# Setting up the Apache Web Server

The Apache Web Server (Hyper Text Transfer Protocol) is the most popular web server available. The project gained popularity with Linux in the 1990's as they teamed up to provide big savings to companies wanting to host many enterprise level websites. Now, almost every web provider runs Apache web services exclusively.

Service Name:
    httpd

Package:
    httpd

Config:
    /etc/httpd/conf/httpd.conf

Access Control:
    iptables
    SELinux
    Built in access control

Log Files:
    /var/log/httpd/{access,error}.log

Apache's default configuration will use /var/www/html as the default Document Root (where you put the files which you are sharing) in RPM based environments. Every website has a Document Root associated with it. By default apache will look for a file named index.html (though this can be changed in the configuration file). If that file does not exist then it will load a default "Apache" page, as long as the Apache daemon is running.

To install:

```
# yum -y install httpd
```

Now start Apache, open a browser and type localhost in the address bar and see if you get the default page:

```
# service httpd start
# chkconfig httpd on
```

Now, create an index.html file in /var/www/html and check the site again.

```
# echo “This is a test.” > /var/www/html/index.html
# elinks --dump localhost
```

This should render a page that says, “This is a test.”

Here is a quick command to see if the syntax of your config file is acceptable (you will only need to run this command if named will not start properly on the previous step):

```
# service httpd configtest
```

---

The world’s most popular web server grew up with Linux. Linux and Apache, with MySQL and an interpreted language (i.e. php or perl) made a great open source alternative to the very expensive webservers that were available in the early 1990’s. This configuration is also known as a LAMP stack (Linux, Apache, MySQL, Php or Perl).

One of the reasons it became so popular is because of it’s simplicity. Apache is modular, so features can be added easily. An example is the mod_ssl module, which adds the capability to run a secure web server over the separate protocol of https and port 443. Apache usually runs as http on port 80. Traffic over https is encrypted and uses certificates, usually signed by a third party trusted Certificate Authority, or CA (i.e. VeriSign, etc.).

How Certificates Work

The config file for secure sites is /etc/httpd/conf.d/ssl.conf. In this file the webserver must point to the certificate key (SSLCertificateKey) and file (SSLCertificateFile).

Certificates can seem difficult, but they are no more difficult in concept than ssh private and public keys. Generating the keys and having them signed is what seems to scare most admins, at first.

There are two types of certificates. The first, is called a “self-signed certificate.” Contrary to it’s name it is not signed at all! “Self-signed” in this case implies that you are not having anybody else sign the certificate because you are doing everything yourself. The second is a third party CA signed certificate. With this type of certificate an organization generates a key and a certificate signing request (or csr), instead of a key and a certificate. The csr is then sent to a the CA and, after verification of certain criteria, a signed certificate is returned to the organization. As long as the CA is in the list of third party trusted CAs of the browser being used to access the secure site, then there will be no warnings and the site will be presented as planned.

It is common to place the CA signed certificate in the /etc/pki/tls/certs directory and the private key used to generate the csr in the /etc/pki/tls/private directory and reference them as described above in the /etc/httpd/conf.d/ssl.conf file. However, you can create a separate directory. Just be sure to change the permissions to 600 on the private key, wherever it goes.

If you are using a version 6 Enterprise Linux there is a tool available called "genkey." Even if it is not installed just try to run it with elevated privileges and you will be presented with the option to install it. Once installed you will be presented with a handful of options to run with the "genkey" command:

Usage: genkey [options] servername
--test Test mode, faster seeding, overwrite existing key
--genreq Generate a Certificate Signing Request (CSR)
--makeca Generate a self-signed certificate for a CA
--days Days until expiry of self-signed certificate (default 30)
--renew CSR is for cert renewal, reusing existing key pair, openssl certs only
--cacert Renewal is for a CA certificate, needed for openssl certs only
--nss Use the nss database for keys and certificates
--gdb For package maintainers, to trace into the nss utilities

The easiest way to generated a key for a domain is:

```
# genkey  www.mydomain.com
```

This tool provides everything needed to successfully generate and renew certificates. If this tool is not available, however, there is another tool used for this purpose. The tool is called openssl. On the surface, openssl does not seem to be very well documented, but by looking in the right place everything necessary for creating certificates and keys is there. The manual page for openssl (man openssl) tells you what options can be used with openssl, but does not explain how to use them very clearly. These options can be used to get additional information from the documentation. For example man ca has an "EXAMPLES" section that provides the equivalent commands as the "genkey --makeca" command.

EXAMPLES

| | |
|---|---|
| Sign a certificate request | openssl ca -in req.pem -out newcert.pem |
| Sign a certificate request, using CA extensions | openssl ca -in req.pem -extensions v3_ca -out newcert.pem |
| Generate a CRL | openssl ca -gencrl -out crl.pem |
| Sign several requests | openssl ca -infiles req1.pem req2.pem req3.pem |

man req also has an “EXAMPLES” section that provides the equivalent commands as the “genkey --genreq” command.

EXAMPLES

| | |
|---|---|
| Examine and verify certificate request | openssl req -in req.pem -text -verify -noout |
| Create a private key and then generate a certificate request from it | openssl genrsa -out key.pem 1024<br>openssl req -new -key key.pem -out req.pem |
| The same but just using req | openssl req -newkey rsa:1024 -keyout key.pem -out req.pem |
| Generate a self signed root certificate | openssl req -x509 -newkey rsa:1024 -keyout key.pem -out req.pem |

Password Protecting Web Directories

Perhaps there is just one directory somewhere under the Document Root for a web site that needs to be protected. In this situation a user and password can be setup through apache to require authentication before access is permitted to the files in the protected directory. Within the server definition for a specific site, a <Document...> opening and closing tag can add password access by adding the following:

```
<Directory /directory/to/protect>
    AuthType basic
    AuthName "Protected Directory"
    AuthUserFile /etc/httpd/htpasswd
    require valid-user
</Directory>
```

In this example an Apache user and password must be created. This can be done as follows:

```
# htpasswd -c -m /etc/httpd/htpasswd user
```

Enter a password for user when prompted and verify password.

!!! NOTE !!!

An alternative to putting the <Directory...> tags into the server definition is to create a .htaccess file with the same contents as the opening and closing <Directory..> tags above in the directory to be protected.

!!! NOTE !!!
Only use the -c option to create the specified file. Every subsequent use will overwrite the file.

---

Lab activity

Setup /etc/hosts to point student.thelinuxclub.com to your IP address and make sure that Apache offers secure http for your webserver.

Create a directory called dino which allows only the user dino to access it with the password of bedrock.