Chapter 14

The Apache Web Server

CERTIFICATION OBJECTIVES

Unix was developed by AT&T in the late 1960s and early 1970s, and it was freely distributed among a number of major universities during those years. When AT&T started charging for Unix, a number of university developers tried to create clones of this operating system. One of these clones, Linux, was developed and released in the early 1990s.

Many of these same universities were also developing the network that evolved into the Internet. With current refinements, this makes Linux perhaps the most Internet-friendly network operating system available. The extensive network services available with Linux are

not only the tops in their field, but they create one of the most powerful and useful Internet-ready platforms available today at any price.

Currently, Apache is the most popular web server on the Internet. According to a Netcraft survey (http://www.netcraft.com), Apache is currently used by nearly 50 percent of all Internet active websites. Apache is included with RHEL 7.

This chapter deals with the concepts surrounding the use the Apache web server at a basic level of configuration.

INSIDE THE EXAM

Inside the Exam

This chapter directly addresses five RHCE objectives. While the objectives specify the HTTP (Hypertext Transfer Protocol) and HTTPS (HTTP, secure) protocols, that is an implicit reference to the Apache web server. It's the only web server currently supported on RHEL 7. The objectives are to

- Configure a virtual host

Virtual hosts are the bread and butter of Apache. They support the configuration of multiple websites on the same server.

- Configure private directories

Private directories on an Apache web server restrict access to a group of users or hosts.

- Configure group-managed content

Sometimes groups of users have to maintain the content of a website jointly. As private directories can be configured for individual users in their home directories, directories can be configured for groups of users in a shared directory.

- Deploy a basic CGI application

Don't worry if you don't know the Common Gateway Interface (CGI), but dynamic content on web pages often depends on scripts such as those associated with CGI.

- Configure TLS security

We have already encountered TLS in the previous chapters. TLS (and its predecessor, SSL) is a suite of protocols used to encrypt network communications. It was originally developed in the mid-1990s to provide certificate-based authentication and secure communications for the Netscape Navigator web browser. Therefore, it's no surprise that today TLS still plays an important role in securing communications on an Apache web server.

In addition, there are the standard requirements for all network services, discussed in Chapters 10 and 11. To summarize, you need to install the service, make it work with SELinux, make sure it starts on boot, configure the service for basic operation, and set up user- and host-based security.

CERTIFICATION OBJECTIVE 14.01

The Apache Web Server

Based on the HTTP daemon (**httpd**), Apache provides simple and secure access to all types of content using the regular HTTP protocol, as well as its secure cousin, HTTPS.

Apache was developed from the server code created by the National Center for Supercomputing Applications (NCSA). It included so many patches that it became known as "a patchy" server. The Apache web server continues to advance the art of the Web and provides one of the most stable, secure, robust, and reliable web servers available. This server is under constant development by the Apache Software Foundation (http://www.apache.org).

For a full copy of Apache documentation, make sure to include the httpd-manual RPM during the installation process. It'll provide a full HTML copy of the Apache manual in the /usr/share/httpd/manual directory, which can be navigated from the local server by pointing a browser to http://localhost/manual.

Apache 2.4

As befits its reliability and stability, RHEL 7 includes an updated version of Apache 2.4. RHEL 6 included an older version of Apache 2.2. However, Apache 2.4 included with RHEL 7 has all the updates needed to support the latest features, with the best possible security from the risks associated with the Internet.

The LAMP Stack

One of the powers of Apache as a web server is the way it can be easily integrated with other software components. The most common set is known as the LAMP stack, which refers to its components: Linux, Apache, MySQL, and one of three scripting languages (Perl, Python, or PHP).

In the RHCE objectives for RHEL 7, you are expected to install MariaDB, a community-developed fork of the MySQL database management system. We will discuss the installation and configuration of MariaDB in Chapter 17.

Installation

The RPM packages required by Apache are included in the Web Server package group. The simplest way to install Apache is with the following command:

```
# yum install httpd
```

However, additional packages are required. It may be simpler to install the mandatory and default packages associated with the Web Server package group with the following command:

```
# yum group install "Web Server"
```

There is also an environment group, named Basic Web Server, that installs the Web server group by default, but also includes some optional groups, such as a MariaDB and PostgreSQL client, Perl and PHP extensions, Java, and so on. If you don't remember the names of available groups, run the **yum group list** command.

