

Securing Apache web servers

James Harris - j.harris@unimelb.edu.au
LAN Server Group, Information Division

Introduction

Apache is easily the most popular web server on the Internet, estimated to power 62% of all websites¹, and is the most visible and successful project of the Open Source movement. The second most popular web server, Microsoft's IIS, is estimated to hold a 28% market share.

Apache originated in the early 1990s from the National Center for Supercomputing Applications (NCSA) as the HTTPd web server initially developed by Rob McCool. The HTTPd project was discontinued but the web server lived on as a collection of modifications ("a patchy" server) and was renamed Apache.

Eight major contributors created the Apache Group in February 1995 and the first public release (Apache 0.6.2) was made available to the public in April 1995. Apache 1.0 was released on December 1 1995 and had surpassed the HTTPd web server as the most popular server within one year of the Apache Group being formed.

Apache is available for free download from <http://httpd.apache.org/> and compiles and runs on more than twenty eight different platforms. As such, it is available to nearly anyone who wishes to install and run a web server.

Today I will cover general Apache security concepts with specific reference to Windows 2000 and Red Hat Linux 7.3. I will use the latest version of Apache 1.3 (1.3.27) rather than Apache 2.0 which I have not yet used but now comes preinstalled with Red Hat 8.

Obtaining the software

In order to have a secure Apache installation, you need to first obtain the software and ensure it has not been tampered with. The best place to obtain Apache is directly from the Apache HTTP Server Project at or one of their mirrors.

All source code and pre-built binaries also have PGP signatures and MD5 hashes to verify the contents of the download. Keys for the Apache group are available at <http://www.apache.org/dist/httpd/#sig>.

```
$ gpg --import KEYS
$ gpg --verify apache_1.3.27.tar.gz.asc apache_1.3.27.tar.gz
gpg: Signature made Wed 02 Oct 2002 23:07:15 EST using DSA key ID 08C975E5
gpg: Good signature from "Jim Jagielski <jim@apache.org>"
gpg:                aka "Jim Jagielski <jim@jaguNET.com>"
gpg:                aka "Jim Jagielski <jim@jimjag.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:                There is no indication that the signature belongs to the owner.
Primary key fingerprint: 8B39 757B 1D8A 994D F243 3ED5 8B3A 601F 08C9 75E5
```

or

```
$ more apache_1.3.27.tar.gz.md5.gz
MD5 (apache_1.3.27.tar.gz) = 65b89365a65dcad71d4402b4862beea
$ md5sum apache_1.3.27.tar.gz
65b89365a65dcad71d4402b4862beea  apache_1.3.27.tar.gz
```

¹ January 2003 survey - <http://www.netcraft.com/survey/>

More information, including information on how to validate using an MD5 hash under Windows, is available at <http://httpd.apache.org/download.cgi#verify>.

If you are using RPM (Red Hat Package Manager) you can use the command

```
$ rpm -K packagename.rpm
```

to verify a PGP / GPG signature of the RPM.

Installation

I will cover installation under Windows 2000 and Red Hat 7.3. Both installations will be using pre-built packages and will not involve compilation and installation from source code. Building from source allows you greater control of your Apache installation but it does make keeping your software up to date more involved than simply upgrading an RPM.

Windows 2000

Under Windows 2000 (and NT and XP), you have two options to install Apache. You can either install Apache as a service that starts on boot or you can install apache so it is manually started by an individual user. Under most circumstances you would install it as a service running on port 80.

When installed as a service, Apache uses the LocalSystem account which is a very privileged account locally, however it has no network privileges and cannot use network resources such as SMB file shares.

As a result of its Unix heritage, Apache logs most errors to its own `error.log` file within the logs folder of the Apache server installation. Only start up errors are logged to the Windows Application Event Log.

You will be asked to choose an installation directory at the time of install with the default being `C:\Program Files\Apache Group\Apache`.

More details on Windows installations are available at <http://httpd.apache.org/docs/windows.html>

Red Hat Linux 7.3

Installation is simple using RPM:

```
$ rpm -ivh apache_1.3.27-2.i686.rpm
```

By default under Red Hat 7.3, Apache runs as user `apache` and group `apache`. You should never allow Apache to run as root or as any user with administrator privileges such as `bin` or `daemon`.

