SECURING APACHE : THE BASICS - I

Targeted at readers with Web security concerns, information security experts, systems administrators and all those who want to jump-start their careers in Web security, this series of articles intends to cover the strengthening of Web applications and the Apache Web server framework, as well as any possible attacks against both

With about a 54-56 per cent share of the Web server market, according to a survey by netcraft.com, Apache is the second most famous project in the open source world, after Linus Torvalds' Linux kernel. It is, indeed, a de facto standard for Web applications. However, because of the high market share, it has always been a hunting ground for attackers, and is still vulnerable to many known and unknown malicious attacks. An unsecured Apache server can have a devastating effect on your website, the server as a whole, and also on your reputation, if your site is broken into.

However, the response to that is to secure our Web applications and Web server framework — and that's what we will be doing in this series. Starting with an introduction to Apache's architecture, we will look at the many attacks that are possible on your Apache Web server or Web application — and of course, we look at some good security solutions, too. Some golden advice, before we start: read these articles with an attacker's mindset, to gain the upper hand and keep your server secured.

Dissecting Apache

It's quite obvious that to secure any system, you need to start with a good knowledge of its architecture. This section looks at Apache's innards, making it clear how Apache handles applications and modules. The components shown have high interactivity with each other, and that makes security a complex issue. Each type of external system (a database, an LDAP server, a Web service) uses a different language, and allows for different attack vectors, increasing the chances of a security failure.

Figure 1 illustrates a typical Apache server set-up, with an overview of Apache components.

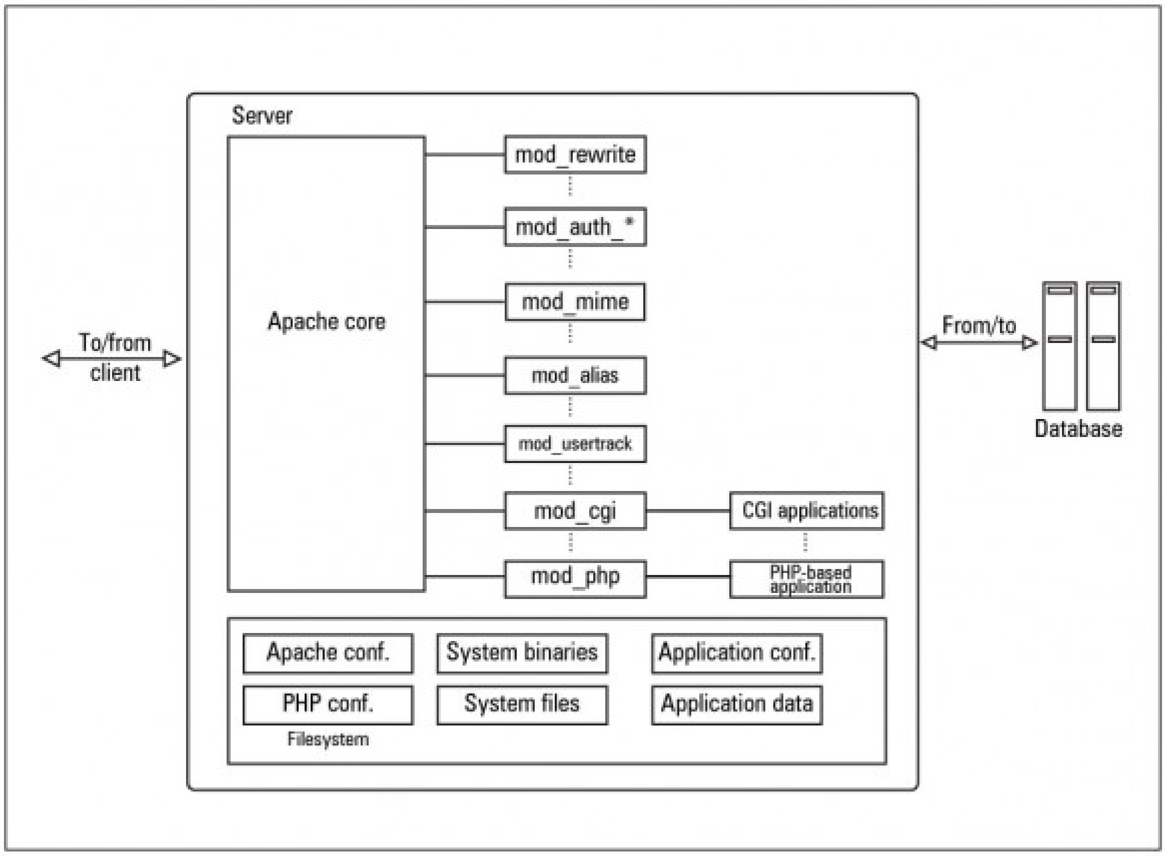

Figure 1: Apache components

The core of Apache implements the basic functionality of the Web server. The following are its main components:

- `http_protocol.c` — contains functions that handle all the data transfers to the client, following the HTTP protocol.
- `http_request.c` — handles the flow of request processing, and controls the dispatching of work to modules, in the appropriate order. It is also responsible for error handling.

- `http_main.c` — starts up the server; this contains the main server loop that waits for, and accepts, connections. It is also in charge of managing timeouts.
- `http_core.c` — is the base component that implements Apache’s basic functionality. Although this component also uses the Apache module API, it is a special one; it has a non-standard file name, `http_core`, instead of the expected `mod_core`. It behaves like a module, but has access to some globals directly, which is not characteristic of a module.
- `http_config.c` — is responsible for managing information about virtual hosts and reading configuration files; it also maintains a list of modules that are called in response to requests.

Apache also supports a variety of features, many implemented as compiled modules, which extend the core functionality. These can range from CGI or PHP support, to authentication and logging schemes. Modules do not interact directly with one another; they interact through the core, since the core contains linked lists of installed modules and their handlers.

Each module has handlers defined for it — for actions like sending a file back to the client (send-as-is handler), treating a file as a CGI script (cgi-script handler) or parsing it for SSI (server-side includes — the server-parsed handler), and many others. Each handler represents a specific action to be performed when a request is received. The core calls the specific handler, thus invoking the module that has defined that handler, for a specific request.

After successful installation and basic configuration of Apache, the first thing you need to do, from the security perspective, is to carefully select your active module set. You should disable the enabled modules (by default)
like `mod_info`, `mod_status`, `mod_userdir` and `mod_include`, unless you have a sound reason for keeping them enabled.
Why, you may well ask. It’s simple — they can expose your server to attack, or yield useful information to an attacker. The first two modules expose the Web server configuration and real-time information as Web pages. The `mod_userdir` module allows

each user account on the server to have a personal website in the home directory, accessible via a `<server URL>/~username` alias.
Apache returns error 404 when a user account, whose personal site is requested, doesn’t exist; and it returns error 403 when a website is not found in that user’s home folder. The errors generated expose valid user account names on the server; it is a common ploy for attackers to try to log in to the server with these discovered accounts, and with commonly used weak passwords. Once they obtain a shell session on the server, there are several privilege-escalation techniques they can try to become the superuser.

The `mod_include` module provides scripting functionality. Though powerful, there are many malicious exploits available that are designed specifically for this module. Disable it if you don’t need it!

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