }

    //Set the number which represents how many times the cached file has been hit
    public void setHitCount(int num)
    {
        hitCount = num;
    }

    public int getHitCount()
    {
        return hitCount;
    }

    //Set the number which represents how many requests the client has sent
    public void setReqCount(int num)
    {
        reqCount = num;
    }

    public int getReqCount()
    {
        return reqCount;
    }

    //Set the number which represents how many files has been cached
    public void setCacheCount(int num)
    {
        cacheCount = num;
    }

    public int getCacheCount()
    {
        return cacheCount;
    }

    //Calculate persentage of caching
    public int calPerCache()
    {
        if(reqCount == 0)
            return 0;
        else
            return((int)(cacheCount * 100 / reqCount));
    }

    //Calculate persentage of hitting the cached files
    public int calPerHit()
    {
        if(reqCount == 0)
            return 0;
        else
            return((int)(hitCount * 100 / reqCount));
    }

    //Calculate RTT and Effective Bandwidth
    public void Calculate(long sendTime, long rcvTime, int length, int id,
String derection)
    {

        if(sendTime == 0)
```

```
                return;

            SampleRTT = (int) (rcvTime - sendTime);
            System.out.println("\n\tRTT from PROXY to " + derection + " is: " +
SampleRTT + " milliseconds");

            effBandwidth = length / SampleRTT * 8 *1000;
            System.out.println("\tEffective Bandwidth from PROXY to " + derection + "
is: " + effBandwidth + " bps\n" );
        }

        public int getSampleRTT()
        {
            return SampleRTT;
        }

        public long getBandwidth()
        {
            return effBandwidth;
        }}

/**********************************************************************
This code is found on
"http://www.gjt.org/servlets/JCVSlet/show/ice/com/ice/convert/Base64.jav
a/HEAD".
I modified it in order to be used in my program

File: Base64.java
Function: Decode the userid and password
**********************************************************************/


import java.io.*;
import java.lang.*;

public class Base64
{
        private static final char   pad64 = '=';
        private static final String charSet =

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+
/";

        //public static int decode(Reader reader, OutputStream output)
        public static int decode(StringBufferInputStream str,StringBuffer sout)
                    throws IOException
        {
            char    inChar;
```

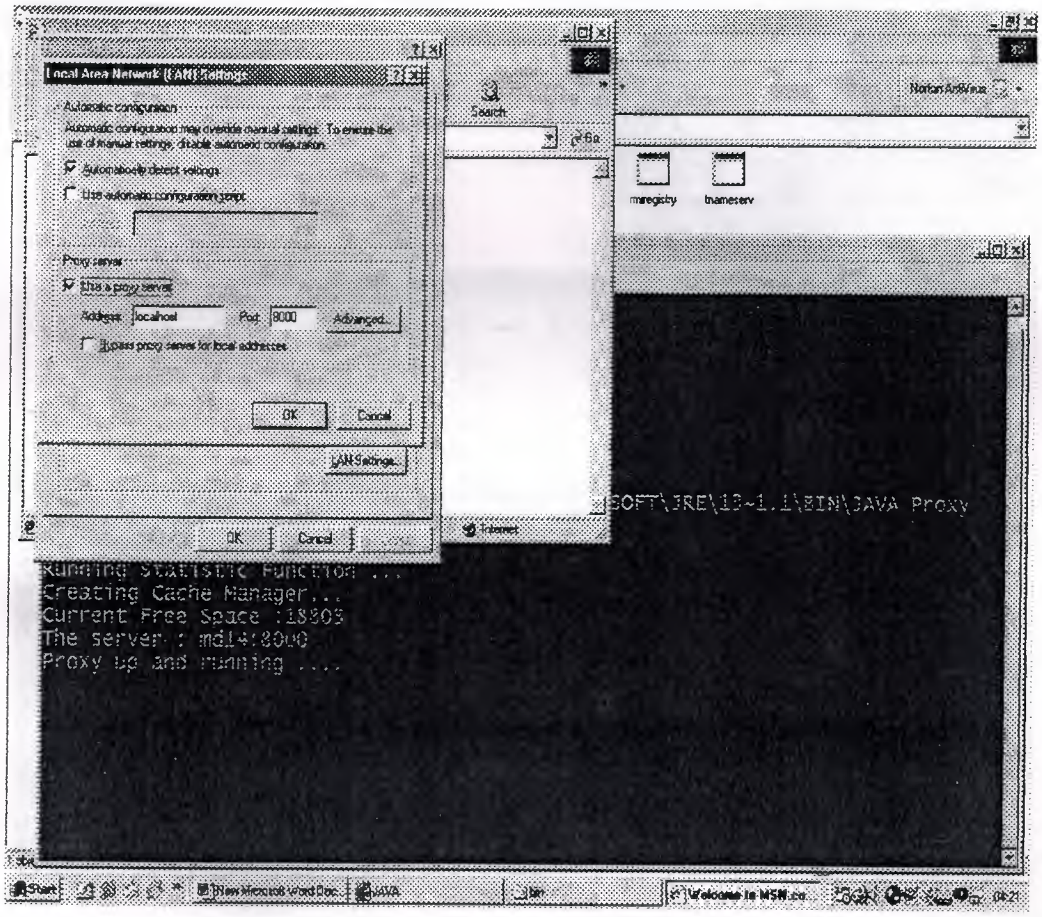
```
byte[]  outNibble = new byte[3];
int     state = 0;