In order to bind to a privileged port (one below 1024), Apache must start as root and then hand off the process to the `apache` user before it accepts any connection requests.

The Red Hat Apache rpm installs files in to a number of locations which you can check using

```
$ rpm -q apache -l | more
```

All recent releases provide fairly secure default directory and file permissions and ownership. The majority of files are owned by `apache` with configuration files being owned by root and only readable by `apache` using the world permissions.

The `DocumentRoot` directory which contains html files and the `cgi-bin` directory are owned by root and only readable by the Apache user. Root should not be updating these files so it is a good idea to change the ownership accordingly but not to the Apache user.

If you configure Apache to follow symbolic links, you must be careful to take in to account the permissions of any other directories or files that can be reached by following these links.

Configuration

Earlier versions of Apache used three configuration files: `access.conf`, `srm.conf` and `httpd.conf`. Although more recent versions still create the three files, all configuration parameters are now stored in `httpd.conf`. You may also include other files using the `include` parameter.

Scope of directives

By default any directives in the main configuration file apply to the entire server. You can change the configuration for a subset of the server by scoping your directives using the following:

`<Directory>` and `<DirectoryMatch>` which refer to file system locations.

`<Location>` and `<LocationMatch>` which refer to URL locations.

`<Files>` or `<FilesMatch>` which refer to filenames.

`<VirtualHost>` which refers to virtual websites running under the Apache installation.

Security related parameters

The following are some of the more important security related parameters from the `httpd.conf` file. More information on these can be found at <http://httpd.apache.org/docs/mod/core.html>. Note that under Windows you must use forward slashes for file locations rather than the traditional back slashes.

ServerType

In new versions of Red Hat, the Apache server can be run as a standalone service or be managed using `inetd`. For ease of use and security, it is recommended to use the default of standalone. Using `inetd` slows down Apache and is best suited for servers that serve few requests and have limited memory.

ServerRoot

The base directory for Apache from which all relative paths are evaluated. Unless you have a good reason to change this, the default of `/etc/httpd` or `C:/Program Files/Apache Group/Apache` is fine.

DocumentRoot

The `DocumentRoot` of your Apache installation is the directory which holds your web pages. By default it is `/var/www/htdocs` under Red Hat Linux and `C:/Program Files/Apache Group/Apache/htdocs` under Windows 2000.

User, Group

The user and group under which Apache runs.

KeepAlive, MaxKeepAliveRequests, KeepAliveTimeout

`KeepAlive` turns persistent HTTP connections on or off, `MaxKeepAliveRequests` sets the number of requests that can be made in one persistent connection, and `KeepAliveTimeout` specifies how long a persistent connection should be kept open and idle before being terminated.

Correctly set, these can help avoid a denial of service aimed at your server using a large number of persistent connections. You should never set `KeepAliveTimeout` to more than 60 seconds or you will lose most benefits of the feature. After that time a client should be forced to open a new request.

`KeepAlive` is on by default under Windows 2000 but off under Red Hat 7.3. Unless your server is relatively underpowered this should be changed to on as it will increase your server's efficiency under most conditions.

MinSpareServers, MaxSpareServers, StartServers, MaxClients, MaxRequestsPerChild

`Min` and `Max SpareServers` set the minimum and maximum number of idle child processes waiting to service requests, `StartServers` sets the number of child processes launched on start up, `MaxClients` sets a hard limit on the maximum number of requests that can be served, and `MaxRequestsPerChild` sets the maximum number of requests a child process can serve before being killed.

In order to avoid swamping your machine with new processes, Apache uses an exponentially staggered system of spawning children when required. One is spawned during the first second, two during the second, four during the third and so on until the required number have been created. When more than four are spawned during one second a log notice is generated.

Note that `MinSpareServers`, `MaxSpareServers`, and `StartServers` have no effect under Windows and `MaxRequestsPerChild` should be set to 0 (never expire).

ServerSignature

This can be set to `on`, `off`, or `email`. If set to `on`, the server will return a line with the server version and virtual host with any server-generated pages such as error pages. Setting this to `email` will also include a `mailto:` link to the server administrator.

