NSC White Paper

A Web Server Primer: *A **PC** Perspective*

Date: November 3, 1998
Authors: Trish Sundgaard, Alan Bodiford, and Sean Takats

Abstract

Among the wealth of information about the Web on the Web, not much exists between "click here for cool stuff" and highly technical and/or product-specific documentation—at least about Web servers. A significant portion of what documentation is available about Web servers, scarce as it is, examines the topic of Web servers from a UNIX-only view; PC-familiar people may find it unfathomable without significant study. This paper provides a concise overview of the basics of this technology by examining the Web server side of the World Wide Web from the PC perspective. Topics covered include communications, configuration and performance, forms and scripts, gateways, Java, and security.

Table of Contents

Introduction

Written for users accustomed to a PC perspective, this paper describes the fundamental functions, attributes, and terms associated with the *Web server* side of the World Wide Web network computing model. While many people find the Web fascinating, most have neither the time (nor the endurance) to slog through and find the information available. When they do, often they may find information either too technical or too simplistic to coherently introduce the topic.

This document attempts to find a balance within that spectrum. Busy people with some familiarity with the Web and even minimal technical background should find this a good introductory paper on Web servers; hence, the name *primer*. Our discussion describes Web servers and how they work. For brevity's sake, we exclude other Internet applications such as FTP, telnet, SNMP, etc. In this paper we explore the following:

I. Web Servers - Overview
II. Configuration and Performance
III. Forms & Scripts, Gateways, and Java
IV. Web Server Security

Web Servers - Overview

What are Web Servers?

A Web server is a network-attached computer running software that can access programs and data arranged in a file hierarchy. The server schedules web client (browser) requests and responds to those requests. Web servers log activity. World Wide Web servers do these things over the Internet.

Web servers differ from one another in terms of their file structure, naming conventions, and the like. They do, however, share some commonalties:

A directory for configuration files
A directory for HTML documents
A directory for executables.

The main configuration file will specify the port number the server runs on, who has administrative access to the server, and the names and directories where other configuration files are located. IBM's Internet Connection Servers, Lotus' Domino Go, and Domino Go Pro servers offer Configuration Forms that you access with a browser to modify the server's configuration. Domino uses a Lotus Notes client to set up. Microsoft and Netscape also offer setup through a graphical user interface. Other files you will find in most Web servers include: an access control file to define who may access what information on the server and a MIME (*Multipurpose Internet Mail Extension)* type mapping file and to identify the file extensions associated with MIME content types. MIME content types tell the browser what kind of data its response contains, which leads us to the next question …

How do servers and browsers communicate?

All computers communicate on the Internet using TCP/IP *(Transmission Control Protocol/Internet Protocol)*, a protocol "suite" composed of numerous protocols providing various functions at the different layers of the suite. Often you will hear this referred to as the TCP/IP *protocol stack* due to its layered model. An HTTP (*Hypertext Transfer Protocol)* exchange takes place over a TCP/IP *socket* that closes at the end of the exchange. (In brief, sockets request network services from the operating system and serve as the endpoint of the connection...but a detailed discussion of sockets is beyond the scope of this document. This is covered in depth by Eamon Murphy, Steve Hayes, and Matthias Enders, in *TCP/IP Tutorial and Technical Overview*, Fifth Edition, Prentice Hall PTR, 1995. See also the Related Publications section for further TCP/IP study.)

Web servers use HTTP to communicate with Web browsers. HTTP is an application-level protocol for distributed, collaborative, hypermedia information systems [RFC 2068, Fielding, et al., p. 7]. Sometimes you will hear Web servers referred to as *httpd,* which stands for *Hypertext Transfer Protocol Daemon*. httpd is a type of information server or small program that uses HTTP protocol. Daemons (a UNIX term) provide background server functions such as name server functions *(named*) or routing *(routed*). The world of the Web reflects UNIX terms and syntax because the Internet—and the Web—were created by people who worked in a UNIX environment. This explains why the syntax of the Internet and the WWW differ from common PC syntax (or Mac for that matter). You can run an HTTP daemon on any multi-tasking operating system (OS) with a TCP/IP stack; however, it works better if you have a multi-tasking system like MacOS Version 7.1 or later, and best on a pre-emptive multi-tasking, multi-threading OS like OS/2 or Windows NT in this environment. We are excluding multi-user operating systems from this discussion.

In HTTP Version 1.0, each browser connection provides a self-contained session for each transfer of information. The new version, HTTP 1.1, a Proposed Standard described in RFC (*Request For Comment)* 2068, permits *persistent connections.* These work much like phone calls, except that the 'Net uses *domain name resolution* to translate domain names like www.ibm.com to IP addresses like 204.146.46.133 while the phone company uses directory assistance—if you know the name and not the phone number (ah...progress). Using HTTP 1.0, you would have to make a new call for each request; HTTP 1.1 permits multiple requests. Either way, each time you access a Web document, four basic processes happen:

ANALOGY

- ✓ YOU CALL DIRECTORY ASSISTANCE **(DOMAIN NAME SERVER OR DNS).**
- ✓ THEY GIVE YOU THE NUMBER **(IP ADDRESS).**
- ✓ YOU PRESS ONE TO PAY EXTRA AND DIAL THE CALL. **(PACKETS GO)**.
- ✓ THE PHONE IS ANSWERED **(SERVER LISTENS).**
- ✓ YOU ASK FOR WHOMEVER YOU CALLED TO SPEAK TO **(REQUEST CONNECTION).**
- ✓ THE PERSON YOU CALLED COMES TO THE PHONE AND SAYS "HELLO" **(SERVER ACCEPTS).**

1. Connect

owser uses the HTTP scheme to locate network resources via HTTP. It requests a URL (Uniform Resource Locator) in the following format:

protocol://hostname.com:portnumber

for example
http://info.cern.ch:8080

The server typically listens for connection requests on port 80 the "well-known port" assigned to HTTP requests. (Other protocols have their own default well-known ports.) In the example above, I used a typical port number for Web server testing. (You configure this, as mentioned above, in the main configuration file.)