for ( ; ; )
{
     //inChar = (char) reader.read();
     inChar = (char) str.read();
     if ( inChar == -1 )
          break;

     if ( Character.isWhitespace( inChar ) )
          continue;

     if ( inChar == Base64.pad64 )
          break;

     int pos = Base64.charSet.indexOf( inChar );
     if ( pos == -1 ) // A non-base64 character
          return -1;

   //   System.err.println( "IN '" + inChar + "'  POS " + pos);

     switch ( state )
     {
          case 0:
               outNibble[0] = outNibble[1] = outNibble[2] = 0;
               outNibble[0] = (byte) ( pos << 2 );

               state = 1;
               break;

          case 1:
               outNibble[0] |= (pos >> 4);
               outNibble[1] = (byte) ( (pos & 0x0f) << 4 );
               state = 2;
               break;

          case 2:
               outNibble[1] |= (pos >> 2);
               outNibble[2] = (byte) ( (pos & 0x03) << 6 );
               state = 3;
               break;

          case 3:
               outNibble[2] |= pos;
     //        System.err.println
     //           (     "[0] '" + outNibble[0]
     //              + "' [1] '" + outNibble[1]
     //              + "' [2] '" + outNibble[2] + "'" );
```

```
                    sout.append((char)outNibble[0]);
                    sout.append((char)outNibble[1]);
                    sout.append((char)outNibble[2]);
                    //output.write( outNibble[0] );
                    //output.write( outNibble[1] );
                    //output.write( outNibble[2] );
                    state = 0;
                    break;
        } }

        // We are done decoding Base-64 chars.  Let's see if we ended
        // on a byte boundary, and/or with erroneous trailing characters.

    if ( inChar == Base64.pad64 ) // We got a pad char
    {
        //inChar = (char) reader.read();
        inChar = (char) str.read();

        switch ( state )
        {
            case 0:     // Invalid = in first position
            case 1:     // Invalid = in second position
                return -1;

            // UNDONE - no integrity checks yet, should check
            // that there are correct number of pads per state.
            //
            case 2:
              //output.write( outNibble[0] );
              sout.append((char)outNibble[0]);
              break;

           case 3:
              sout.append((char)outNibble[0]);
              sout.append((char)outNibble[1]);
              //output.write( outNibble[0] );
              //output.write( outNibble[1] );
              break;
        }
    }
    else if ( state != 0 )
    {
          // We ended by seeing the end of the stream.
          // But we had unprocessed and unpadded bytes!
        return -1;
    }
    return 0;    }}
```

APPENDIX B



```
Microsoft(R) Windows 98
   (C)Copyright Microsoft Corp 1981-1999.

