Running the Apache Web Server

Since Apache is the most popular web server in use today, we will examine some of the many options and configuration settings used to control how Apache works. Since Apache is such a complex program, there are numerous options and settings that we will not have time to explore. Visit the Apache web site for additional information at http://www.apache.org.

Important Files and Directories

Apache can be run either standalone (as a daemon) or from the inetd super-server. Usually you will probably want to run it in standalone mode for optimal performance. If you do run it from inetd, remember every request will launch a new copy of Apache, which means Apache must reread its configuration file, load any needed modules, create/open log files, etc. Basically standalone mode is much faster since those actions are only performed during startup.

Generally, you will find a startup script for standalone mode named **rc.httpd** (or similar) under your **/etc/rc.d** (or perhaps **/etc/rc.d/init.d**) folder.

Before starting Apache for the first time, you should first check its configuration file. This file is always named **httpd.conf**. Finding the location of the file is sometimes a problem however. Use one of the following commands to locate the file:

```
locate httpd.conf
find / -name httpd.conf
```

It is quite possible that several copies of the file may exist on your system. You should edit this file and make changes as needed. I recommend modifying/adding the following two lines initially:

```
ServerAdmin root@localhost
ServerName www.example.com:80
```

Enter the appropriate values and then restart (or start) Apache with a command similar to this:

```
/etc/rc.d/rc.httpd start
```

or

```
/etc/rc.d/init.d/httpd restart
```

Fedora system also have a **service** command that can be used like this:

```
service httpd start
```

or

```
service httpd restart
```

You should now be able to visit the default web site using a web browser by visiting this URL:

http://localhost

or

http://*server-name*

The main directory for the web server is controlled by the *DocumentRoot* setting in the **httpd.conf** file. In most Linux systems, this defaults to the **/var/www/htdocs** or **/var/www/html** directory.

Apache will attempt to load the file named **index.html.*lang*** and send this back to the user. The ***lang*** part is replaced by information sent to Apache by the web browser, such as fr for French, it for Italian, and so on. In our case, we should be viewing the file **index.html.en**, which is the English version of the default web page.

You probably want the web server to startup automatically every time the system is rebooted. The exact procedure to enable this varies slightly depending on whether your version of Linux follows BSD or SysV standards.

For a BSD-style system, you will need to enable the Apache startup script by marking it as executable. Here is the command for a Slackware (BSD-based) system:

```
chmod a+x /etc/rc.d/rc.httpd
```

Under a SysV system, things are more complex. You have to decide which run levels should enable the Apache server. Remember, for a RedHat/Fedora system, the run levels are defined as:

0 – Halt system (power down mode)
1 – Single user mode (also called maintenance mode)

2 – Multiuser mode, without NFS support
3 – Full multiuser mode, with NFS support
4 – Unused
5 – X11 mode (networking with GUI login)
6 – Reboot mode

You should examine the /etc/inittab file since the run levels do vary slightly.

Now, you must decide the run levels where you want Apache to be started. Normally, this will probably be 2, 3 and 5 for a RedHat system. In order to enable this, run the following command:

chkconfig --levels 235 httpd on

That will enable the httpd startup script for run levels 2, 3 and 5.

Other commands used to manage services are:

chkconfig --list
system-config-services
serviceconf (older Fedora versions)
ntsysv

Building Your Own Web Site

To begin creating your own web site, all you have to do is create one or more HTML documents using whatever editor your prefer. Remember to name the main starting point **index.html**. Now you can either copy the new files into **/var/www/htdocs**, or edit the **httpd.conf** and change the *DocumentRoot* setting to point to any directory you choose.

Permissions

If you examine the **httpd.conf** file, you will find a couple of lines similar to this (the user name varies from system to system):

```
User nobody
Group nobody
```

These lines force Apache to assume the permissions of that user and group account. You must make sure your HTML documents (and the directories where they live) can be opened by those user accounts. You can either modify the permissions of the files and directories, or change the ownership. Generally the change in ownership is recommended, since this is safer.

Command to change ownership of the files and/or directories:

```
chown -R nobody.nobody /var/www/htdocs
```

Command to change permissions of the files and/or directories:

```
chmod -R a+r /var/www/htdocs
```

Protecting Web Pages (Authentication)

While dynamic web pages will generally force a user to login by verifying a password against a database, you can easily protect web site pages without needing to to develop dynamic web pages using PHP, Perl or another programming language. This is accomplished by creating a hidden file named “.htaccess” in the directory you wish to protect.