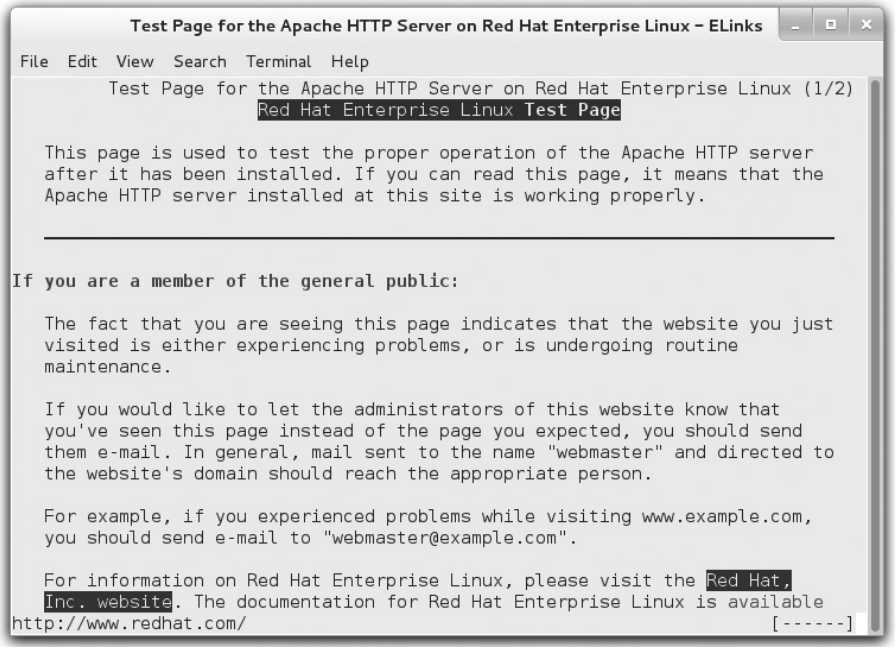
The standard method to start Linux services is via the **systemctl** utility. However, you can stop and start Apache, as well as reload the configuration file gracefully, with the following commands:

```
# apachectl stop
# apachectl start
# apachectl graceful
```

The default Red Hat Apache package supports basic operation, without additional configuration. Once Apache is running, start a web browser and enter a URL of **http://localhost**. For example, Figure 14-1 displays the default home page for Apache, based on the default configuration, in the **elinks** web browser.

FIGURE 14-1

Default installed Apache home page



The web page is based on the contents of the /etc/httpd/conf.d/welcome.conf file, which displays the /usr/share/httpd/noindex/index.html file if there is no index.html file for the default website.

EXERCISE 14-1

Install the Apache Server

In this exercise, you'll install all the packages generally associated with the Apache server. Then you'll configure the system so Apache is active the next time Linux is booted. The twist here is that you'll do it all from the command-line interface. This assumes you've already taken the steps discussed in Chapter 7 to either register with the Red Hat Portal or connect the system to the RHEL 7 (or rebuild DVD) media as a repository.

1. If you're in the GUI, open a command-line console. Press ALT-F2 and log in as the root user.
2. Run the following command to review available groups. You should see "Basic Web Server" in the list of available environment groups.

   ```
   # yum group info
   ```

3. Check which groups are included within the "Basic Web Server" environment group. You should see web-server in the list of mandatory groups.

   ```
   # yum group info "Basic Web Server"
   ```

4. Display the packages included in the web-server group with the following command:

   ```
   # yum group info web-server
   ```

5. You can install all default packages in the web-server package group with the following command:

   ```
   # yum group install web-server
   ```

   If you just install the httpd RPM package, other important packages may not get installed, including mod_ssl, for the secure websites cited in the RHCE objectives.
6. Run the following command to see if Apache is already configured to start at boot:

   ```
   # systemctl is-enabled httpd
   ```

7. Now use the following command to make sure Apache starts in the default target the next time Linux boots normally:

   ```
   # systemctl enable httpd
   ```

8. Start the Apache service with the following command:

   ```
   # systemctl start httpd
   ```

9. If you haven't already done so in Chapter 2, install a text-based web browser. The RHEL 7 standard is **elinks**, which you can install with the following command:

   ```
   # yum install elinks
   ```

10. Now start the ELinks browser, pointing to the local system, with the following command:

    ```
    # elinks http://localhost
    ```

11. Review the result. Do you see the Apache test page?
12. Exit from the ELinks browser. Press Q, and when the Exit ELinks text menu appears, press Y.
13. Back up the default httpd.conf configuration file; a logical location is your home directory.
14. Run the **rpm -q httpd-manual** command to confirm the installation of Apache documentation. Since that package is a default part of the Web Server package group, you shouldn't get a package "not installed" message. However, if you do get that message, install that package with the **yum install httpd-manual** command.
15. Browse the documentation by pointing the ELinks browser to the following URL:

    ```
    # elinks http://localhost/manual
    ```

The Apache Configuration Files

The two key configuration files for the Apache web server are httpd.conf in the /etc/httpd/conf directory and ssl.conf in the /etc/httpd/conf.d directory. The default versions of these files create a generic web server service. All the configuration files are located in three directories: /etc/httpd/conf, /etc/httpd/conf.d, and /etc/httpd/conf.modules.d. They're illustrated in Figure 14-2.

Apache can work with a lot of other software, such as Python, PHP, the Squid Proxy server, and more. If installed, associated configuration files can generally be found in the /etc/httpd/conf. d/ directory.