Analogy
- ✓ You ask the person you called for whatever information you need **(Request URL)**.

HTTP 1.0
- ✓ You call back for each question (A major reason for World Wide Wait) **(New connection for text and each graphic element)**.

HTTP 1.1
- ✓ You ask questions until you have what you need **(Persistent connection)**

2. Request

The request specifies the actual resource, e.g., the above example would return the default home page of the test database. The following example shows a URL with the path and Web page explicitly identified. The request will also contain a request method, in this case, GET

METHOD GET http://info.cern.ch/WWW/HTML.html HTTP/1.0

GET is the most common request method. Other methods include: HEAD, POST, PUT, and DELETE; the last three require more than read access to the files and data on the server; however, POST may be used in CGI (Common Gateway Interface) scripts (discussed in this document under Scripts). The last part of the GET request specifies which version of HTTP you are using; this example uses 1.0.

Analogy
- ✓ You get your information.

3. Respond

The server interprets the request and sends the response to the browser. Preceding the data is a MIME (*Multipurpose Internet Mail Extension)* header that tells the browser how to interpret the response (that is, what's in it and if the response includes something beyond HTML (*Hypertext Markup Language)* like images, sound, or video). The browser parses the request, and either (a) issues another connection request (HTTP 1.0) or query (HTTP 1.1) or (b) renders the response and displays it.

Analogy
- ✓ You hang up.

4. Close

After pumping the response down the socket, the server closes the connection...usually. Either side of this connection can shut it down, like when you push the STOP button on your browser to stop loading a Web page that someone used uncompressed graphics on ….

HTTP 1.0 remains the most common version of this protocol in use since the vast majority of installed browsers are Netscape Navigator and Microsoft Internet Explorer which still support HTTP 1.0. The latest versions of these browsers support HTTP 1.0 plus extensions; these include an extension to support persistent connections. Full HTTP 1.1 support has not been achieved at this time. Persistent connections eliminate a significant amount of network traffic generated by establishing connections for each request. In addition, persistent connections conserve CPU activity. Speaking of CPU, we next explore the server from a hardware and software performance and configuration standpoint ….

Configuration and Performance

As the Internet handles increasingly heavy traffic, performance becomes a primary concern for any Web site. Users do not want to wait needlessly for a page to load. While the topology of the Internet's backbone are beyond the control of most webmasters, they can still tune their servers in several ways to improve performance. Web server performance is usually measured in three variables:

Connections per second
Bytes per second
Round trip time

Comparative papers exist to discuss the details of individual manufacturer's Web server's performance in specific figures, so we will not. (Use any Web search engine and search on *web servers compare* or *web servers performance*. Or look up the manufacturer's web server web site.) We will look at a few of the most effective methods of tuning a Web server. These general concepts will help to improve the performance of virtually any Web server.

Four major methods of improving performance are as follows:

Upgrading hardware
Caching
Using multiple/networked drives
Clustering servers

These concepts are all interrelated and interdependent. For example, a hardware upgrade may be necessary to take full advantage of caching, as we will see.

How can hardware upgrades affect performance?

Hardware upgrades offer the easiest method for improving web server performance. Just as you would optimize any computer for maximum performance, all aspects of a server should be optimized in order to handle a heavy load of client requests. Any of the following hardware elements could create a bottleneck in your web server environment:

Processor speed
Memory (RAM)
Storage access (Disk) speed
Network (LAN) speed

Upgrading your hardware can mean great improvement for relatively little investment. Adding RAM, for example, costs very little, but may result in superior performance. And having sufficient memory will permit you to employ another performance enhancing technique: caching.

How does caching help?

Caching (loading files or directories into memory) allows the Web server to respond more quickly to browser requests. Essentially, the Web server retains these cached files, and the ones most frequently requested, in memory (RAM). When a user requests a cached page or file, the web server can serve the file directly from memory, rather than opening, reading and closing the file. While disk access times are measured in milliseconds, RAM access time is measured in nanoseconds—noticably faster. High-load servers, in particular, benefit from this performance boost. Caching provides another benefit: caching curtails disk access, and thus lengthens disk life.

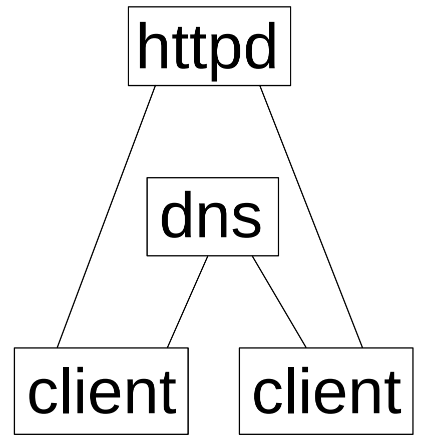
In order to enable caching, the Webmaster selects individual files or whole directories to place into the system's memory. For example you might choose to cache the site map. When the server receives a request for this page, it can send it directly from memory, rather than access the storage media, which accelerates the response time. One thing to keep in mind when caching, is that you need ample memory for the server's other requirements, including the operating system and the Web server itself.

Caching frequently accessed files can improve web server performance in any web site model. All of the following site configurations will benefit from caching.

How does Clustering affect performance?

