PrivilegeSeparation

Apache httpd is often used in a setup where the same server has content from multiple independent groups. For example, it may be used for a shared virtual-hosting service where many independent virtual hosts are run under the same instance of httpd.

In some cases, it may be necessary to have some level of privilege separation for the different groups sharing a server. For example, you may need to prevent the content authors on one virtual host from having access to the data files used by CGI scripts on another virtual host.

Why not just switch userids to serve each request?

Feel free to skip this section if the technical details are not important to you.

Unfortunately, you cannot simply configure Apache httpd to adopt different privileges depending on what virtual host is being accessed. This is due to the basic nature of the unix permission model.

In particular, httpd processes are long-lived, since starting new processes takes substantial resources. Further, any httpd process must be able to serve a request for any URL available from the server, since the URL is not known until after httpd processes the request headers. Then to serve each request from a different userid, httpd would need to process request headers as root, and then switch to the appropriate userid after determining the requested URL. There are two major problems with this:

1. Request header processing is one of the most dangerous tasks for a web server. Doing this as root would open the server up to many potential security problems. Instead, Apache httpd is designed to do all request processing as a less-privileged user.
   2. Once the server switches to a less-privileged userid, there is no way to go back to root in order to process further requests. (If there was a way to get back to root, an attacker could obviously use this to subvert any restrictions on the less-privileged userid.) That means a new process would need to be created for each connection, substantially increasing resource usage.

Privilege separation with CGI scripts

If privilege separation is required only for CGI scripts, there are two standard tools available: suexec (included in the Apache HTTP Server distribution) and cgiwrap (a third-party tool).

These suid launchers allow CGI scripts to be run under different userids depending on the URL of the request. Each imposes different restrictions and has different capabilities.

Even in cases where you would ordinarily use an Apache httpd module like php or mod_perl, the simplest, safest, and most performant way to get privilege separation will often be to instead use php or perl as a CGI script under suexec or cgiwrap. For php in particular, suPHP provides an suid launcher (like suexec) combined with an Apache httpd module that makes configuration of PHP as a CGI somewhat easier.

In cases where the performance hit from using CGI scripts is not acceptable, the best alternative is often to use fastcgi to run long-lived cgi processes outside the httpd process. The most up-to-date fastcgi module for Apache httpd is mod_fcgid. Work is currently underway to incorporate fastcgi management into the Apache HTTP Server distribution for a future version.

Be especially careful when configuring and using suid launchers like suexec, cgiwrap, and suPHP. These scripts allow the less-privileged Apache httpd user to do things beyond its standard permissions level (change userid, in particular). Any flaw in these programs or in your configuration of them can lead to serious security problems. The Apache HTTP Server developers have made every effort to keep suexec simple and secure. They cannot vouch for the security of third-party tools like cgiwrap and suPHP.

Using Unix permissions for privilege separation

If privilege separation is needed simply to keep administrators of one site from reading files deployed on another site, proper configuration of unix permissions will often suffice.

In particular, instead of making all files to be served by Apache httpd world-readable, create a group such as `web-content` and run httpd under this group (using the Group directive in httpd.conf). Then site administrators simply place any content needed by the server in the `web-content` group with `chmod 640` (or similar) permissions.

On some operating systems, much-more granular and flexible permissions management is also possible. For example, SELinux allows very precise assignment of permissions beyond what would be possible with the standard Unix permissions model.

Complete privilege separation using a reverse proxy

If complete isolation of virtual hosts (or even particular URLs) is necessary, then the only real solution is to run a separate instance of Apache httpd for each host. When using IP-based virtual hosting, you can simply have each instance listen on a separate IP address. For name-based virtual hosting (or to isolate particular URLs with a virtual host), you can set up a reverse proxy server that dispatches requests to independent httpd instances on different ports. Full details are provided in another recipe.

This solution is flexible, powerful, and very efficient (although the reverse proxy must double-process the http requests). It can, however, be complicated to configure for a large number of hosts.

MPMs that do privilege separation

Several different multi-processing modules (MPMs) have been written to address this problem. These include perchild (now defunct), Metux mpm, Peruser MPM, and MPM-ITK. The first three keep a pool of threads or processes available under each userid and dispatch each request to the appropriate thread pool. Essentially, they are a more efficient version of the reverse proxy solution discussed above. None of these modules are apparently production-ready.

MPM-ITK, on the other hand, processes request headers under root, switches to the target userid, and then kills the httpd process when finished serving the connection. As discussed above, this has serious security and performance implications.

At the present time, you should only attempt to use one of these MPMs if you fully understand the security and performance trade-offs involved.

A caveat on the effectiveness of privilege separation

It is important to remember that, no matter how hard you try to isolate different sites, denial of service attacks will always be possible.