C:\WINDOWS>cd..

C:\>cd C:\WINDOWS\DESKTOP\FILTER~1

C:\WINDOWS\Desktop\filterproxy>C:\PROGRA~1\JAVASOFT\JRE\13~1.1\BIN\JAVA Proxy
.
Initializing ..
Running Statistic Function ...
Creating Cache Manager...
Current Free Space :18803
The server : md14:8009
Proxy up and running ....
```



Local Area Network (LAN) Settings

Automatic configuration

Automatic configuration may override manual settings. To ensure the use of manual settings, disable automatic configuration.

[✓] Automatically detect settings

[ ] Use automatic configuration script

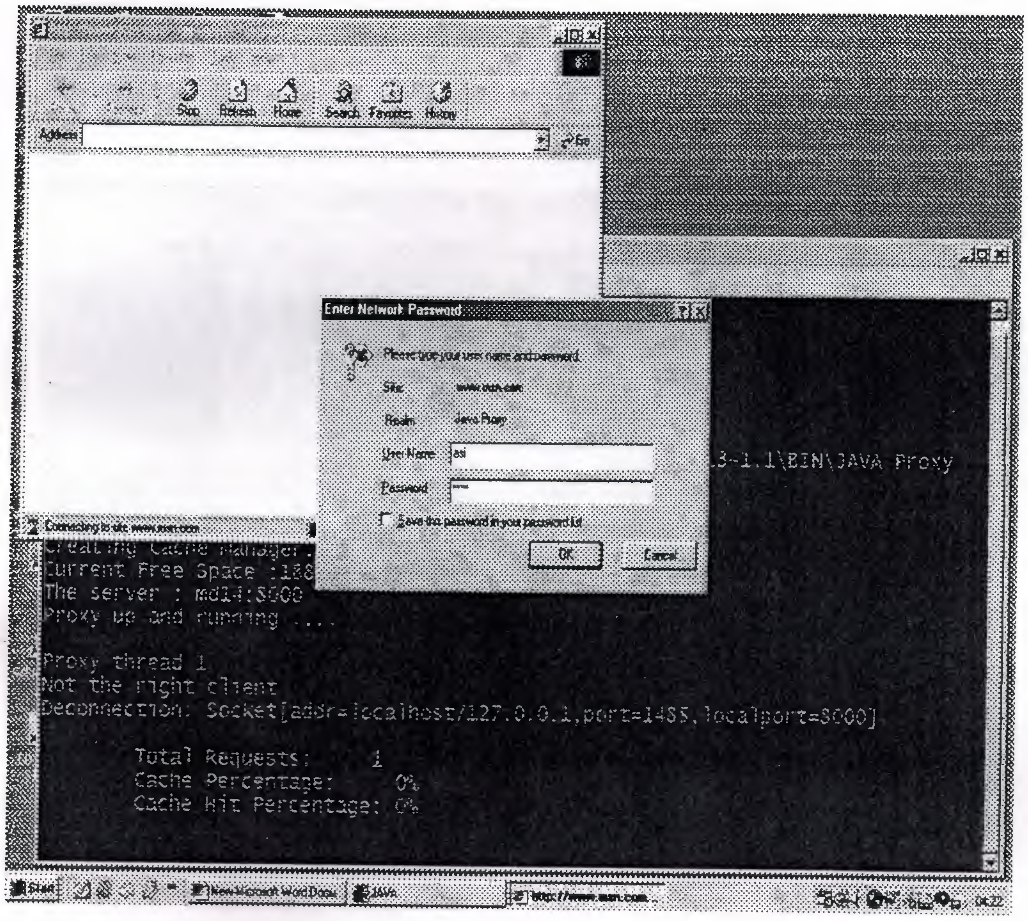
Proxy server

[✓] Use a proxy server

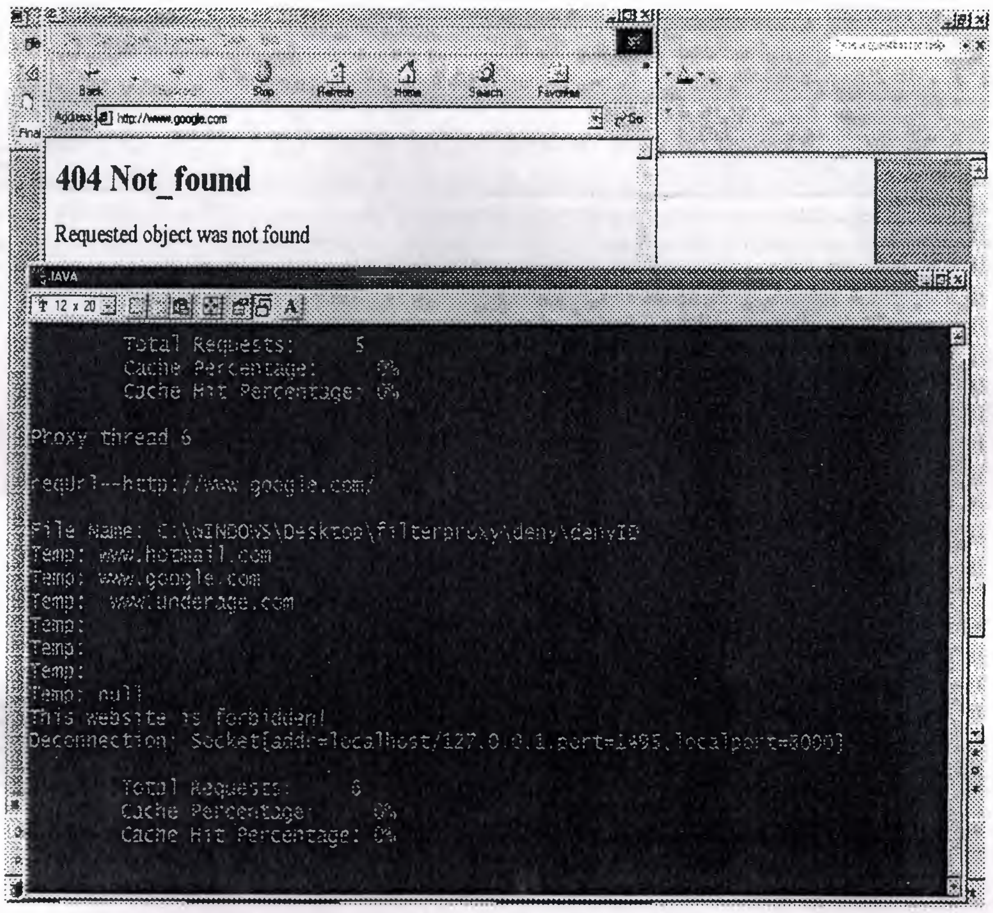
Address: localhost Port: 8000

[ ] Bypass proxy server for local addresses