This file is used to override the options in Apache's main httpd.conf file with settings unique to the directory in which the “.htaccess” file is stored. The most common usage for this file is to require users to login before Apache will grant access to the directory and its subfolders.

Example:

```
AuthType Basic
AuthName "Password Protected Area"
AuthUserFile /var/www/htdocs/secret-area/.passwords
Require user randy
```

Explanation:

Directive	***Description***
AuthType	Defines the type of authentication that is required in order to access the folder. Options: Basic – Standard username/password combination. Digest – MD5 encrypted username/password combinations.
AuthName	This setting holds the string that will be displayed to the user when they are asked to login.
AuthUserFile	This defines the file that holds user names and passwords for users. This file is created and managed using the 'htpasswd' or 'htdigest' utilities.
Require	This option defines who is authorized to access the contents of the folder. This can consist of a space separated list of user names or a group names. You may also use the “valid-user” option to allow any user with a valid password to have access.

The Basic authentication type transmits user names and passwords in clear-text and should really only be used with SSL encrypted connections.

The Digest authentication type encrypts passwords using the MD5 algorithm which is safer than clear-text mode, however older web browsers may not support this mode.

Access Control

You may also restrict access to web pages by host name or IP address. This is done using the Allow and Deny keywords in either the "httpd.conf" or ".htaccess" files.

Example:

```
Order Deny,Allow
Deny from all
Allow from class.bamafolks.com
```

Directive	***Description***
Order	Defines the sequence in which the Allow and Deny options are tested.
Deny from	Lists the hosts or IP addresses that will be denied access. Computers that match this setting will be denied access, unless specifically granted access via the "Allow from" option.
Allow from	Similar to the "Deny from" option except it grants access to matching computers instead of denying access.

Combining Authentication and Access Control

If needed, you can combine both access control and authentication so some computers are granted access based on their IP address, while other users on other computers can only gain access by supplying a user name and password. The "Satisfy" directive is used to do this.

Example:

```
AuthType Basic
AuthName "Sensitive Documents"
AuthUserFile /var/www/.passwords
AuthGroupFile /var/www/.groups
Require group customers
Order allow,deny
Allow from 192.168.0.0
Satisfy any
```

Naturally, if you are using dynamically generated web pages, you can also authenticate your users using code if desired. There are also several different modules available for Apache that can be used to authenticate user accounts including the following:

mod_auth – Basic authentication using plain text files (described).
mod_auth_dbm – Authentication via DBM data files.
mod_auth_digest – Encrypted authentication support.
mod_auth_anon – Supports anonymous user access. (FTP style)
mod_auth_ldap – Authentication via LDAP lookups.

See the Apache web site for details on these modules.

Enabling SSL (Secure Socket Layer)

In many cases, you may wish to encrypt the communications between the web server and web browser using SSL. This adds a level of protection since all information will be encoded using cryptography routines, making it less likely that hackers or network spies can grab sensitive information out of the communications packets.

There are several different ways to use SSL within Apache. First, you can just turn on SSL and let it encrypt the data. Second, you can use SSL to restrict the users allowed to visit the web pages using authentication.

In any case, SSL support is added to Apache using a module named mod_ssl. Slackware has installed this module for you, but did not activate it. The primary reason it is not enabled is that before you can really run mod_ssl, you must first obtain or generate a server certificate that Apache will use.

Certificates

A certificate is like a digital fingerprint that uniquely identifies your web server to web browsers. In order to be considered a valid certificate several conditions must be met. First the proper kind of certificate is needed. Most web servers use an X.509 certificate encoded using the RSA algorithm. This certification must be signed by a certificate authority (CA). There are several companies that will sign a certificate for your company, for a fee of course. They include Verisign, Thawte and Uptime Commerce in the United States. Web browsers have a list of recognized certificate authorities which they recognize as valid signers of certificates. Take a look at Konqueror's Crypto configuration settings for a long list of SSL Signers.

When you are first developing a web site, you probably do not want to purchase a certificate immediately. Part of the problem is that the name of the web server included within the certificate itself, therefore you cannot move a certificate from one server to another. The answer in this case is to create your own certificate and sign it yourself. This requires that you act as a certificate authority (CA).

Creating a Self-Signed Certificate

NOTE: Apache version 2 has changed how this works. See the section below for information on using SSL under Apache 2.

