Chapter 1

Managing Advanced Apache Features

1.1 Setting up Authenticated Web Servers

1.1.1 The HTTPD Manual

The first thing that we're going to do is use `yum -y install httpd-manual` to install the httpd-manual. This package installs a set of local web-pages that serve as a manual for all Apache configurations. Thus, it provides an easy way to look up information and/or commands offline when needed. Once installed, the manual can be browsed from `http://localhost/manual`.

1.1.2 Apache Users for basic Authentication

Let us consider a website with 3 sections: a public space, a private space and an exclusive space for a certain user '*lisa*'. For authentication we need users - and rather than use Linux users, Apache has it's own system to create users. The users and their passwords are stored in password-files, which are created using the `htpasswd` utility that comes bundled with Apache. To create a password and show it on screen, we use `htpasswd -n <username>` and enter the password when prompted:

```
1 # htpasswd -n somu
2 New password:
3 Re-type new password:
4 somu:$apr1$ejEfs5E7$wzlrLgYWYNKSrG7BQVLIa1
```

However, this is of no use for authentication, since the information isn't stored and Apache can't use it to verify users. Thus, to securely store the password in some passwordfile (that is inaccessible from the internet, so that people can't just *download* it), we choose to store it in `/etc/httpd/htpasswd` with:

```
1 # htpasswd -c /etc/httpd/htpasswd lisa
2 New password:
3 Re-type new password:
4 Adding password for user lisa
5 # cat /etc/httpd/htpasswd
6 lisa:$apr1$pUh9Uxin$zCRJuoWcbkkpDBw04ZaxS0
```

The `-c` option enables the creating of a new password file, or replace (overwrite) an existing one if present. It's very important to only use the -c option for a brand new password file. Every subsequent use should be without any option:

```
# htpasswd /etc/httpd/htpasswd lori
New password:
Re-type new password:
Adding password for user lori
# cat /etc/httpd/htpasswd
lisa:$apr1$r6Xj/zbR$MTFl/9Oq/vcmO1.PLue5W0
lori:$apr1$gZ8ZnGGD$q3GL3wpB0T.JTCa2pw/jD0
```

1.1.3 Directory rules in httpd.conf

Now that we have users, we need to add the specifications for the protected directory in the Apache configs, and dictate when the server should ask for a password and which users should be allowed access. The next part of the required config can either be added in `/etc/httpd/conf/httpd.conf` (preferably at the very bottom to maintain organization) or in a seperate `.htaccess` file within the directory whose access it'll control (For this, the `AllowOverride` directive for the directory can't be set to `none`). In either case, the `<Directory>` directives must be defined in the `httpd.conf` file, or any `.conf` file in the `conf.d` directory. Let us assume we've set up a virtual host called *authtest.somuvmnet.local* which will house the files we need. So, the file `/etc/httpd/conf.d/authtest.conf` will contain:

```
# Rules for the directory private and all its subdirs (which have .htaccess files)
<Directory "/var/www/html/authtest/private">
    AllowOverride all
</Directory>
# Virtual host config
<VirtualHost *:80>
    DocumentRoot /var/www/html/authtest
    ServerName     authtest.somuvmnet.local
    ServerAlias      aut.somuvmnet.local
    ServerAlias      aut.svmn.loc
    ServerAdmin    root@aut.somuvmnet.local
    ErrorLog     "logs/aut_error_log"
    CustomLog    "logs/aut_access_log" combined
</VirtualHost>
```

In the next step, either the steps in Section 9.1.4 or 9.1.5 should be followed:

1.1.4 Controlling access via .htaccess files

The final directory structure of our site will look like:

```
# tree -a /var/www/html/authtest
/var/www/html/authtest
├── index.html
└── private
        ├── .htaccess
        ├── index.html
        └── lisaZone
                ├── .htaccess
                └── index.html
```