Single server model

In the simplest model, a web site URL corresponds to one physical server. For example, Somecompany, Inc.'s site, www.someco.net, would have a DNS entry



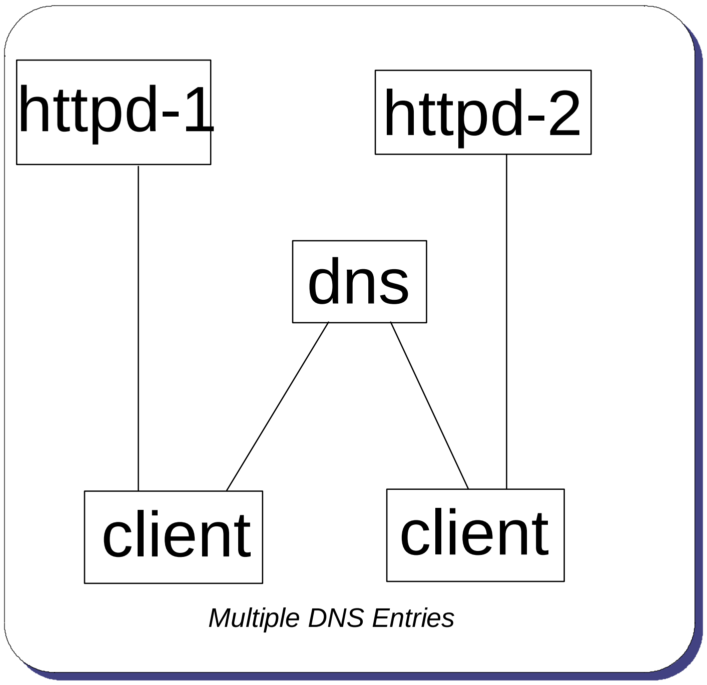
Single Server model

for a single IP address that corresponds to the network card on the web server serving someco.net's site.

Clearly with a site as popular and important as Somecompany, Inc.'s, you would not want to rely on a single machine to handle the load of millions of requests. Moreover, what would happen if that machine went down? The site would go down as well. There are several different methods to protect against such failure; we examine a couple of possible solutions below.

The above diagram shows the single server model. Each client connects with the DNS via TCP/IP to identify the web server's IP address. Each client then accesses the Web server.

Multiple DNS entries



Multiple DNS Entries

Multiple DNS entries provide one solution to this scenario. Rather than one server, one address scheme, the DNS will have multiple IP addresses for a given name.

for example:

Name: www.someco.net

Addresses: 9.19.139.4, 9.19.139.5, 9.19.139.6, 9.19.139.12

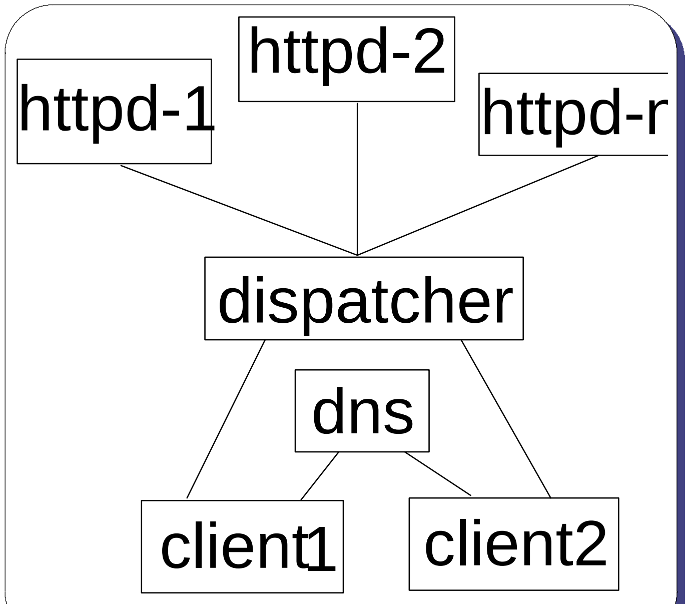
When the browser requests the URL www.someco.net, the DNS returns one of the listed numerical addresses, in round-robin order. This simple solution may effectively balance the load in the long run. If the request load continues to increase, you can expand this solution by adding another IP address and another machine.

In the above diagram, each client requests the IP address of the same Web server from the DNS. The DNS returns a different answer from the list to the second requesting client. Thus, the DNS distributes the load among multiple Web servers. This "blind" form of load distribution does not take into account the load levels of the servers. The DNS merely rotates through a list of IP addresses that correspond to a given hostname.

Network Dispatching

Server clustering with a network dispatcher provides the best solution to this scenario. A network dispatcher assumes the distribution role played by the DNS in the multiple DNS entries solution. The dispatcher assigns client requests, and provides some additional functions.

With dispatching, one server acts as the “front door” for the organization. This server routes browsers’ requests to any number of Web servers. To the browser, again, there seems to be only one address corresponding to one server. In reality, the dispatching server balances the server load among several servers “behind” it. Network dispatching improves upon the simple use of multiple DNS entries in a number of ways.



Network Dispatching

Most significantly, the network dispatcher actively balances the server load. The dispatcher remains aware of the load of any given server at all times; DNS-based load balancing relies on chance. Thus, a network dispatcher allows more effective load balancing.

Another feature improves security: a dispatcher screens the topography of the network from end-users. To the browsers, there is only one address through which they may access any number of services. Screening the internal topography significantly reduces opportunities for malicious external attack on your intranet.

A dispatcher also makes site maintenance easier. As daily hits to the site increase, you can add additional servers to reduce the load on any one server. Similarly, if a web server fails, you can remove or replace it without bringing the whole site down.

What other factors can affect performance?

Web servers can take advantage of networked drives and network file systems. This allows multiple servers to access one identical set of resources, which is particularly useful in implementing either the round-robin DNS or network dispatching solutions. Essentially, each web server mounts the shared resources as a virtual disk, eliminating the need to replicate identical data out to each Web server. File systems that support remote mounting in the PC environment include AppleShare, and Netware.

