Comparison of the HOB WebSecureProxy and the Apache Web Server (httpd / HTTP Daemon)

1. The Apache Web Server

The Apache Web Server was originally created in 1995. It was based on and derived from the earlier NCSA server, written by the National Center for Supercomputing Applications (which also developed the Mosaic browser, predecessor to most of today's browsers, with a direct line to Netscape and Mozilla, and considerable influence over others, including MSIE). The first production server under the Apache name was version 1.0.0, released in December 1995.

The Apache Web Server is developed in the C programming language.

The Apache Web Server (httpd) consists of different modules which are loaded at start-time. These modules are Dlls (Dynamic-link library) or shared objects (.so). Since httpd version 2.0, most of these modules need certain load points defined thru the precompiler macro AP_MODULE_DECLARE_DATA.

The original World-Wide-Web worked as follows:
The client = browser creates a TCP connection to the Web server and sends an HTTP request. In the HTTP request, the URL (Uniform Resource Locator) contains the path and file-name on the disk of the Web server. The Web server just reads the file from disk, makes an HTTP header before the content, and sends all this back to the client = browser. In this form, HTTP is half-duplex, meaning the client sends a request and the server sends the response. The Apache Web server, from the ground up, is made for these short-living half-duplex requests.

The memory management of the Apache Web server has pools for the memory acquired. Components of the Apache Web Server, modules, get memory from one of the pools, but there is no explicit "free" command to get rid of this memory again. Freeing of memory in the Apache Web Server works the way, that at certain points, the memory of a complete pool is freed. The pool which has the shortest live-time is the pool associated with the request, meaning the request lives from the time the server has received the HTTP header from the client till the time the server has completely sent the response to the client. During the live-time of a single request, memory is not freed.

For the Apache Web Server, there are choices for the Multi-Processing-Module (MPM). Unix systems did not have threads for a long time. Only within the last few years have threads become available in Unix systems. So the original Apache Web Server used processes (forked) for processing parallel HTTP requests. For the lifetime of an HTTP request, the same process was

used. This was common praxis in Unix, and the Apache MPM prefork works this way. As fork() never existed on Windows, the Apache Web Server used threads in Windows.

Newer Apache MPMs also support threads on Unix.

When the Apache Web Server / HTTP processes long-running Web transactions, for each of the long-running Web Transactions = connections, a process or thread stays in use. This means, that the operating system (Unix / Linux / Windows) needs to handle a large number of processes or threads, giving quite some overhead.

When multiple processes are used by the Apache Web Server (httpd), there needs to be communication between these processes - so-called inter-process-communication (IPC). Shared memory and Mutexes are used. In the older Unix systems, shared memory was always backed-up by a disk-file, so when bytes in the shared memory were changed, there had to be slow access to the disks.

Also the kernel needs to get involved in interprocess communication, and context switches are necessary. This also results in an additional penalty for performance, which would not be needed when a single process would be used.

Overall, using multiple processes for an advanced server means quite some overhead.

For server applications, the memory subsystem is an important design point. Also, copying of the data means high CPU-usage. With Apache httpd 2.0, the concept of filters and bucket brigades were introduced. By using bucket brigades, copying of memory can be avoided in many places.

The complete Apache Web Server (httpd) version 2.4.x, with all the modules distributed by the Apache Software Foundation (ASF), consists of something between 500 and 600 thousand lines of code.

Newer requirements, which implement a Rich Client over the public Internet, require full-duplex communication. For this full-duplex communication, WebSockets were created and implemented. But for the Apache Web Server, all this full-duplex communication does not fit to the concepts of the Apache Web Server, especially not for the memory management and also process and thread handling.

When the Apache Web Server is used with SSL (HTTPS), mostly mod_ssl is used. mod_ssl includes OpenSSL.

HOB has developed mod_hob_ssl for the Apache Web Server. mod_hob_ssl includes the highly secure HOB-SSL. mod_hob_ssl can be used in Linux Apache Web Servers, starting from version 2.0. mod_hob_ssl replaces mod_ssl. mod_hob_ssl is closed-source, not open-source.

2. The HOB WebSecureProxy (WSP)

Development of the HOB WebSecureProxy (WSP) started in the year 2000. The first versions of the WSP were used for 3270 (IBM
mainframes) and RDP (Microsoft Remote Desktop Protocol for WTS = Windows Terminal Servers).

Before HOB developed the WSP, starting in 1982, HOBCOM, an IBM mainframe application, was developed by HOB. HOBCOM was made for Rich Clients, meaning full-duplex communication over SNA (IBM Systems Network Architecture). HOBCOM was already made to manage terminals and printers, also providing HOBTEXT, word processing on the IBM mainframe. HOBCOM was able to handle hundreds or even thousands of parallel connected devices.

HOB learnt techniques for handling multiple clients from the IBM transaction managers CICS (Customer Information Control System) or IMS (Information Management System). CICS and IMS are still widely in use.

So HOB uses the concept of a transaction manager in HOBCOM and also in the newer WSP (WebSecureProxy). CICS and IMS are also made for half-duplex communication, so HOB changed parts of the concept for full-duplex communication.

The WSP is mostly built around non-blocking APIs. In the WSP, there is a variable number of networking threads. These networking thread handle sockets for listen, TCP-sessions or UDP. One networking thread handles n sockets, for example 60. So, when a client has no associated server over TCP, one networking thread handles around 60 clients when n is 60. When the WSP session with the client also has a TCP connection with a server, one networking thread handles around 30 clients (when n is 60).

APIs in Unix permit these networking threads to handle more sockets, but then these networking threads could become a bottleneck (since a thread can run only on a single CPU core), and for the user, the solution would get less reactive.