This should be turned off if possible to prevent revealing any information about your server software. If contact information needs to be displayed for error messages then custom error pages can be used instead.

Options Indexes

A browser request to a directory can be handled in one of three ways by Apache: return a listing of the directory, return a file, or return an error page stating that access is denied. If `FancyIndexing` is on, the directory listing will include the modified date, size, and description for each file.

If the option `Indexes` is set and no index file is present the server will return a listing of the directory. This can expose information on your file system you wish to keep private.

The different levels of logging are listed at <http://httpd.apache.org/docs/mod/core.html#loglevel>:

Level	Description	Example
emerg	Emergencies – system is unusable	“Child cannot open lock file. Exiting”
alert	Action must be taken immediately	“getpwuid: couldn’t determine user name from uid”
crit	Critical condition	“socket: Failed to get a socket, exiting child”
error	Error conditions	“Premature end of script headers”
warn (default)	Warning conditions	“child process 1234 did not exit, sending another SIGHUP”
notice	Normal but significant conditions	“httpd: caught SIGBUS, attempting to dump core in...”
info	Informational	“Server seems busy, (you may need to increase StartServers, or Min/MaxSpareServers)...”
debug	Debug-level messages	“Opening config file...”

ScriptAlias

An alias to the file system location of your cgi-bin directory. Your cgi-bin directory should be outside your DocumentRoot and is located at /var/www/cgi-bin and C:/Program Files/Apache Group/Apache/cgi-bin by default.

You should only allow scripts to be executed from within the cgi-bin directory. The Apache documentation at has this to say at http://httpd.apache.org/docs/misc/security_tips.html:

“Allowing users to execute CGI scripts in any directory should only be considered if;

1. You trust your users not to write scripts which will deliberately or accidentally expose your system to an attack.
2. You consider security at your site to be so feeble in other areas, as to make one more potential hole irrelevant.
3. You have no users, and nobody ever visits your server.”

It would be a fairly rare website where all these conditions were met.

AccessFileName, AllowOverride

This parameter defines the type of file used to control access to Apache documents. The default is .htaccess and these files are hidden from browser requests.

.htaccess files can be used to configure security settings on a per directory basis. These settings can override features you have configured in the httpd.conf file. One way of ensuring that .htaccess files are only read in specifically enabled directories is to use:

```
<Directory />
AllowOverride None
</Directory>
```

You can then specifically enable access files on subdirectories:

```
<Location /subdirectory>
AllowOverride All
</Location>
```

Setting AllowOverride None at the root directory will give you slightly better performance as Apache does not have to check for a .htaccess file on every request. You can also set AllowOverride on a

per directive basis. For example you could allow users to protect their directories using `allow/deny` directives in their `.htaccess` files but forbid them from modifying any other directives.

```
<Location /protected_subdirectory
Allow Limit
</Location>
```

LimitRequestBody, LimitRequestFields, LimitRequestFieldSize, LimitRequestLine

These parameters respectively limit the size of the request body, the number of HTTP fields in the request header, the size of the largest field in the request header, and the largest request line that will be accepted by the server.

RLimitCPU, RLimitMEM, RLimitNPROC

Limit server CPU time, memory usage and number of processes the server can spawn. These are not used by default but may be useful in a shared situation such as a professional web host.

CGI scripting

CGI scripting can be one of the biggest security holes in an Apache installation and there are a few basic rules you should follow when configuring your server and when writing or checking scripts. You need to trust the writer of the CGI scripts or be confident that you can spot any potential security flaws.

Firstly, ensure scripts cannot be executed from other sites. This can be done by checking the `HTTP_REFERER` environment variable and will prevent someone malicious passing in variable names and values designed to interfere with the script.

Secondly, ensure scripts do not pass CGI input to the command line without validating, and if necessary, modifying that input. A common breach of security is caused by scripts `execing` or `running` commands without validating the input first. Most major programming languages offer easy functions to strip of input of potentially dangerous data.

Remember all CGI scripts run under the same user and with the same permissions so they have the potential to affect other scripts. One way to avoid this is to use `suEXEC` (<http://httpd.apache.org/docs/suexec.html>) which allows CGI scripts (and SSIs – see below) to be executed under a different user.