```
SOFT\JRE\13~1.1\BIN\JAVA Proxy
Running Statistic Function ...
Creating Cache Manager...
Current Free Space :18803
The server : md14:8000
Proxy up and running ....
```



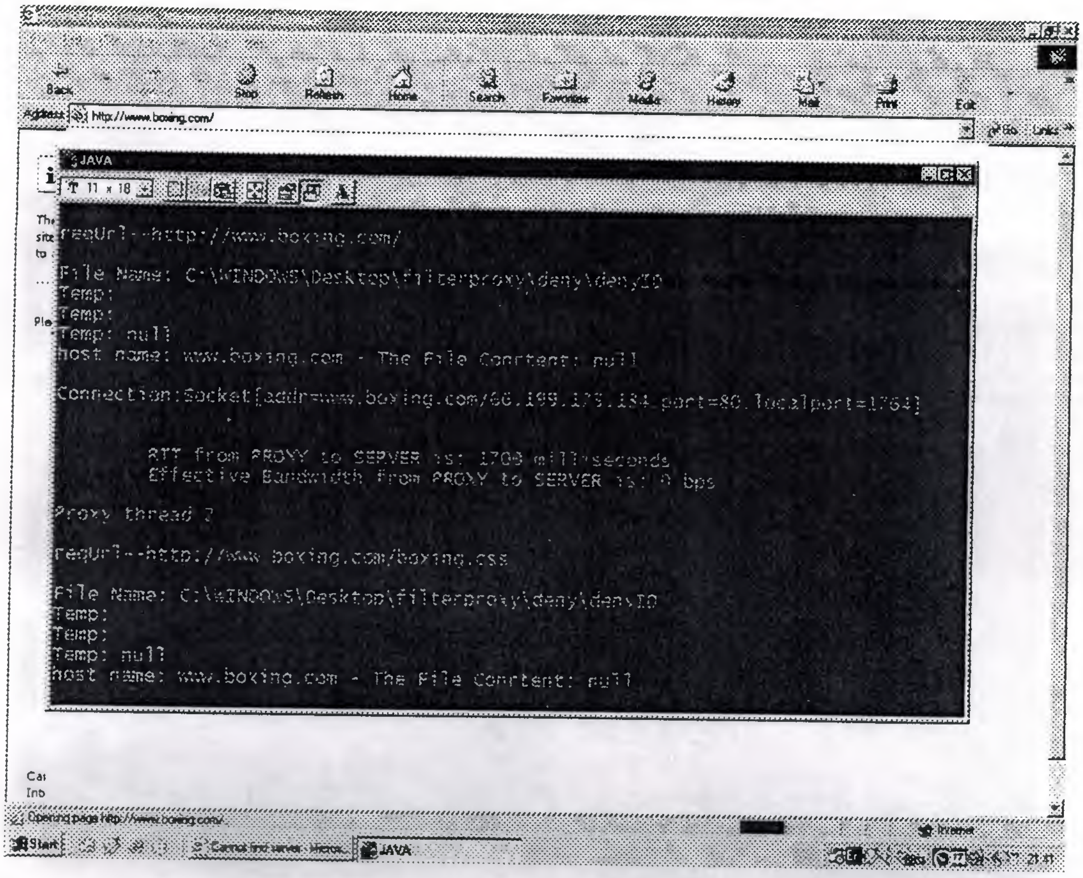
```
                                        3-1.1\BIN\JAVA Proxy

