SECURING APACHE : THE BASICS - II

Phases in Apache request processing

For Apache to return a complete response to a client request, more than one module is needed. As already noted, all the modules communicate with or respond to each other through the core. Thus, control is switched back and forth between the core and different modules. All this is done by dividing the request into a set of different phases; let's look at each of the typical request-to-response phases.

URI to file-name translation phase

In a normally configured server, two basic modules, mod_userdir and mod_rewrite, are used in this phase. (However, you should already have disabled mod_userdir, for reasons stated above.) The mod_rewrite module provides you with a flexible mechanism for rewriting the requested URL to a new one. It uses custom rules, which state that if a predefined pattern is found in the requested URL, then it is rewritten to a new one. It is a good idea to install this module, since it reduces the chances of malicious codes in the URL (such as local and remote file inclusions, which we'll discuss in upcoming sections).

Note: URI (Uniform Resource Identifier) is the generic term for a family that includes Uniform Resource Names (URN), Uniform Resource Characteristics (URC), Location-Independent File Names (LIFN), and of course, the URL, the grand daddy of them all.

The authentication and authorisation phase

This phase has sub-phases:

1. Checking the host address and other available information: Here, `mod_access` comes into the picture. Basically, `mod_access` enables you to authorise access based on the host name or IP address from which the request came.
2. Authenticating user ID in the HTTP request: Here `mod_auth`, the default authentication module, is used. It enables you to authenticate users whose credentials (a username and an encrypted password) are stored in text files. From the security perspective, this module is not recommended, since it is not designed to handle a large number of users. Just a few thousand user requests can cause lookup performance to drop dramatically, which may suggest a syn-flooding DoS attack to an attacker. To deal with this problem, we'll be using traffic-shaping modules in subsequent sections. The `mod_auth_anon` module is used for anonymous authentication.

Determining the MIME type of request

In this phase, the content type, encoding, language and other related parameters of the request are worked upon. The `mod_mime` and `mod_mime_magic` basic modules are used here. The former enables Apache to determine MIME type by using file extensions, and provides clients with meta-information about documents. It also enables you to define a handler to determine how the document is processed.
The `mod_mime_magic` module does the same task, but by comparing the first few bytes of the file with a magic value stored in a file. It is only needed when `mod_mime` fails to determine a known MIME type.

The fixing-up phase

The `mod_alias` module works to map one part of the user-perceived file system to another. For example, if a request is received for `http://www.example.com/info/info.php`, then it can be redirected to `http://www.example.com/info/lfu/info.php`. In such tasks, `mod_rewrite` has the

upper hand over mod_alias since mod_rewrite can also map URLs with different hostnames, and can perform complicated tasks such as manipulating the query string. Also in this phase, the mod_env module is used to enable you to pass environment variables to external programs, such as CGI, PHP, mod_perl scripts, or SSIs.

The response phase

This phase can use different modules to create a response. For example: mod_asis can be used for static pages; it enables you to send documents "as-is" to clients, without HTTP headers. This can be useful when redirecting clients without the help of any scripting. mod_cgi invokes CGI scripts and returns the result. mod_include handles server-side includes. (Typically, an SSI is an HTML page with embedded commands for Apache and many others.)

The request logging phase

Knowing who (or what) is accessing your website is very important for security reasons. Apache provides you with mod_log_config, which is responsible for basic logging, and writes CLF files by default. In addition, mod_usertrack can log information on cookies, which is also crucial.

Have a look at the official Apache docs for a complete list of modules, with links to pages documenting their use and directives. Module directives are configured in /etc/httpd.conf, Apache's main configuration file.

These concepts on Apache might have been just revision for you, but we will move to learning how to use these to secure Apache configurations, in articles that follow this one. For now, though, we take a look at something else that is important from the security perspective!

What do you use Apache for? Obviously, to let the world access your Web applications! Now, if your Web application itself has security flaws, then it's no use trying to create

even the most secure configuration for the Apache Web server, because an attacker can enter your site without restriction, through a flawed Web application!

This is what our next section focuses on: identifying the potential flaws in your Web application, beginning with the most common ones, and on how to seal the holes so attackers face a much more secured Web application.

Source : http://www.opensourceforu.com/2010/08/securing-apache-part-1/