When there is a networking event, or any other event in the WSP for a certain WSP session with a client, a work thread (some call it worker) is scheduled and the work thread processes what needs to get done. The work thread is left after the actual instructions, leaving the work thread waiting for other WSP sessions with clients.

As the work threads always use a CPU core, not waiting for blocking APIs, the number of work threads configured for the WSP should relate to the number of CPU cores in the server.

When all work threads are busy, and there is more work to do, the work goes to a certain backlog and is processed as soon as one of the work threads gets free.

Usage of the memory subsystem is an important design point for big applications. In the WSP, there are buffers of a certain fixed size which are used for networking buffers and also intermediate data. These buffers are called work areas. These work areas are acquired from the C library (malloc()), but not freed after usage (which is mostly for a short time only). In the WSP, these work areas are recycled. So, when work areas are needed, the caller mostly gets a recycled work area of fixed size. WSP sessions use these work areas, and when work is done (work processing a single event), a small garbage collector, in the context of a single WSP session, returns work areas which are no longer being used, back for recycling.

The WSP uses lock-free queues, something that can only be programmed in Assembler language.

The WSP uses gathers, similar to the Apache bucket brigade, to avoid freqent copying of data. When gathers are used, input data (or after SSL decryption) may be sent directly to the server without being copied.

The WSP maintains end-to-end flow-control for all types of sessions (TCP or other) thru the WSP.

The WSP always, or mostly, works with SSL connections; SSL on server side and sometimes also SSL on client side. The WSP is not made for a specific protocol like HTTP. But as a base functionality, the WSP knows the protocol going thru the SSL tunnel, like RDP, 3270, HOB-PPP-T1, SSTP or HTTP.

There is the base WSP, and, as the WSP follows the concept of a transaction program manager (TPM), there are different kinds of external components. These external components are Dlls or .so (shared object), loaded when the WSP is started. One type of these external components is called Server-Data-Hook, short SDH. A SDH follows a similar concept as the Apache filter. Another important component is the Authentication Library.

When there is an incoming TCP connection to the WSP, after the SSL handshake, the client sends something so that the used protocol (inside the SSL tunnel) is defined. This protocol may be HTTP, but it also may be any other protocol. With the WSP, when, for example, RDP for Windows Terminal Servers is tunneled thru SSL, there is not any HTTP involved. Inside the SSL there is just plain RDP, meaning no additional overhead.

To find out the protocol, and to determine what the WSP does for a certain client, mostly an extended version of the protocol Socks 5 is used. The client sends the protocol, and there is authentication and selection of the server over the WSM protocol - WebSecureProxy socks mode. No HTTP or HTML is involved.

That the client first sends the protocol is very important, since a client that uses the RDP protocol (for Windows Terminal Servers) cannot connect to an IBM mainframe, which normally uses the protocol telnet 3270.

The basic WSP supports authentication against:

- configured users
- Radius
- LDAP
- Kerberos

Authentication against SAML is in the pipeline. The Authentication Library or a Server-Data-Hook uses the authentication support built into the basic WSP. The WSP supports multitenancy, meaning the users can be> organized in groups, and each group can authenticate differently. Any number of Radius servers, LDAP servers or KDCs (Kerberos Key Distribution Center) can be configured and is supported. The WSP also supports any number of input points (called "connection" in the XML configuration), meaning Internet address and associated TCP ports (normally 80 and 443), each with certificates for SSL. But at this moment, the configuration tool (WSP-GUI) is limited here; the configuration can still be done by manual XML configuration.

On Unix (including Linux), the WSP needs to call APIs which require superuser rights. This includes bind and listen for well-known ports like 80 and 443. For the WSP on Unix, there is a so-call Listen-Gateway. The WSP is connected to the Listen-Gateway over Unix-domain-sockets and sends commands for special APIs which require superuser rights. The communication is encrypted and protected against replay-attacks. The small Listen-Gateway needs to run with superuser rights, and so the WSP itself can run without superuser rigths and is more secure.

The HOB RD VPN version 2.1 on Linux, including the WSP and its external components, and also the Java application HOBLink JWT (RDP client), are certified according to Common Criteria EAL 4+. Of special interest in the Common Criteria certification was HOB-SSL including the Random-Generator. So HOB can prove that the WSP and HOB-SSL are really secure.

The WSP has numerous diagnostic and trace facilities, called WSP-trace. The WSP-trace can make finely grained recordings of single events or a single client. The WSP-trace is designed in such a way that it also can be used in production environments.

When the configuration of the WSP is changed, it is not necessary to stop and restart the WSP. The WSP can take over the new configuration on the fly. In this way, sessions which were started before the configuration change continue to use the old configuration. Sessions with the WSP, that are started after the configuration change, use the new configuration parameters.

A single WSP can serve more than 100,000 clients simultaneously, depending on the server it runs on. More than 100,000 simultaneous clients have been successfully tested on a server with 4 sockets for Intel CPUs. Each client produced the load of a typical RDP connection, equivalent to a VPN connection between a client and a gateway.

The WSP has cluster functionality, with load-balancing, to find the WSP with the least load for a client. There is communication over TCP between the cluster members, including the load of the WSP. Any number of WSPs can form a cluster, giving a single image for any number of clients.

The WSP is developed in the C and C++ programming languages with small parts in Assembler language.

HOB RD VPN version 2.1 consists of more than 1.2 million lines of code, including the WebSecureProxy, its external components, and the parts running on the client, mostly programmed in Java or JavaScript.

3. Abstract

The Apache Web Server (httpd) was designed and developed to manage short-living half-duplex Web transactions. The HOB WebSecureProxy was designed and developed for Rich Clients, for long-living full-duplex VPN connections.

23.01.16 KB