Server Side Includes

Server Side Includes (SSI) add the ability to a web server to provide simple parsing of documents on the fly. They can be used to automatically add things like a last modified time and date, or include a standard header or footer on a series of pages. They run directly from Apache, require no additional software, are and simple to enable and hence are widely used. They do however present some potential problems.

Firstly, they increase the load on the server. All files matching the SSI file extension (often `.shtml`) must be parsed by Apache whether or not they include any SSI directives. This is a relatively minor increase in load however in some cases it can become significant.

Secondly, they can pose some of the same risks that CGI scripts in general pose. The `exec` cmd element allows SSI files to execute any CGI script or program with the permissions of the Apache user and group.

If you use SSI's on your Apache server, if possible you should configure your server to use an extension other than .html. This will reduce the load of your server and quickly identifies which files will be parsed. For many sites this is not an option as they wish to parse all their .html files for SSI's.

Unless absolutely necessary you should disable the ability to run scripts from SSI pages. This can be done by replacing `Includes` with `IncludesNOEXEC` in the `Options` directive. Users are still able to include scripts from SSI's if the scripts are in directories specified in a `ScriptAlias` directive.

Chroot Apache

If you wish to go the extra step in securing your Apache installation, you could consider running it in a chroot environment. This means that the root directory Apache sees will not be the same as the real root directory. Hence an Apache request for a path beginning with / will not return the real file system path beginning with /.

This can be time consuming to set up and it is possible to break out of a chroot jail. It also has an effect on your system resources and is not generally widely used. More (relatively old) information can be found at <http://en.tldp.org/LDP/solrhe/Securing-Optimizing-Linux-RH-Edition-v1.3/chap29sec254.html>.

Usage, maintenance and backup

When you have finished installing and configuring your Apache server you should ensure your server is being regularly backed up and that your backups restore successfully. Not having to reinstall and configure Apache can save you a lot of time and effort, not to mention being able to recover all your web content if disaster strikes.

Review your log files regularly and look for errors and anomalies which may indicate security problems. Analog (<http://www.analog.cx/>) or Webalizer (<http://www.mrunix.net/webalizer/>) can help you understand your access logs but you should regularly review your `error_log` file for any unusual entries.

Install TripWire (<http://www.tripwire.org/> - free for Linux, Commercial for other operating systems) to monitor the integrity of your filesystem. If a file has been modified, TripWire will identify the change for you.

Monitor security notifications and keep your software up to date! Like any other piece of software, Apache has security holes which can and will be exploited by crackers. The Carnegie Mellon Software Engineering Institute (<http://www.cert.org>) and the SysAdmin Audit Networking and Security (<http://www.sans.org>) websites are good resources to begin with.

Bugtraq is an email list for information on security related issues. Although the list covers all software products, a search tool is available online at <http://online.securityfocus.com/bid>. If you wish to subscribe to bugtraq, visit <http://www.securityfocus.com/popups/forums/bugtraq/intro.shtml>.

Recent security vulnerabilities found in Apache 1.3

From <http://www.apacheweek.com/features/security-13>

Although Apache is regarded as one of the more secure web servers it has still been affected by a number of security vulnerabilities. These are fixed quickly so keeping your system up to date will be effective at protecting your server from exploitation.

The following vulnerabilities have been found in Apache 1.3 in the last twelve months.

Fixed in Apache httpd 1.3.27

Buffer overflows in ab utility - <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0843>

Shared memory permissions lead to local privilege escalation - <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0839>

Error page XSS using wildcard DNS - <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0840>

Fixed in Apache httpd 1.3.26

Apache Chunked encoding vulnerability - <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0392>

Fixed in Apache httpd 1.3.24

Win32 Apache Remote command execution - <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0061>

Fixed in Apache httpd 1.3.22

Requests can cause directory listing to be displayed - <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0729>

split-logfile can cause arbitrary log files to be written to - <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0730>

Multiviews can cause a directory listing to be displayed - <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0731>

Fixed in Apache httpd 1.3.20

Denial of service attack on Win32 and OS2 - <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-1342>