FIGURE 14-2

Apache configuration files

```
[root@server1 ~]# ls /etc/httpd/conf
httpd.conf  magic
[root@server1 ~]# ls /etc/httpd/conf.d
autoindex.conf  manual.conf  ssl.conf      welcome.conf
fcgid.conf      README       userdir.conf
[root@server1 ~]# ls /etc/httpd/conf.modules.d
00-base.conf  00-lua.conf  00-proxy.conf  00-systemd.conf  10-fcgid.conf
00-dav.conf   00-mpm.conf  00-ssl.conf    01-cgi.conf
[root@server1 ~]#
```

To configure a regular and a secure web server, you'll need to understand the httpd.conf and ssl.conf configuration files in some detail.

Analyze the Default Apache Configuration

Apache comes with a well-commented set of default configuration files. In this section, you'll examine some key directives in the httpd.conf configuration file. Browse through this file in your favorite text editor or using a command pager such as **less**. Before beginning this analysis, remember that the main Apache configuration file incorporates the files in the /etc/httpd/conf.d directory with the following directive:

```
IncludeOptional conf.d/*.conf
```

The httpd.conf file also includes the configuration for external modules with the following directive:

```
Include conf.modules.d/*.conf
```

The difference between **IncludeOptional** and **Include** is that the former does not generate errors if the path specified does not match any file.

There are a couple of basic constructs in httpd.conf. First, directories, files, and modules are configured in "containers." The beginning of the container starts with the name of the directory, file, or module to be configured, inside directional brackets (< >). Examples of this include

```
<Directory "/var/www/html">
<Files "^\.ht*">
<IfModule mime_magic_module>
```

The end of the container is also an expression inside brackets (<>), which starts with a forward slash (/). For the same examples, the ends of the containers would look like

```
</Directory>
</Files>
</IfModule>
```

Next, Apache includes a substantial number of directives—commands that Apache can understand that have some resemblance to English. For example, the **ExecCGI** directive supports executable CGI scripts.

While this provides an overview, the devil is often in the details, which are analyzed (briefly) in the next section. If you've installed the httpd-manual RPM, get the Apache server going and navigate to http://localhost/manual.

The Main Apache Configuration File

This section examines the default Apache configuration file, httpd.conf. We recommend that you follow along on a test system such as server1.example.com. Only the default active directives in that file are discussed here. Read the comments; they include more information and options.

Once Apache and the httpd-manual RPMs are installed per Exercise 14-1, refer to http://localhost/manual/mod/quickreference.html. It provides detailed information on each directive. The default directives are summarized in the following tables. Table 14-1 specifies directives shown near the beginning of the file.

In Tables 14-1 and 14-2, directives are listed in the order shown in the default version of httpd.conf. If you want to experiment with different values for each directive, save the change and then use **systemctl restart httpd** to restart the Apache daemon or **systemctl reload httpd** to just reread the Apache configuration files.

Table 14-2 specifies directives associated with the Main Server Configuration section.

Basic Apache Configuration for a Simple Web Server

As described in Table 14-2, Apache looks for web pages in the directory specified by the **DocumentRoot** directive. In the default httpd.conf file, this directive points to the /var/www/html directory. In other words, all you need to get your web server up and running is to transfer web pages to the /var/www/html directory.

The default **DirectoryIndex** directive looks for an index.html web page file in this directory. A standard RHEL 7 index.html page is available in the /usr/share/doc/HTML/en-US directory. Copy that file to the /var/www/html directory and then navigate to http://localhost with a browser such as ELinks.

TABLE 14-1 Global Environment Directives

Directive	Description
ServerRoot	Sets the default directory for configuration files; any relative path referenced in the configuration is a relative path to the ServerRoot directory.
Listen	Specifies a port and possibly an IP address (for multihomed systems) to listen for requests.
Include	Adds the content of other configuration files.
User	Specifies the username that Apache runs as on the local system.
Group	Specifies the group name that Apache runs as on the local system.

TABLE 14-2 Main Server Configuration Directives

Directive	Description
ServerAdmin	Sets the administrative e-mail address; may be shown (or linked to) on default error pages.
AllowOverride	Supports overriding of previous directives from .htaccess files.
Require	Grants or denies access to a directory for all users or specific users/groups.
DocumentRoot	Assigns the root directory for website files.
Options	Specifies features associated with web directories, such as ExecCGI, FollowSymLinks, Includes, Indexes, MultiViews, and SymLinksIfOwnerMatch.
DirectoryIndex	Specifies files to look for when navigating to a directory; set to index.html by default.
ErrorLog	Locates the error log file, relative to **ServerRoot**.
LogLevel	Specifies the level of log messages.
LogFormat	Sets the information included in log files.
CustomLog	Creates a customized log file, using an existing log format, with a location relative to **ServerRoot**.
ScriptAlias	Similar to **Alias**, maps a web path into a filesystem location outside of **DocumentRoot**; in addition to **Alias**, it tells Apache that the noted directory contains CGI scripts.
TypesConfig	Locates mime.types, which specifies file types associated with extensions.
AddType	Maps filename extensions to a specified content type.
AddOutputFilter	Maps filename extensions to a specified filter.
AddDefaultCharset	Sets a default character encoding.
MIMEMagicFile	Normally uses the file /etc/httpd/conf/magic to determine the MIME type of a file.
EnableSendfile	Uses the sendfile system call to send static files to clients for better performance.