For example, if we're trying to restrict access to `/var/www/html/authtest/private` we'll add a new `.htaccess` file in it, with the contents:

```
AuthType Basic
AuthName "Private Space"
AuthUserFile /etc/httpd/htpasswd
Require valid-user
```

Now, we create a directory `/var/www/html/authtest/private/lisaZone` and create a `.htaccess` file within it with the contents:

```
# Only one user (lisa) allowed in lisaZone
AuthType Basic
AuthName "lisaZone"
AuthUserFile /etc/httpd/htpasswd
Require user lisa
```

The above `.htaccess` files sets up the permissions for two folders: *private* and *private/lisaZone*. While any valid Apache user is allowed in the `private` directory, only user lisa can enter `private/lisaZone`, due to the `Require user` directive.

1.1.5 Controlling access via httpd.conf

If we were to put these settings in `httpd.conf` instead of `.htaccess` files in the proper directories, we'd need `<Directory>` directives to define the location where these'd be applied. The following lines would then need to be added to `httpd.conf`:

```
<Directory /var/www/html/authtest/private>
    AllowOverride none
    AuthType Basic
    AuthName "Private Space"
    AuthUserFile /etc/httpd/htpasswd
    Require valid-user
</Directory>

<Directory /var/www/html/authtest/private/lisaZone>
    AllowOverride none
    AuthType Basic
    AuthName "lisaZone"
    AuthUserFile /etc/httpd/htpasswd
    Require user lisa
</Directory>
```

Finally, we add the html content for the site.

1.1.6 Adding HTML Content

The `/var/www/html/authtest/index.html` should contain:

```
<html>
    <head>
        <title>Public Space</title>
    </head>
```

```
    <body>
        <h1>Welcome to the public space!</h1>
        <p>This portion of the website should be accessible by all!</p>
        <p>The only reason this page exists is to act as a control page to test the
        ↪  reactions of the private page to the new configs. Regardless, try to click
        ↪  the link below and see if you can actually access it...</p>
        <a href="private/">Click me to Enter the Restricted Area!</a>
    </body>
</html>
```

The `/var/www/html/authtest/private/index.html` should contain:

```
<html>he \verb|/var/www/html/authtest/private/index.html| should contain:
    <head>
        <title>Private Space</title>
    </head>
    <body>
        <h1>Welcome to the PRIVATE Space</h1>
        <p>This portion of the website should be accessible to only authenticated
        ↪  users</p>
        <p>If you can see this page without authenticating something is wrong with the
        ↪  configs!!!</p>
        <a href="../">Go Back</a>
        <a href="lisaZone/">Go to lisaZone!</a>
    </body>
</html>
```

Finally, the `/var/www/html/authtest/private/lisaZone/index.html` should contain:

```
<html>
    <head>
        <title>LisaZone -- the ultimate protected space</title>
    </head>
    <body>
        <h1>Welcome to the lisaZone</h1>
        <p>This portion of the website should be accessible to user lisa</p>
        <p>If you can see this page and aren't user LISA, something is wrong with the
        ↪  configs!!!</p>
        <p><a href="../">Go Back to Private Zone</a></p>
        <p><a href="../../">Go Back to Public Zone</a></p>
    </body>
</html>
```

Now our site is ready. So, we need to check the httpd config syntax using `httpd -t`. If the syntax is correct, we restart httpd using `systemctl restart httpd`. Then we visit the website by `elinks http://authtest.somuvmnet.local`. Based on authentication, we should be allowed to access the different parts of the sites.

1.2 Configuring Apache for LDAP Authentication

Manual user creation via `htpasswd` gets cumbersome and inefficient when large sites are concerned. In those cases, we can choose LDAP authentication instead. This, however, doesn't mean that local users and groups have to be omitted from the config.