Forms & Scripts, Gateways, and Java

This is where the plot thickens...as anyone who spends time surfing the Web knows, a lot more goes on than simply requesting static HTML (*Hypertext Markup Language)* and graphics. No doubt: the trend for greater interactivity will continue to sweep the Web. Forms were the first vehicle used to elicit information so the server could provide custom responses based on that input, so we'll begin there.

What are Forms & Scripts?

Forms are HTML documents with places for users to input information. Forms permit an interactive exchange between the server and the browser. You create the form within an HTML document using HTML tags. The form is actually the part of the HTML document between the opening tag

<FORM ACTION="http://www.somecompany.com/cgi-bin/location.cgi" METHOD=GET>

and the closing tag **</FORM>.**

The body of the form resides within the opening and closing parameters of the form tags. Within the body of the form, other HTML tags define the kind of fields—INPUT, SELECT, or TEXTAREA. These tags all require a symbolic NAME defined for the data field; INPUT requires the type of data (text, password, checkbox, radio/on-off toggle, submit, or reset), the default value, and the physical size of the data field displayed. A maximum length parameter can also be defined for text or password entry fields. SELECT displays a menu from which to choose a response. TEXTAREA provides a larger field for lengthier input. (A more detailed tutorial on HTML and how to create forms will not be included here; however, this topic is well documented elsewhere. See the Related Publications section for suggestions for HTML instruction.)

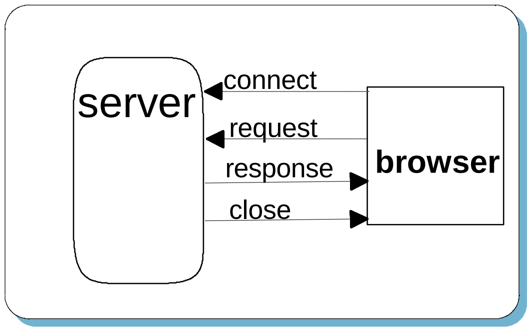
The **METHOD=XXX** part of the opening tag defines the way the form returns the responses to the server. GET appends the data to the end of the URL, with the parameters following a question mark. POST sends the data as a separate MIME-encapsulated transaction. (*MIME* stands for *Multipurpose Internet Mail Extension.*) POST is the preferred method if any significant amount of data will be returned.

When the user fills out and submits this Form, *Web scripts—a*lso known as *CGI scripts—c*ome into play. CGI, *Common Gateway Interface*, is another suite of specifications. From the browser's perspective these specifications allow any browser on any platform to send data to any server on any platform that supports CGI. The differences appear on the server side of the interaction: the specifications for the gateway interface depend on the server's operating system.

CGI scripts run on the Web server; they receive information, execute, and return (via the HTTP server) the requested output to the browser that initiated the contact —if all goes well. These scripts usually reside in their own separate directory or group of directories for security purposes, specifically to limit access to the executable programs. Many servers use the name \cgi-bin for this directory.

CGI "script" is a bit of a misnomer; you can write CGI scripts with a wide variety of languages, both compiled, e.g., C or C++, or interpreted scripting languages, e.g., Perl or REXX. Compiled languages are better for some tasks; interpreted languages are better for others. Some say Perl is easier to write and modify; however, interpreted language scripts use more memory and run more slowly than compiled programs. On the other hand, in his book The Web Architect's Handbook, Charles Stross [p. 97] illustrates this issue with the example of a UNIX filter to count the number of words in a text file—this takes four lines of Perl compared to over a hundred lines of C. Right now on the Web, a large proportion of CGI scripts have been written in Perl (*Practical Extraction and Reporting Language* or, attributed to the language's inventor, Larry Wall, *Pathologically Eclectic Rubbish Lister*...really!) [Stross, p.102].

Here is one IBM developer's perspective on the script versus compiled debate: "The dominant cost of the script execution is having to create a new process for it and later terminate it. The speed of the script itself is therefore negligible and should not be considered. You are better off with [P]erl scripts because of ease of implementation" [Bulka]. Based on implementation, the majority of programmers writing scripts appear to agree.



How does this process differ from serving static HTML?

Obviously, serving forms and scripts require the server to work harder and to use more resources than when it serves static HTML. The more complex the task required by the script, the greater the load on the server. Initially, the process follows the same steps as when the server serves static HTML:

The server listens
The server receives a request
The server performs the requested method
The server closes the connection

Now the browser has the form. The server does not await the form; the server has finished its part of this task.

This brings us to the point where things change, however. When the user fills in the form and presses the submit button, a new series of activities takes place. The browser builds this request using the **ACTION** specified in the form and the input from the fields.

The resulting request includes:

> The URL for the script in the /cgi-bin or executable directory.
> The name-value pairs separated by ampersands (&) to identify the symbolic names of the fields in the form.
> The values of the responses.

The request could look something like this (except in a one-line string):

**GET http://www.someco.net:80/cgi-bin/phonebook?nserver=someco.db.com&
name=yes&phone=yes&email=trishs@us.ibm.com**

The question mark identifies the beginning of the form options which the server passes to the script as the QUERY_STRING environment variable (this conforms with a CGI specification). Then the script

> Converts the QUERY_STRING into a useable format for the program.
> Divides up the name-value pairs.
> Executes the program....could be
> - ✓ Retrieve the time and date or
> - ✓ Access a database or
> - ✓ Calculate complex computations using the variables provided or
> - ✓ "Observe" something like the fishcam web site.
>   (http://www.netscape.com/fishcam/fish_refresh.html)
>
> Returns the result to the server.

Then the Server

> Composes an HTTP header for the response to the browser.
> Formats the output in the response.
> Sends the result.
> Closes the connection.