The base location of configuration and log files is determined by the **ServerRoot** directive. The default value from httpd.conf is

```
ServerRoot "/etc/httpd"
```

Figure 14-2 confirms that the main Apache configuration files are stored in the conf/, conf.d/, and conf.d.modules/ subdirectories of **ServerRoot**. Run the **ls -l /etc/httpd** command. Note the soft-linked directories. You should see a link from the /etc/httpd/logs directory to the directory with the actual log files, /var/log/httpd.

Apache Log Files

As suggested earlier, while Apache log files are configured to be saved in the /etc/httpd/logs directory, they're actually stored in the /var/log/httpd directory. In fact, /etc/httpd/logs is a symbolic link to /var/log/httpd. Standard logging information from Apache is stored in two baseline log files. Custom log files may also be configured. Such log files may have different names, depending on how virtual hosts are set up, how secure websites are configured, and how logs are rotated.

Based on the standard Apache configuration files, access attempts are logged in the access_log file and errors are recorded in the error_log file. Standard secure log files include ssl_access_log, ssl_error_log, and ssl_request_log.

In general, it's helpful to configure different sets of log files for different websites. To that end, you should also set up different log files for the secure versions of a website. The traffic on a website is important when choosing a log-rotation frequency.

There are standard Apache log file formats. For more information, take a look at the **LogFormat** directive in Figure 14-3. Three different formats are shown: common, combined (similar to common, but also includes the web page used to get to your site and the user's web browser type and version), and combinedio (same as the combined format, plus a log of the bytes received and sent by the server and client). The **LogFormat** lines include a number of percent signs followed by lowercase letters. These directives determine what goes into the log.

FIGURE 14-3

Specific log formats

```
# LogLevel: Control the number of messages logged to the error_log.
# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.
#
LogLevel warn

<IfModule log_config_module>
    #
    # The following directives define some format nicknames for use with
    # a CustomLog directive (see below).
    #
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
    LogFormat "%h %l %u %t \"%r\" %>s %b" common

    <IfModule logio_module>
      # You need to enable mod_logio.c to use %I and %O
      LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" com
binedio
    </IfModule>

    #
    # The location and format of the access logfile (Common Logfile Format).
    # If you do not define any access logfiles within a <VirtualHost>
    # container, they will be logged here.  Contrariwise, if you *do*
    # define per-<VirtualHost> access logfiles, transactions will be
    # logged therein and *not* in this file.
    #
    #CustomLog "logs/access_log" common
```

You can then use the **CustomLog** directive to select a location for the log file, such as logs/special_access_log, and the desired log file format, such as common. For more information on log files and formats, refer to http://localhost/manual/logs.html.

Some web log analyzers have specific requirements for log file formats. For example, the popular open-source tool AWStats (Advanced Web Statistics) uses the combined log format. AWStats is a great tool for graphically displaying site activity. You can install it from the EPEL (Extra Packages for Enterprise Linux) repository.

CERTIFICATION OBJECTIVE 14.02

Standard Apache Security Configuration

You can configure several layers of security for the Apache web server. Firewalls based on the **firewall-cmd** command can limit access to specific hosts. Security options based on rules in Apache configuration files can also be used to limit access to specific users, groups, and hosts. Of course, secure Apache websites can encrypt communication. If there is a problem, SELinux can limit the risks.

Ports and Firewalls

With the **Listen** and **VirtualHost** directives, the Apache web server specifies the standard communication ports associated with both the HTTP and HTTPS protocols, 80 and 443. To allow external communication through the noted ports, you can set up both ports as trusted services in the Firewall Configuration tool.

Of course, for systems where HTTP and HTTPS are configured on nonstandard ports, you'll have to adjust the associated **firewall-cmd** rules accordingly.

If you just open these ports indiscriminately, the firewall allows traffic from all systems. It may be appropriate to set up a rich rule to limit access to one or more systems or networks. For example, the following custom rich rule allows access to every system except the one with IP address 192.168.122.150, over port 80:

```
firewall-cmd --permanent --add-rich-rule='rule family=ipv4 source \
address=192.168.122.150 service name=http reject'
firewall-cmd --reload
```

Similar rules may be required for port 443. Of course, that depends on the requirements of the job and possibly the RHCE exam.

Apache and SELinux

Take a look at the SELinux settings associated with Apache. To review, SELinux settings mostly fall into two categories: boolean settings and file labels. Start with the file labels.

Apache and SELinux File Labels

The default file labels for Apache configuration files are consistent, as shown in the output to the **ls -Z /etc/httpd** and **ls -Z /var/www** commands. Individual files use the same contexts as their directory. The differences in the file contexts are shown in Table 14-3.

The first five are just the default SELinux contexts for standard directories. For websites where scripts read and/or append data to web forms, you'd consider the last two contexts, which support read/write (rw) and read/append (ra) access.