The process of creating and signing certificates yourself can be quite complex. I highly recommend downloading the source code for OpenSSL, which includes a handy script called CA.sh (or a Perl version named CA.pl). These scripts makes both creating certification requests and signing them much easier. While you can do so manually, it is very tedious. Some versions of Linux have this utilities already installed these utilities for you. Look for a /etc/ssl/misc directory or search the system for the CA.pl file.

Let's create and install a new certificate for Apache by following these steps:

```
$ cd /etc/ssl/misc
$ ./CA.pl -newca
```

At this point you will be prompted for some information. Remember you are creating your own CA at this point, so a pass phrase is needed. Enter other information as needed to match your organization/server.

Here is an example of responses to make:

```
$ ./CA.pl -newca
CA certificate filename (or enter to create)
{Enter}
Making CA certificate ...
Generating a 1024 bit RSA private key
.....++++++
..................++++++
writing new private key to './demoCA/private/cakey.pem'
Enter PEM pass phrase: {Enter Pass Phrase Here}
Verifying - Enter PEM pass phrase: {Repeat Pass Phrase Here}
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: US
State or Province Name (full name) [Some-State]:Alabama
Locality Name (eg, city) []:Huntsville
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Pearson Consulting
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:ca.bamafolks.com
Email Address []:rlp@bamafolks.com
```

NOTE: If you edit the file /etc/ssl/openssl.cnf you can customize the default values for this process.

Now that you have created a new Certificate Authority, we can next generate a new certificate request by doing the following:

```
$ **./CA.pl -newreq**
Generating a 1024 bit RSA private key
......................++++++
..............++++++
writing new private key to 'newreq.pem'
Enter PEM pass phrase: **{Enter request pass phrase}**
Verifying - Enter PEM pass phrase: **{Repeat pass phrase}**
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:**US**
State or Province Name (full name) [Some-State]:**Alabama**
Locality Name (eg, city) []:**Huntsville**
Organization Name (eg, company) [Internet Widgits Pty Ltd]:**Pearson Consulting**
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:**legolas.home.bamafolks.com**
Email Address []:**rlp@bamafolks.com**

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:**{Enter}**
An optional company name []:**{Enter}**
Request (and private key) is in newreq.pem
```

At this point, you could send the file "newreq.pem" to Verisign (or other CA) where they will sign the request and issue a "real" certificate for a fee. Since we created our own Certificate Authority, we can sign it ourselves by following these steps:

```
$ ./CA.pl -sign
Using configuration from /etc/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem: **{Enter CA pass phrase}**
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Apr 22 19:56:19 2003 GMT
            Not After : Apr 21 19:56:19 2004 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = Alabama
            localityName              = Huntsville
            organizationName          = Pearson Consulting
            commonName                = legolas.home.bamafolks.com
            emailAddress              = rlp@bamafolks.com
        X509v3 extensions:
            X509v3 Basic Constraints:
            CA:FALSE
            Netscape Comment:
```

```
        OpenSSL Generated Certificate
        X509v3 Subject Key Identifier:
        05:AF:65:04:A4:8F:FD:30:25:5C:6D:E3:5B:57:DA:CE:1B:76:10:BA
        X509v3 Authority Key Identifier:
        keyid:76:C6:D7:3F:12:C0:62:0E:A8:17:D1:17:58:D2:DA:BD:06:EA:99:C2
        DirName:/C=AU/ST=Alabama/L=Huntsville/O=Pearson
Consulting/CN=ca.bamafolks.com/emailAddress=rlp@bamafolks.com
        serial:00

Certificate is to be certified until Apr 21 19:56:19 2004 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
Signed certificate is in newcert.pem
```

We have actually created two files at this point. The first is named **newreq.pem** and it holds our pass phrase. The second is named **newcert.pem** and it holds the actual certificate. We must now copy the files so Apache can use them as follows:

```
$ cp newreq.pem /etc/apache/ssl.key/server.key
$ cp newcert.pem /etc/apache/ssl.crt/server.crt
```

Your new certificate is now ready to use. We just need to enable SSL mode for Apache and tell it where to locate the new certificate files we just created. First, you must edit your **httpd.conf** file again and uncomment the following line (near the end of the file):

```
Include /etc/apache/mod_ssl.conf
```

Now we can restart Apache to use SSL. Try doing this:

```
$ /etc/rc.d/rc.httpd restart
```

If you do so, Apache does get restarted, but not with SSL support. The reason is that the script /etc/rc.d/rc.httpd uses the following command internally:

```
apachectl start
```

But in order to activate SSL, this must be modified to read:

```
apachectl startssl
```

You should edit the script, but before you do, let's try starting Apache manually first. Run these commands:

```
$ apachectl stop
$ apachectl startssl
```

You may be a bit surprised by the fact that you need to enter a pass phrase. This is normal. The file /etc/apache/ssl.key/server.key is encrypted and the pass phrase is needed in order to load the file. Unfortunately, this means that even if you edit the /etc/rc.d/rc.httpd file to add the "startssl" option, somebody must still enter the pass phrase whenever the server is restarted.