Let's use our example above. We'll say this form requests phone numbers based on the e-mail address of the person. The process would look like this:

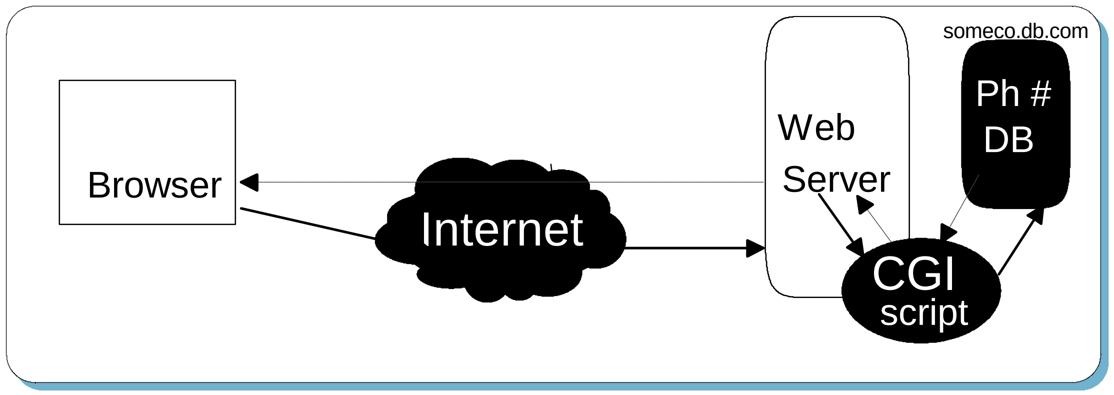
In this example, the script acts as a gateway to the data in the database. This brings us to our next topic: gateways.

What are gateways and how do they work?

Gateways provide Web browsers access to information not natively available to web browsers—often legacy systems (e.g., flat-file mainframe), relational databases (e.g., IBM's DB2), or transaction applications (e.g., CICS based). Mostly, these gateways work very much the way the example appears above, built upon the CGI standard and the three-tier model of browser/webserver/legacy system. The gateway acts as an interpreter:

translating the request from the browser for information into a protocol that the legacy system understands
translating the protocol used by the legacy system in its response to HTTP
formatting the response into browser-friendly HTML

Another possible gateway implementation uses *server-side includes* (SSIs). This is far less common than CGI. Server-side includes are macros or execs that run on the server when the server responds to a request for an HTML document that contains these executables. This request could mean simply incorporating date and time information or may execute a program whose output is included in the page requested. SSI has some potential drawbacks. First, the server must parse every page it serves



looking for includes in the HTML whether the page requested contains includes or not which adds considerable overhead on the server and diminishes performance. Second, many find the potential security risks outweigh the benefits. It is important to be aware of the exposure inherent in letting an external entity initiate any executable on your server.

The third, and up-and-coming possibility for a gateway implementation, is using Java or Java Beans.

So what about Java?

Developed by Sun Microsystems, Sun describes Java thus in The Java Language: An Overview available at **http://java.sun.com/docs/Overviews/java/java-overview-1.html:**

> Java: A simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language.

Java is also described (more succinctly) as a programming language platform. We'll examine how this relates to the world of the Web.

Java was designed to be easy for programmers to use—it's based on C++; to work in a networked environment; and to be architechture-independent. Programmers can use Java language to create several different types of programs:

Applications - standalone programs
Applets - tiny Java programs embedded in webpages that run on a Java-enabled browser when the page is served
Servlets - programs that run on the Webserver

Java has also spawned ***Java Beans***, the application development model for assembling Java using components. Java Beans are intended to be reusable components or objects in an object-oriented development environment. Simple Java Beans, such as pushbuttons, text fields, and check boxes, can be incorporated into more complex Java Beans, such as a calculator. You can use both simple and complex Java Beans as components in an application or applet (to continue our example, incorporating the text field Bean and push button Bean and calculator Bean in a GUI (*graphical user interface*) of a cash register program). Java Beans can be visual in nature, such as a list box, or non-visual, such as a schema map of a DB2 database table. Java Beans' reusability and portability make Java developers more productive since they have at their disposal the standard object components, they can create and add new objects when necessary, and they do not have to start from scratch with each new development project.

Java applications manage architectural independence through running in a Java-enabled environment, the Java Virtual Machine (*JVM*). An example of this is the HotJava browser. In any kind of Java language program, Java code is both compiled and interpreted. Rather than producing machine code like C or C++, the Java compiler generates bytecode which is interpreted by the JVM. This provides the "Write Once, Run Anywhere™" capability. Java can also convert this bytecode into machine code at runtime, if an applet or application requires the fastest possible implementation. Java uses a JIT (*Just In Time*) compiler to enable this conversion, which is just as fast as C or C++ programs. Thus, Java finds the happy medium between the high-level, portable but slow scripting languages and low-level, fast, and sometimes labor-intensive to debug machine-code produced by compiled languages.

Java applets were created for use on the Web, providing most of the animated images on the Web today. When a Java-enabled browser encounters the tags below in an HTML page it knows to download and run the applet.

```
 <applet code=GeneralAnimate.class width=125 height=108>
<! - - The animTime parameter contains the number of milliseconds to wait - ->
<! - - between displays of images.                                          - ->
<param name=animTime value=500>
<! - - The following three parameters contain the RGB values of the color - ->
<! - - of the background of the window the animation runs in.  The double  - ->
<! - - quotes are needed so that the value of 0 will be recognized.        - ->
<param name=rColorVal value=”0”>
<param name=gColorVal value=”0”>
<param name=bColorVal value=”0”>
<! - - The following parameters contain the number of images shown in the - ->
<! - - animation and the files containing the images.  The image parameters- ->
<! - - must be listed in the format “pieceX”, where X represents the order  - ->
<! - - the image is shown.                                                  - ->
<param name=numPieces value=12>
<param name=piece1 value=”F01.jpg”>
<param name=piece2 value=”F02.jpg”>
<param name=piece3 value=”F03.jpg”>
<param name=piece4 value=”F04.jpg”>
<param name=piece5 value=”F05.jpg”>
<param name=piece6 value=”F06.jpg”>
<param name=piece7 value=”F07.jpg”>
<param name=piece8 value=”F08.jpg”>
<param name=piece9 value=”F09.jpg”>
<param name=piece10 value=”F10.jpg”>
<param name=piece11 value=”F11.jpg”>
<param name=piece12 value=”F12.jpg”>
</applet>
```