Creating Cache Manager.
Current Free Space :188
The server : md14:8000
Proxy up and running ....

Proxy thread 1
Not the right client
Deconnection: Socket[addr=localhost/127.0.0.1,port=1485,localport=8000]

        Total Requests:      1
        Cache Percentage:      0%
        Cache Hit Percentage: 0%
```



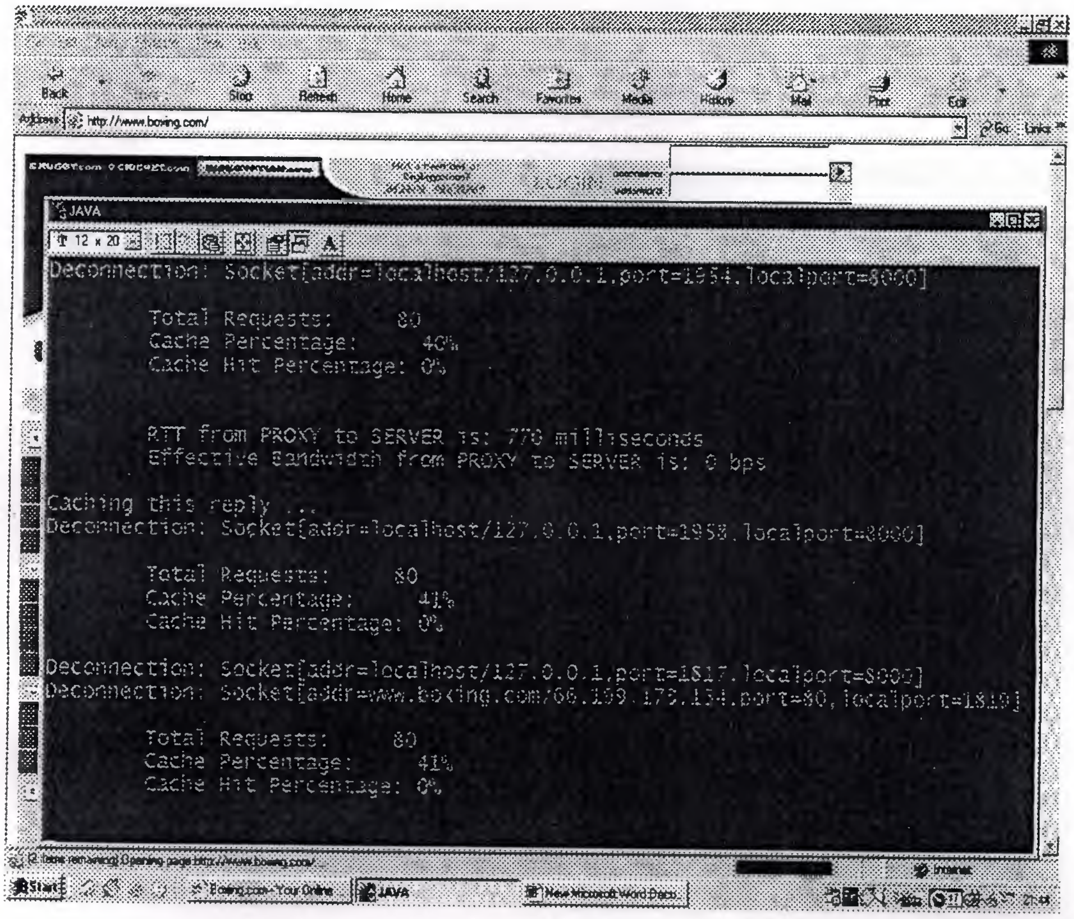
404 Not_found

Requested object was not found

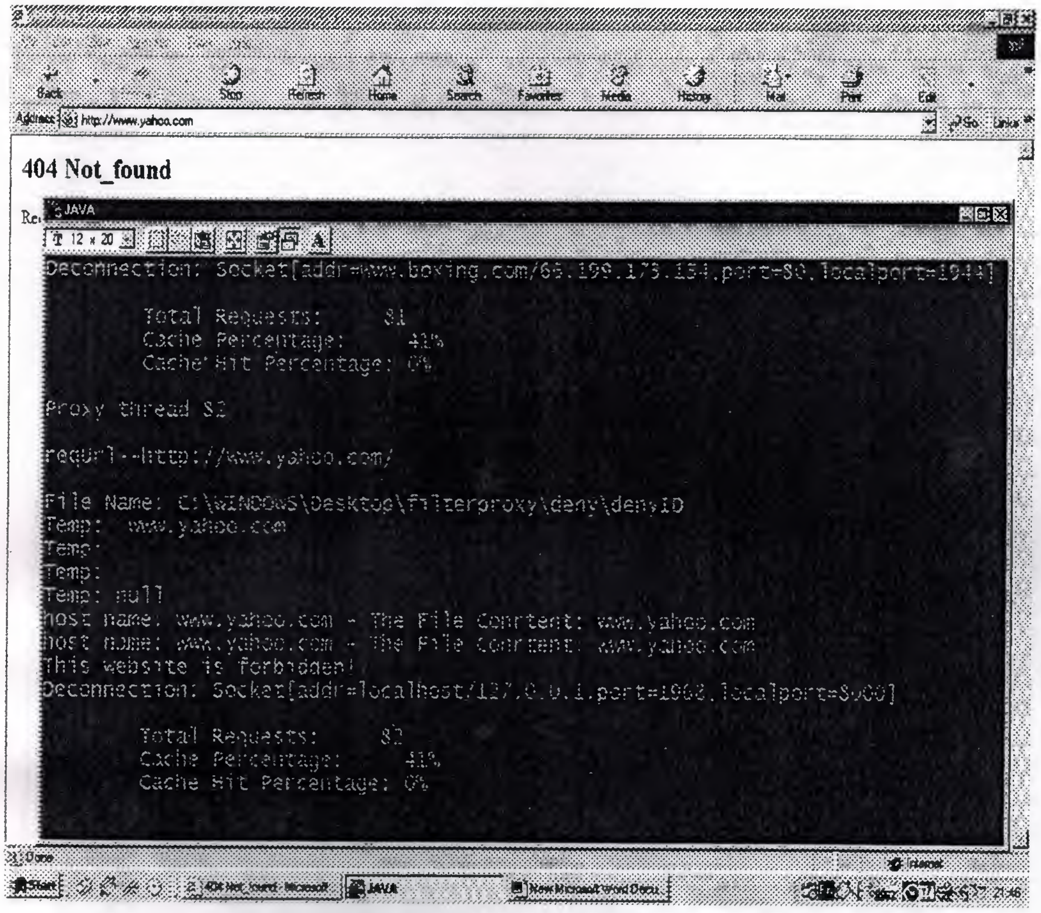
```
        Total Requests:      5
        Cache Percentage:      0%
        Cache Hit Percentage: 0%