The contexts listed in Table 14-3 are the most common ones. For a full list of all file contexts related to the Apache web server and their corresponding SELinux labeling rules, run the following command:

```
# semanage fcontext -l | grep httpd_
```

Create a Special Web Directory

In many cases, you'll create dedicated directories for each virtual website. It's better to segregate the files for each website in their own directory tree. But with SELinux, you can't just create a special web directory. You'll want to make sure that new directory at least matches the SELinux contexts of the default /var/www directory.

Run the **ls -Z /var/www** command. Note the SELinux contexts. For most subdirectories of /var/www, the default type is http_sys_content_t. For a newly created /www directory, you could just create a new SELinux rule and change the file contexts with the following

TABLE 14-3 SELinux File Contexts for the Apache Web Server

Directory	SELinux Context Type
/etc/httpd, /etc/httpd/conf, /etc/httpd/conf.d, /etc/httpd/conf.modules.d, /etc/httpd/run	httpd_config_t
/usr/lib64/httpd/modules	httpd_modules_t
/var/log/httpd	httpd_log_t
/var/www, /var/www/html	httpd_sys_content_t
/var/www/cgi-bin	httpd_sys_script_exec_t
n/a	httpd_sys_rw_content_t
n/a	httpd_sys_ra_content_t

commands. The **-R** applies the changes recursively, so the new contexts are applied to all files and subdirectories.

```
# semanage fcontext -a -t httpd_sys_content_t '/www(/.*)?'
# restorecon -R /www
```

The first command creates a file_contexts.local file in the /etc/selinux/targeted/contexts/files directory. If there's also a cgi-bin/ subdirectory, you'll want to set up appropriate contexts for that subdirectory as well with the following command:

```
# semanage fcontext -a -t httpd_sys_script_exec_t '/www/cgi-bin(/.*)?'
```

Apache and SELinux Boolean Settings

Boolean settings are more extensive. For display purposes, we've isolated them in the SELinux Administration tool, as shown in Figure 14-4. Only a few SELinux boolean settings are enabled by default, and they're described in Table 14-4.

FIGURE 14-4 Apache-related SELinux boolean settings

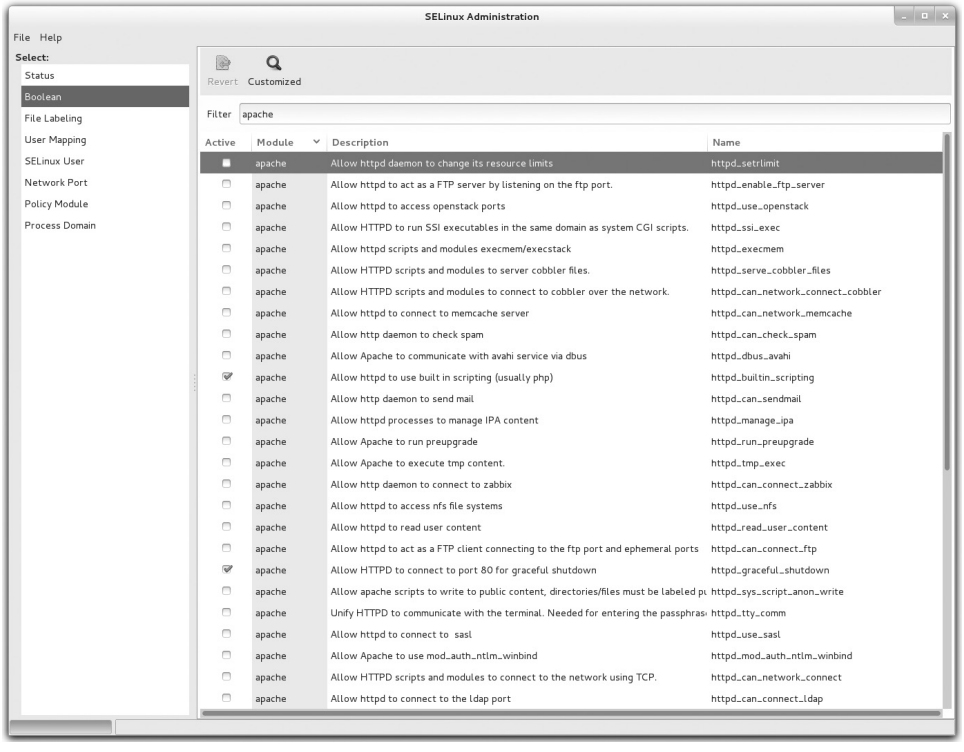


TABLE 14-4 Default Active Apache-Related SELinux Boolean Settings

Active Boolean	Description
httpd_builtin_scripting	Supports the use of scripts (such as PHP)
httpd_enable_cgi	Allows HTTP services to execute CGI scripts, labeled with the httpd_sys_script_exec_t type
httpd_graceful_shutdown	Allows Apache to connect to port 80 for graceful shutdown

Out of the many other SELinux options, pay attention to httpd_enable_homedirs, which supports access to files on user home directories. Other scripts of potential interest relate to interactions with other services, specifically httpd_enable_ftp_server, httpd_use_cifs, and httpd_use_nfs. These options allow Apache to act as an FTP server, as well as to read shared Samba/NFS directories.