(*applet courtesy of Kelly Westphal, IBM Advanced Technical Support, Personal Solutions Systems Center, Application Development Team)*

The Java-enabled browser invokes subroutines called *methods* which tell the applet to initialize itself, draw the space in which it will display, and do whatever the applet is required to do. This technology promises to be the *de facto* standard in the Internet world soon.

Web Server Security

Placing a Web server on the Internet opens a door into your organization, inviting the public in. This includes those interested in your organization, your products, or your services; but unfortunately, this could also include undesirables interested in snooping and hacking their way into your private files and your secure network. Fortunately, you can take reasonable precautions to secure your Web server.

Webster defines security, “as relating to protection, as measures taken to guard against espionage or sabotage, crime, attack, or escape.” This definition corresponds easily to the context of Web servers. The measures available to protect against these dangers constitute Web server security. The degree of security required depends on the location and purpose of the Web server; whether the server exists outside a company firewall directly connected to the Internet, or is connected to the company’s secure intranet.

So what’s involved in Web Server Security?

Security has two major components: access control, and confidentiality or privacy.

Access Control ensures that only expressly permitted users can access the data. ***Confidentiality/Privacy*** ensures the information transferred is not visible or usable to anyone other than the intended recipient. Several other concepts fall under this heading:

- ✓ *Authentication* ensures the identity of the parties involved; simply, does the user ID and password match.
- ✓ *Authorization* ensures that given the user ID and password match, is this user permitted to access this resource.
- ✓ *Integrity* ensures the data sent equals the data received.
- ✓ *Accountability* ensures the data exchange or transaction has taken place, also called *non-repudiation*.

Access Control

This concept covers a lot of ground—from the of the server machine’s physical safety to the directory permissions controlling file access. The Web server software controls who can connect to the server and what the user can retrieve, alter, or delete through *access controls.* These are specific mapping statement *directives* that modify the server configuration file. Through these access controls, you can

- Deny access to the documents not intended for users.

- Grant access to documents for selected users by requiring user ids and passwords.
- Restrict access to selected IP addresses or domain names.
- Allow “read only” access.
- Enforce any combination of the above controls.

Applying the “*that which is not expressly permitted is prohibited*” philosophy to the server access controls will assure the tightest security. Give access only to specific documents by selected, authorized users to fulfill the mission of the server and deny all others access.

Directives are based on *matching requests.* When you request a URL, the server checks to see what instructions exist that relate to that particular URL by looking for that URL template defined in the configuration file.

The directives in a mapping statement can have any of the following values:

- Pass - Accept matching URL requests/ permit access to this resource.
- Fail - Reject matching URL requests/ do not permit access to this resource.
- Map - Change to a new string (directory, file name or both) if the URL matches.
- Exec - Run a CGI program if the URL request matches.
- Redirect - Send to another server if the URL request matches.

These directives allow you to create a virtual hierarchy of the server resources by redirecting or passing client requests to other systems. This virtual layout enhances the server security by locating sensitive files and directories on different drives and systems, yet is transparent to the user.

Confidentiality/Privacy

Once you set up server security for the directories, you can allow individual users entry into areas that are off limits to other users. By creating files containing lists of users and their passwords the protection directives define access to the server resources. These password files can then be used by Access Control Lists (or ACL files) and by the protection setups. Access can be limited to specific user(s), IP address(es) or domain name(s) at the directory or file level.

Authentication

Authentication, in the context of Web server security, really refers to two different concepts. First, *password authentication* refers to the scenario common in multi-user environments (like in Lotus Notes) in which the user must log in with a password to access the system. Second, *basic authentication*, a part of the HTTP protocol standard, “is a non-secure method of filtering unauthorized access to resources on an HTTP server. It is based on the assumption that the connection between the client and the server can be regarded as a trusted carrier. As this is not generally true on an open network, the basic authentication scheme should be used accordingly" [RFC-1945]. Basic authentication does not define permissions, it is simply the mechanism for requiring and providing credentials (user ID/password) much like that in password authentication; however, the encoding of the password is trivial, so it is trivial to steal. Serious security requires SSL (which we discuss later in this section).

You can define *authorizations* in the Web server configuration file to establish access control specifications which limit access to files in a directory by the request method (e.g. GET, POST, PUT) and identifies specific users with access to the files.

When the browser requests a restricted resource, the HTTP daemon looks for *authorization* in the header information that initiates the session. If the server does not find the authorization field, rather than connecting the browser to the resource, the server will send a status code 401 response—Unauthorized—to the browser. The browser will then ask the user for a password and resend the request with the password and the user's name (encoded) in the authorization field right under the method, URL, and HTTP version information.

```
GET http://www.somecompany.com/private/secret.html HTTP/1.1
Authorization:  Basic 0(766^*>:9=84%7&54#"@
```

If the access control list for this resource includes your name and password, and your request employs an acceptable method, you will receive the usual response and get access to the Web page. If not, you will receive the default error 403, Forbidden, or the custom error message for this server, if set up. (See RFC-1945 for further reading regarding error codes.)