Proxy thread 6

reqUrl--http://www.google.com/

File Name: C:\WINDOWS\Desktop\filterproxy\deny\denyID
Temp: www.hotmail.com
Temp: www.google.com
Temp:  www.underage.com
Temp:
Temp:
Temp:
Temp: null
This website is forbidden!
Deconnection: Socket[addr=localhost/127.0.0.1,port=1495,localport=3000]

        Total Requests:      6
        Cache Percentage:      0%
        Cache Hit Percentage: 0%
```


Back
Stop
Refresh
Home
Search
Favorites
Media
History
Mail
Print
Edit
Address http://www.boxing.com/
Go
Links
JAVA
Opening page http://www.boxing.com/...
Internet
Start
Cannot find server - Micros...
JAVA

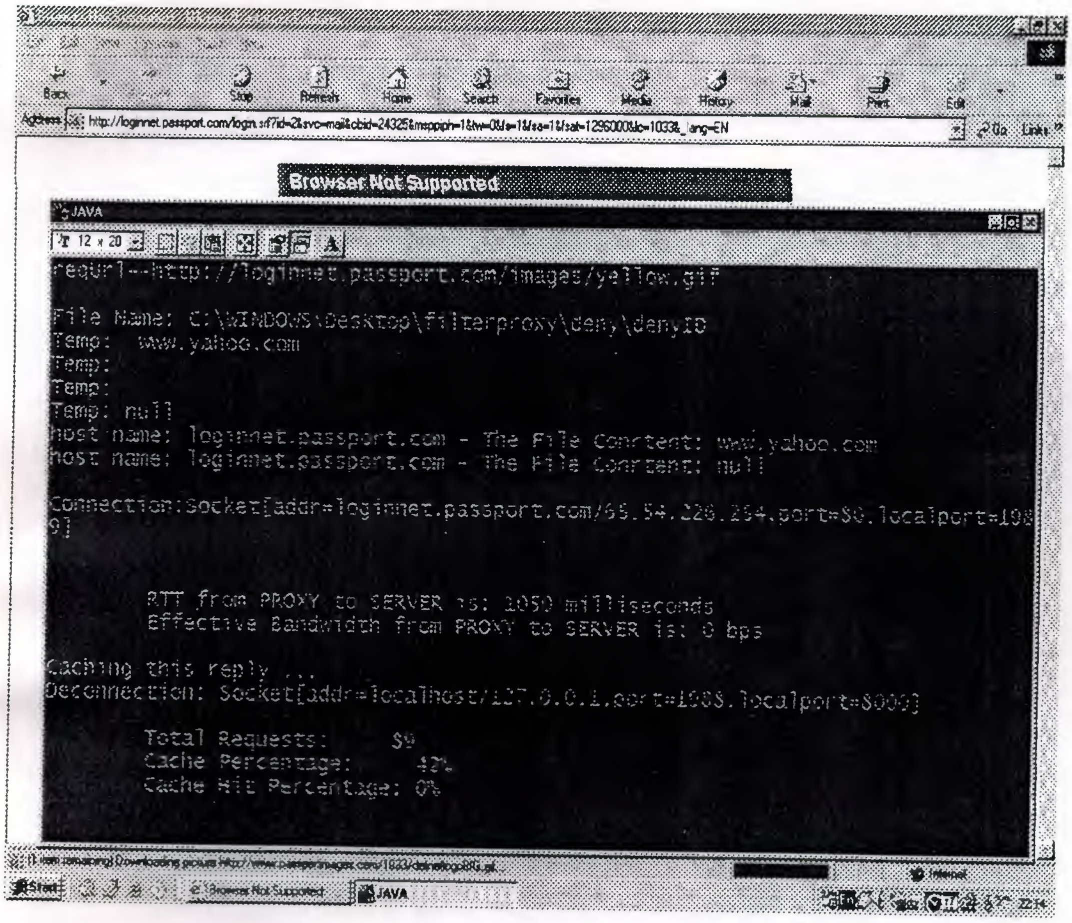
```
reqUrl--http://www.boxing.com/