Let us consider an organization `myorg`. Let it contain a group `mygroup`. We want to restrict access to directory `/www/docs/private` to either Apache or LDAP users and groups. Then, configuration in `/etc/httpd/conf/httpd.conf` will look like:

```
<Directory /www/docs/private>
    AuthType Basic
    AuthName "Private"
    AuthBasicProvider file
    AuthUserFile /usr/local/apache/passwd/passwords
    AuthLDAPURL ldap://ldaphost/o=myorg
    AuthGroupFile /usr/local/apache/passwd/groups
    Requrie group GroupName
    Require ldap-group cn=mygroup,o=myorg
</Directory>
```

Here, we have an order of checking for the users/groups, just like in the case of the Pluggable Authentication Module (PAM). First Apache tries to find the user in the password file. If it can't *then* it checks LDAP.

1.3 Enabling CGI Scripts

On any web server, there might be a requirement for dynamic content. This kind of content is generated by a script using a server side scripting language such as CGI, PHP or even python. Scripts becomes crucial when databases are involved since the scripts often fetch information from a database.

1.3.1 CGI

CGI is an abbreviation for **Common Gateway Interface**. It is a specification for information transfer between a web server (such as Apache) and a CGI program/script. CGI is the oldest standard, and even though it can be used by both PHP and python, it's not optimal. To use CGI, we need to use:

```
ScriptAlias    /cgi-bin/    "/var/www/cgi-bin"
```

The CGI scripts must be executable by the `apache` user and group. There are also a certain file context (`httpd_sys_script_exec_t`) on the directory `/var/www/cgi-bin` which is needed by SELinux to permit the execution of such scripts.

1.3.2 PHP

PHP has been much more common than CGI for quite some time now. For PHP scripting to be enabled, the **mod_php** Apache module must be installed and loaded. This simple act itself adds all the necessary configuration to the http configuration, such as setting:

```
SetHandler    application/x-httpd-php
```

This line ensures that PHP can be run from the Apache web server, and other than the installation of *mod_php* Apache module, no manual intervention is needed by default.

1.3.3 Python

In case of python, the name of the required module is **mod_wsgi**. Then we'd need to define a `WSGIScriptAlias` to redirect to the correct application:

```
1 WSGIScriptAlias    /myapp/    /srv/myapp/www/myapp.py
```

To connect to a local database, no additional configuration besides that in the script is necessary. However, if the database is a remote database, then certain SELinux booleans need to be set to true. These are: `httpd_can_network_connect_db` and depending on the database we're using, perhaps `httpd_can_network_connect` as well.

1.4 Understanding TLS protected Web Sites

TLS stands for **Transport Layer Security** and it performs data encryption and identity verification to enhance security. For example, if we visit the website of our bank, we would want to ensure that it's indeed the website of the bank that we're visiting and not some other site that some nefarious agent may have copied to steal data.

Further, we'd also want to ensure that the login credentials, or our personal data that the bank holds (such as account numbers or balances) are not being leaked during transit. Both of these features are provided by TLS. The entire basis of TLS are certificates that act as public keys for websites.

1.4.1 Certificate Authorities

The validity of the certificates are guaranteed by a Certificate Authority (CA), who are 3rd party agents who verify that the organization handing out the certificate are the true owner of the server you're about to access. If so, they sign a digital certificate and provide it to them which they can then give to people who're interested in visiting their site. Now, when we communicate with the site, we can by ourselves check whether their credentials match the one on the certificate to determine if we're communicating with the right server.

1.4.2 Self-Signed Certificates

Certificates can be self-signed too. There are mechanisms via which any server can generate it's own certificate and provide/install it on a client's computer. However, we can't be sure if the organization who just provided us the certificate is really who they claim to be. For example, a site impersonating our bank's website may also hand out a TLS Certificate that matches it's signature. Now, unless we involve CAs, there's no way for us to determine which certificate is the genuine one belonging to our bank.

However, for testing environments, a self-signed certificate is good enough, since no actual valuable data is being passed around, and in case of internal networks, attacks such as *man-in-the-middle* attack (which TLS Certs actively protect against) are useless/impractical. However, signed certificates are essential for production due to the concerns noted above.

1.5 Setting up TLS protected Web Sites