Your need for security extends beyond basic authentication and access to data on the server. Next we look beyond the server itself to securing the data enroute from server to browser.

How can you secure the data while it's enroute between the server and the browser?

Let's say that your Web server setup provides for maximum available security. Since the server is secure, then your data is secure. Right? ***Wrong***. TCP/IP provides a transport mechanism to ensure your data gets from your server to the browser that requested it. Data security needs to be in place above the transport layer *before* you send it across the Internet.

Without data security, all transmissions on the Internet are open for viewing, interception, or alteration. Data security attempts to:

- Make the information unreadable/unusable by anyone other than the intended recipient: **confidentiality/privacy** and **authenticity**.

- Alert the parties involved that transmissions have been tampered with or changed: **integrity**.
- Prove the transmissions or transactions took place and were received: **accountability**.

Two protocols were developed to secure data: SSL (*Secure Sockets Layer),* developed by Netscape Communications, and, S-HTTP *(Secure HTTP).* Most Web servers and browers support one or both of these protocols. SSL is the by far the most widely implemented.

How does SSL work?

The Secure Sockets Layer Protocol, or SSL, has two main objectives:

To ensure confidentiality and privacy by encrypting the data that travels between a browser and a Web server.
To ensure authentication of the Web server and the browser.

SSL encapsulates HTTP as it travels down the TCP/IP stack. The SSL protocol resides between the application layer (HTTP) and the transport layer (TCP). SSL connects to port 443 of the server, rather than to the HTTP standard port 80. SSL uses several encryption techniques to provide confidentiality/privacy and authenticity. It also uses digital signatures to ensure integrity and accountability. First we'll go through the SSL process, then we'll examine encryption..

The secure session process differs from a regular server/browser exchange. All the above information passes back and forth to establish the secure socket and enable encryption well before any HTTP messages are transmitted. This process is transparent to the user who initiates the request.

The parameters set up during the handshake are kept in the CipherSpec file. The requested URL can now be sent using the negotiated symmetric-key bulk cipher, with a new set of session keys calculated for each new session.

Clearly, starting an SSL session generates considerable overhead compared to HTTP. The protocol escapes some of this overhead when resuming sessions already negotiated.

Now you can begin what you started out to do: securely transfer data.

Once a SSL has been implemented, graphical signals on your browser screen let you know you are in a secure session. If, for example, you use Netscape Navigator, look at the lower left corner and you will see a key icon. If the key is broken, you are not in a secure session. If the key is whole and only has one tooth on it, you are in a secure session using 40-bit encryption (for export). If the key is whole and has two teeth on it, you are in a secure session with 128-bit encryption (US and Canada only). 128 bit encryption is, in general, illegal to export outside of the US and Canada; however, several US companies are working on strategies to work this out and exceptions, such as the banking industry, already exist. This ventures into the realm of laws and lawyers and as such we'll leave it there.

Hello? (the SSL session begins...)

Browser

Initiates a secure session using a URL starting with https:// (instead of the usual http://) which asks for a session with the server on port 443.

Provides its SSL version number, a list of supported encryption options, and a random number.

Server

Responds with the corresponding SSL version and encryption option information to allow the browser and server to find common ground for communicating.

Server sends the client its identifying *certificate* (see certificates section, following) containing its public key, a random number and, if client authentication is required, requests a client certificate.

Browser

Generates the session key using its own random number and the server's random number.

Encrypts the session key with the servers public key.

Sends the encrypted session key to the server, as well as its certificate (if requested).

Server

Decrypts the session key with its private key.

Sends a message back to the browser encrypted with the session key (authenticating itself) known as the message digest or Message Authentication Code (MAC).

So how does encryption work?

Encryption provides privacy and confidentiality by scrambling data in a manner that makes it unreadable or unusable until the intended recipient unscrambles it. The sender uses an algorithm pattern or *key* to scramble the data. The recipient decrypts it with the corresponding algorithm or key. The level of security that encryption gives directly relates to the length of the encryption key. The larger the key, the longer it will take someone to break the key.

There are two common encryption keys: symmetric and asymmetric (or public) key.

- Symmetric-Key Encryption (symmetric): The sender and receiver have copies of the same, shared, secret key.

- Public-Key Encryption (asymmetric): This method has a public and private *key pair*. The private key is known only to the owner, while the public key is available to anyone. Since the keys are different, only the owner of the private key can decrypt any data scrambled by the public key, which can also authenticate a client.

Most security protocols use both public-key and symmetric-key encryption. The public-key provides authenticity and privacy, but is slow in encrypting large amounts of data. The symmetric-key is faster in encrypting the data, but lacks the security of public-key in using a secure encryption key. Usually the public key encryption method is used to exchange a symmetric key for the session key.

Digital Signatures use public key encryption to authenticate the sender and validate the integrity of the data, incorporating the entire message as well as the key in the computation. This method of signing a message can be costly in both time and resources if the message is a very long one. Another way of signing a message that is quicker is the *one-way hash function* or *message digest function*. See the IBM Redbook, *Safe Surfing: How to Build a Secure WWW Connection* (SG24-4564-00) for more detail on digital signatures and hash functions.

What about Certificates?

Certificates, also known as digital IDs and digital certificates, confirm the identity of the sender's public key. This certificate has the highest credibility if it is issued by an independent third party Certificate Authority (CA). Businesses comply with a rigorous application process and pay for this certificate. Individuals have a less rigorous, but still precise, process to obtain a digital ID. Certificate authorities are responsible for validating that the certificate holders are, in fact, who they claim to be. Within a company's own intranet, it can sometimes be more efficient to have the server become its own certificate authority—for a specific project or situation that requires high security and limited access within that already secure environment. For any e-commerce activity out on the Internet, the validating authority should be one of the standard CAs recognized by your browser, beyond this ...*caveat emptor* — let the buyer beware.