File Name: C:\WINDOWS\Desktop\filterproxy\deny\denyIO
Temp:
Temp:
Temp: null
host name: www.boxing.com - The File Conrtent: null

Connection:Socket[addr=www.boxing.com/66.199.173.134,port=80,localport=1764]


        RTT from PROXY to SERVER is: 1700 milliseconds
        Effective Bandwidth from PROXY to SERVER is: 0 bps

Proxy thread 7

reqUrl--http://www.boxing.com/boxing.css

File Name: C:\WINDOWS\Desktop\filterproxy\deny\denyIO
Temp:
Temp:
Temp: null
host name: www.boxing.com - The File Conrtent: null
```



```
Deconnection: Socket[addr=localhost/127.0.0.1,port=1954,localport=8000]

        Total Requests:      80
        Cache Percentage:      40%
        Cache Hit Percentage: 0%


        RTT from PROXY to SERVER is: 770 milliseconds
        Effective Bandwidth from PROXY to SERVER is: 0 bps

Caching this reply ...
Deconnection: Socket[addr=localhost/127.0.0.1,port=1958,localport=8000]

        Total Requests:      80
        Cache Percentage:      41%
        Cache Hit Percentage: 0%

Deconnection: Socket[addr=localhost/127.0.0.1,port=1817,localport=8000]
Deconnection: Socket[addr=www.boxing.com/66.199.179.134,port=80,localport=1819]

        Total Requests:      80
        Cache Percentage:      41%
        Cache Hit Percentage: 0%
```


404 Not_found

```
Deconnection: Socket[addr=www.boxing.com/66.199.173.134,port=80,localport=1944]

        Total Requests:     81
        Cache Percentage:     41%
        Cache Hit Percentage: 0%

Proxy thread 82

reqUrl--http://www.yahoo.com/

File Name: C:\WINDOWS\Desktop\filterproxy\deny\denyID
Temp:  www.yahoo.com
Temp:
Temp:
Temp: null
host name: www.yahoo.com - The File Conrtent: www.yahoo.com
host name: www.yahoo.com - The File Conrtent: www.yahoo.com
This website is forbidden!
Deconnection: Socket[addr=localhost/127.0.0.1,port=1968,localport=8000]

        Total Requests:     82
        Cache Percentage:     41%
        Cache Hit Percentage: 0%
```



```
requrl--http://loginnet.passport.com/images/yellow.gif

File Name: C:\WINDOWS\Desktop\filterproxy\deny\denyID
Temp:  www.yahoo.com
Temp:
Temp:
Temp: null
host name: loginnet.passport.com - The File Conrtent: www.yahoo.com
host name: loginnet.passport.com - The File Conrtent: null

Connection:Socket[addr=loginnet.passport.com/65.54.228.254,port=80,localport=108
9]



        RTT from PROXY to SERVER is: 1050 milliseconds
        Effective Bandwidth from PROXY to SERVER is: 0 bps

Caching this reply ...
Deconnection: Socket[addr=localhost/127.0.0.1,port=1088,localport=8000]

        Total Requests:     59
        Cache Percentage:     42%
        Cache Hit Percentage: 0%
```