This is a good idea from a security viewpoint, but not much help when your server loses power at 8 pm on Friday and can't be restarted until someone enters the pass phrase on Monday morning. You can remove the encryption (and the need for a pass phrase) as follows:

```
$ cd /etc/apache/ssl.key
$ cp server.key server.key.orig
$ openssl rsa -in server.key.orig -out server.key
```

It is highly recommended that you also change the permissions on the file so only root has read access.

Apache 2 SSL Certificates

Apache version 2 is a complete rewrite of the Apache system. While it is still mostly backward compatible, it offers a more modular design and better compatibility with a wider range of operating systems along with performance enhancements. Many newer Linux distributions are now bundling this new Apache version.

In addition, the OpenSSL system has also been updated with new utilities, especially related to creating and managing SSL certificates. The new **genkey** tool can be used to both generate and sign SSL certificates much easier than the older CA.pl script.

The tools to create and manage certificates under the latest SSL version should be found in the **/etc/pki/tls/certs** directory. After making that your current working folder, you can enter one of the following commands to generate new certificates:

make XXX.key – generates a public/private key pair
make XXX.csr – generates a certificate signing request
make XXX.crt – generates a self-signed certificate
make XXX.pem – generates both a key pair and self-signed certificate

To generate Apache compatible SSL certificates, use the following commands:

make genkey – generates Apache-compatible key pair

make certreq – generates Apache-compatible signing request
make testcert – generates Apache-compatible self-signed certificate

The only real difference between the first set of commands and the second set is that the second set has built-in default location and names for where the files will be created. Those defaults match the default names Apache uses in its mod_ssl.conf file.

Run this command to generate a test certificate:

```
make testcert
```

Answer the questions as appropriate. When prompted for the Common Name, you must enter the same value as the ServerName directive in the httpd.conf file.

Now you can restart the Apache web server using the **service httpd restart** command. Unfortunately, you will find that you have to enter the passphrase you assigned when generating the SSL certificate.

To remove the passphrase from the private key file, do the following:

```
cd /etc/pki/tls/private
cp localhost.key localhost.key.orig
openssl rsa -in localhost.key.orig -out localhost.key
```

You will have to enter the passphrase one last time. The openssl command will then strip out the encryption and overwrite the localhost.key file with a new version of the certificate. You should then make the files read only using a **chmod 400 *filename*** command.

Adding More Features to Apache

PHP Support

PHP is a very popular way to add support for dynamically generated web pages. This is most useful when you need to extract data from database servers, make remote network connections or perhaps even generate charts and graphs on the fly. PHP support is added by telling Apache to load an extra module via its configuration files. Other languages such as Java, Perl or Ruby can also be enabled this way. To enable PHP, editing **/etc/apache/httpd.conf** and adding/uncommenting the following line:

```
Include /etc/apache/mod_php.conf
```

After you restart Apache, you can create a test script to verify PHP is working. Create the following and copy it to your *DocumentRoot* folder:

```
<?php
echo phpinfo();
?>
```

It should be named something like **test.php**. That means you can view the results by visiting http://localhost/test.php.

Status Information

Apache has a couple of modules included that can be used to monitor and check the health of the server using a web browser. If you edit **/etc/apache/httpd.conf** you can enable the following sections:

```
ExtendedStatus On

<Location /server-status>
   SetHandler server-status
   Order deny,allow
   Deny from all
   Allow from .home.bamafolks.com
</Location>

<Location /server-info>
   SetHandler server-info
   Order deny,allow
   Deny from all
   Allow from .home.bamafolks.com
</Location>
```

This lines enable support for status and server information pages, which you can access by visiting http://localhost/server-status or http://localhost/server-info.

More Modules and Options

Apache supports many other modules in addition to SSL and PHP. Visit the Apache web site at http://www.apache.org for more information.