The uses of these and the other disabled SELinux Apache-related options from Figure 14-4 are summarized in Table 14-5. All descriptions are based on the perspective "What would happen if the boolean were enabled?" For variety, the terms HTTP and Apache are used interchangeably; strictly speaking, Apache is one option for HTTP and HTTPS services.

Module Management

The Apache web server includes many modular features. For example, it's not possible to set up SSL-secured websites without the mod_ssl package, which includes the mod_ssl.so module along with the ssl.conf configuration file.

A number of other similar systems are organized in modules. Loaded modules are included in standard Apache configuration files with the **LoadModule** directive. A full list of available modules is located in the /usr/lib64/httpd/modules directory, but available modules aren't used unless they're loaded with the LoadModule directive in appropriate Apache configuration files within the /etc/httpd/conf.modules.d directory.

Security Within Apache

You've read about (and hopefully tested) Apache security options related to the zone-based firewall as well as SELinux. Now you'll examine the security options available in the main Apache configuration file, httpd.conf. That file can be modified to secure the entire server or to configure security on a directory-by-directory basis. Directory controls secure access by the server, as well as users who connect to the websites on the server.

To explore the basics of Apache security, let's start with the **ServerTokens** directive:

```
ServerTokens OS
```

TABLE 14-5 Default Inactive Apache-Related SELinux Boolean Settings

Inactive Boolean	Description
httpd_anon_write	Allows the web server to write to files labeled with the public_content_rw_t file type.
httpd_can_check_spam	Works with web-based e-mail applications to check for spam.
httpd_can_network_connect	Allows Apache scripts/modules to establish TCP network connections.
httpd_can_network_connect_cobbler	Enables Apache scripts/modules to connect to Cobbler over the network.
httpd_can_network_connect_db	Allows Apache scripts/modules to connect to a database server over the network.
httpd_can_network_memcache	Enables Apache to connect to a memcache server.
httpd_can_network_relay	Supports the use of the HTTP service as a forward or reverse proxy.
httpd_can_sendmail	Allows Apache to send mail.
httpd_enable_homedirs	Grants Apache permission to access files in user home directories; the files must be labeled with the httpd_sys_content_t SELinux type.
httpd_execmem	Supports access from HTTP modules to executable memory regions; some Java applications may require this permission.
httpd_mod_auth_ntlm_winbind	Supports authentication to Microsoft Active Directory if the mod_auth_ntlm_winbind module is loaded.
httpd_mod_auth_pam	Enables access to PAM authentication modules if the mod_auth_pam module is loaded.
httpd_setrlimit	Allows Apache to modify its resource limits, such as the maximum number of file descriptors.
httpd_ssi_exec	Allows Apache to execute Server Side Include (SSI) scripts in a page.
httpd_tmp_exec	Supports the execution of scripts in the /tmp directory.
httpd_tty_comm	Supports access to a terminal; needed by Apache to prompt for a password if the private key of a TLS certificate is password-protected.
httpd_use_cifs	Enables Apache access to shared Samba directories when labeled with the cifs_t file type.
httpd_use_fuse	Allows Apache to access FUSE file systems, such as GlusterFS volumes.
httpd_use_gpg	Grants Apache permissions to run gpg.
httpd_use_nfs	Enables Apache access to shared NFS directories when labeled with the nfs_t file type.
httpd_use_openstack	Allows Apache to access OpenStack ports.
httpd_sys_script_anon_write	Configures write access by scripts to files labeled with the public_content_rw_t file type.

This line looks deceptively simple; it limits the information that Apache sends in its "Server" response header. This information is sometimes displayed if you navigate to a nonexistent page, but you can also fetch the HTTP headers that Apache sends to clients using the following command:

```
$ curl --head http://localhost
```

Edit the httpd.conf file and add a **ServerTokens OS** line at the top. Then, reload the server configuration by running **systemctl reload httpd** and open the default web page in a browser. You should see the following Server header:

```
Server: Apache/2.4.6 (Red Hat Enterprise Linux)
```

Contrast that output with what happens if you change that line to **ServerTokens Full**:

```
Server: Apache/2.4.6 (Red Hat) OpenSSL/1.0.1e-fips mod_auth_kerb/5.4
mod_fcgid/2.3.9 mod_wsgi/3.4 Python/2.7.5
```

In other words, with one option, outsiders can see whether modules such as FastCGI have been loaded, along with their version numbers. As not everyone updates their software in a perfectly timely manner, imagine what happens when a black hat hacker sees a version that has been compromised. For this reason, we recommend that you set **ServerTokens Prod** to limit the amount of information about the server that is sent to clients.

Next, look at the default access settings for all files and directories in the root filesystem:

```
<Directory />
    AllowOverride None
    Require all denied
</Directory>
```

This configures a very restrictive set of permissions. The **Require all denied** line denies access to all content within the root filesystem for all users. The **AllowOverride None** line disables any .htaccess files. A .htaccess file is placed inside a web directory and contains directives that can override the default web server settings.

However, there's an appropriate use for .htaccess files. For example, in a shared hosting environment, when placed in a subdirectory such as /www/html/customer023, an .htaccess file can override the default server settings and permit access to authenticated users, and such changes would apply only to that directory and its subdirectories.