Summary

This paper covers Web server basics: an overview of how Web server technology works; possible ways to improve performance; and how multiple servers can be used to improve site performance. We looked at the elements involved in interactive Web communications between browsers and any other kind of computer through scripts, executables, gateways to legacy systems, and Java. We looked into how to protect all this at the server. This is only the beginning—in many ways. The Internet, and specifically the World Wide Web, is changing the world.   How we do business, how we share information, how we learn, and more, have changed, and will continue to do so. The more we can promote understanding of how this technology works, the smoother this transition can be. Peter Andrews, another IBMer, said this well in his "The Grease Monkey Theory of the Web" *"...the more you understand a medium, the more effectively you can use it for communication*". This Web Server Primer is a beginning.

Acknowledgments

The authors would like to thank the other people who have contributed to this paper. Many people within the Personal Solutions Systems Center read this paper as a work in progress and offered helpful perspectives. In particular, Wendell Crosley and Van Landrum, our teammates, Joe Arnold, from the DB2 team, and Kelly Westphal from the AD team took the time to make suggestions in writing and/or contribute to this work. Donna Su, from the Publications team, contributed greatly to the readability of this paper through her recommendations (and is the only person who has read this paper as many times as I have…T.S.). We would especially like to thank Dov Bulka and Richard Gray from the IBM Web Server Development Lab in Raleigh for their insight and recommendations.

Additional Information and Sources

Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

Murphy, Eamon, Steve Hayes, and Matthias Enders, *TCP/IP Tutorial and Technical Overview*, Fifth Edition, Prentice Hall PTR, 1995.

Stross, Charles, *The Web Architect's Handbook*, Addison Wesley, 1996.

Yeager, Nancy and Robert E. McGrath, *Web Server Technology*, Morgan Kaufmann Publishers, Inc., 1996.

IBM Redbooks:

Building a Firewall with the IBM Internet Connections Secured Network Gateway
Document Number SG24-2577-01.

Safe Surfing: How to Build a Secure WWW Connection
Document Number SG24-4564-00.

A Guide to the Internet Connection Servers
Document Number SG24-4805-00.

Building The Infrastructure for the Internet
Document Number SG24-4824-00.

Accessing the Internet
Document Number SG24-2597-00.

Using the Information Super Highway
Document Number GG24-2499-00.

Internet Request for Comments via internic.net:

[RFC-822] revised by Crocker, D. H.,“Standard for the Format of ARPA Internet Text Messages,” RFC 822, August 1982.
[RFC-1009] Braden, R. and J. Postel, “Requirements for Internet Gateways,” RFC 1009, June 1987.
[RFC-1118] Krol , B., “The Hitchhikers Guide to the Internet,” RFC 1118, September 1989.
[RFC-1386] Cooper, A., and J. Postel, “The US Domain,” RFC 1386, December 1992.
[RFC-1739] Kessler, G., and S. Shepard, “A Primer On Internet and TCP/IP Tools,” RFC 1739, December 1994.
[RFC-1825] Atkinson, R. Security Architecture for the Internet Protocol,” RFC 1825, August 1995.
[RFC-1828] Metzger Pierpont , P., and W. Simpson Daydreamer “IP Authentication using Keyed MD5,” RFC 1828, August 1995.
[RFC-1918] Rekhter, Y., B. Moskowitz, D. Karrenberg, G.J. de Groot, and E. Lear, “Address Allocation for Private Internets,” RFC 1918, February 1996.
[RFC-1945] Berners-Lee, T., R. Fielding, and H. Frystyk, “Hypertext Transfer Protocol,” RFC 1945, May 1996.
[RFC-2068] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, “ Hypertext Transfer Protocol—HTTP/1.1,” January 1997.
[RFC-2069] Franks, J., P. Hallam-Baker, J. Hostetler,P. Leach, A. Luotonen, E. Sink, and L. Stewart, “An Extension to HTTP: Digest Access Authentication,” RFC 2069, January 1997.
[RFC-2076] Palme, J. “Common Internet Message Headers,” RFC 2076, February 1997.

Other Sources

Bulka, Dov, Private Communication.

Web Sites

IBM maintains extensive and timely information on the World Wide Web. Visis the following sites for more information on IBM PC Servers and other IBM products. These sources contain product information, performance data, and technical literature.

IBM Home Page
http://www.ibm.com

IBM PC Company Home page
http://www.pc.ibm.com

IBM PC Server Home page
http://www.pc.ibm.com/server

IBM PC Company Support
http://www.pc.ibm.com/support.html

Lotus Domino Go Webserver
http://www.ics.raleigh.ibm.com/dominogowebserver

TechConnect Program
http://www.pc.ibm.com/techlink

IBM PSSC Home page
http://pssc.dfw.ibm.com

File repositories
http://www.pc.ibm.com/files.html
ftp://ftp.pcco.ibm.com

Other Helpful Web Sites:

CGI
http://hoohoo.ncsa.uiuc.edu/cgi/intro.html

Dictionary of the Internet Language
http://www.netlingo.com

Directory of Internet Information Sources
http://www.december.com/cmc/info

Glossary of Internet Terms
http://www.nw-direct.com/web_dict.htm

HTML
http://www.sandia.gov/sci_compute/elements.html
http://www.cc.ukans.edu/info/HTML_quick.html

Java
http://java.sun.com/docs/Overviews/java/java-overview-1.html

RFCs
http://www.internic.net/ds/dspg1intdoc.html

TCP/IP
http://www.datacomm-us.com/technow/scan05/scan05.html

The Webmaster's Handbook
http://www.infomatik.th-darmstadt.de/%7Eneuss/Handbook/sample/sample.html

WWW Servers in the World (all of the)

http://www.vtourist.com

Special Notices