You can also limit access to all but explicitly allowed domains or IP addresses by adding the following commands to the desired **<Directory>** container:

```
Order Allow,Deny
Allow from example.com
Deny from all
```

The next **<Directory>** container limits access to /var/www, the default location for website files and CGI scripts:

```
<Directory "/var/www">
    AllowOverride None
    # Allow open access:
    Require all granted
</Directory>
```

The **Require all granted** directive grants access to the content of /var/www unconditionally. The next **<Directory>** block regulates access to the /var/www/html directory, which corresponds to the same path referenced by the **DocumentRoot** directive (while the following directives are divided by numerous comments, they are all in the same container):

```
<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

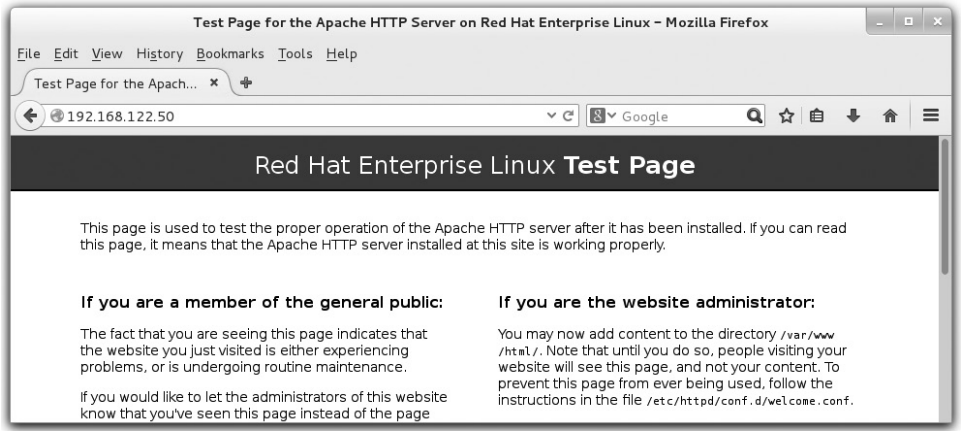
The **Options** directive enables two features: the **Indexes** setting allows readers to see a list of files on the web server if no index.html file is present in the specified directory, and the **FollowSymLinks** option supports the use of symbolic links.

But wait a second! By default, there are no files in the /var/www/html directory. Based on the description, you should navigate to the system in question and see the screen shown in Figure 14-5. As there are no files in the /var/www/html directory, no files are shown in the output.

However, when you navigate to the default website associated with the Apache server, the page shown in Figure 14-6 appears. For more information on how that works, see Exercise 14-2.

FIGURE 14-5 Browse to an index of files.

FIGURE 14-6 Browse to the default Apache test page.



Finally, the **Listen** directive defines the IP address and TCP/IP port for this server. For example, the default shown next means that this server will work with every client that requests a web page from any of the IP addresses of your server on the standard TCP/IP port, 80:

```
Listen 80
```

If more than one IP address is available on the local system, the **Listen** directive can be used to limit access to one specific IP address. For example, if a system has two network cards with IP addresses 192.168.0.200/24 and 192.168.122.1/24, the following directive can help limit access to systems on the 192.168.122.0/24 network:

```
Listen 192.168.122.1:80
```

exam Watch

The RHCE objectives suggest that you need to be ready to configure regular HTTP and secure HTTPS websites.

For secure websites, there's a second **Listen** directive in the ssl.conf file in the /etc/httpd/conf.d directory. The data from this file is automatically incorporated into the overall Apache configuration, courtesy of a directive described in Exercise 14-2. It includes the following directive, which points to the default secure HTTP (HTTPS) port for TCP/IP, 443:

```
Listen 443 https
```

EXERCISE 14-2

The Apache Welcome and the noindex.html Story

In this exercise, you'll trace the story behind the standard test page associated with the Apache web server, like that shown in Figure 14-6. This exercise assumes the httpd package is already installed and the Apache service is running. You'll also see what happens when the path to that web page is broken, with an index of a bunch of test files in the /var/www/html directory.

1. Open the httpd.conf file in the /etc/httpd/conf directory. Find the following line:

   ```
   IncludeOptional conf.d/*.conf
   ```

   The **IncludeOptional conf.d/*.conf** directive includes the contents of *.conf files from the /etc/httpd/conf.d directory in the Apache configuration. Exit from the httpd.conf file.
2. Navigate to the /etc/httpd/conf.d directory. Open the welcome.conf file.
3. Identify and make a note of the parameters of the **Alias** directive.
4. Note the **ErrorDocument** page. While it points to the /.noindex.html file, that's based on the aforementioned **Alias** directive. In other words, you should be able to find the index.html file in the /usr/share/httpd/noindex directory.
5. Take a look at the /usr/share/httpd/noindex/index.html file. To open it up in the ELinks browser, run the **elinks /usr/share/httpd/noindex/index.html** command. The web page that appears should now be familiar.
6. Exit from the browser. Move the welcome.conf file from the /etc/httpd/conf.d directory to a backup location.
7. Reload the Apache configuration with the **systemctl reload httpd** command.
8. Navigate to the localhost system with the **elinks http://127.0.0.1** command. What do you see?
9. Open a second terminal, navigate to the /var/www/html directory, and run the **touch test{1,2,3,4}** command.
10. Reload the browser in the original terminal. In ELinks, CTRL-R reloads the browser. What do you see?
11. Exit from the browser. Restore the welcome.conf file to the /etc/httpd/conf.d directory.

EXERCISE 14-3

Create a List of Files

In this exercise, you'll be setting up a list of files to share with others who access your web server. The process is fairly simple; you'll configure an appropriate firewall rule, create a subdirectory of **DocumentRoot**, fill it with several files, set up the appropriate security contexts, and activate Apache.

1. Make sure the firewall does not block access to ports 80 and 443. One way to do so is with the **firewall-cmd --list-all** command, which displays all services enabled in the default zone. Alternatively, you could use the **firewall-config** GUI tool.
2. Create a subdirectory of **DocumentRoot**, which is /var/www/html by default. For this exercise, we've created the /var/www/html/help directory.
3. Copy the files from the /var/www/manual directory:

   ```
   # cp -a /usr/share/httpd/manual/* /var/www/html/help/
   ```

4. Ensure that the Apache service is running with the following command:

   ```
   # systemctl status httpd
   ```

5. Make sure Apache starts the next time you boot:

   ```
   # systemctl enable httpd
   ```

6. Use the **ls -Zd /var/www/html** and **ls -Z /var/www/html/help** commands to review the security context for the sharing directory and copied files. If the security context doesn't already correspond to the contexts shown here, set them up with the following command:

   ```
   # restorecon -R /var/www/html/help
   ```

7. Start the ELinks browser on the local server, directed at the help/ subdirectory:

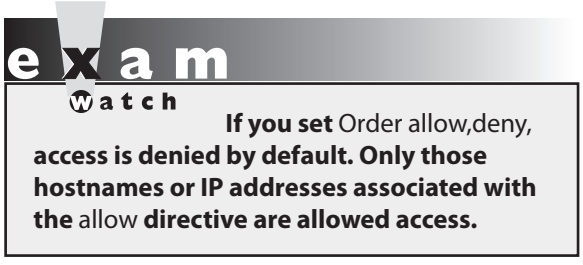
   ```
   # elinks http://127.0.0.1/help
   ```

8. Go to a remote system and try accessing the same web directory. For example, if the IP address of the local system is 192.168.122.50, navigate to http://192.168.122.50/help. If possible, try this a second time from a conventional GUI browser.

Host-Based Security

You can add the **Order**, **allow**, and **deny** directives to regulate access based on hostnames or IP addresses. The following standard command sequence allows access by default. It reads the **deny** directive first:

```
Order deny,allow
```



You can **deny** or **allow** from various forms of hostnames or IP addresses. For example, the following directive denies access from all computers in the mheducation.com domain:

```
Deny from mheducation.com
```

If you don't want to rely on the DNS service, you may prefer to use IP addresses. The first of the following sample directives uses a single IP address; alternatively, you can set up the 192.168.122.0 subnet in partial, netmask, or CIDR (Classless InterDomain Routing) notation, as shown here:

```
Deny from 192.168.122.66
Allow from 192.168.122
Deny from 192.168.122.0/255.255.255.0
Allow from 192.168.122.0/24
```

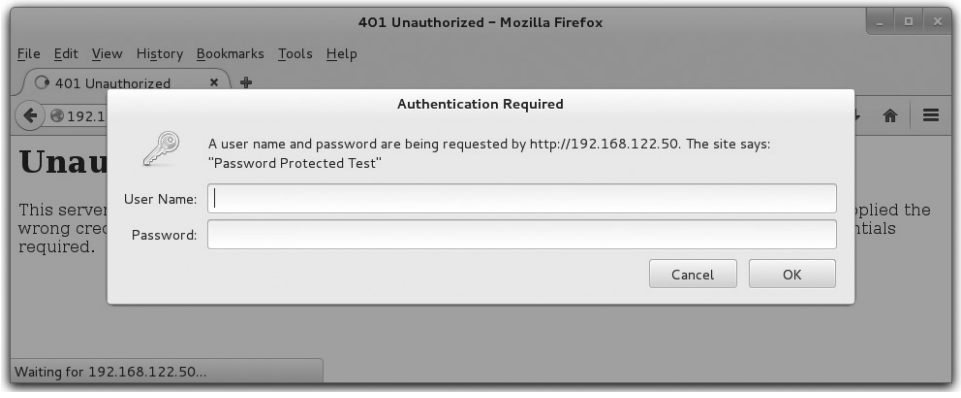
User-Based Security

You can limit access to websites configured on the Apache server to authorized users with passwords. As described shortly, these passwords can be different from the system authentication database.

For example, to configure user-based security for the website described in Exercise 14-3, you'll need to set up a **<Directory>** container on the /var/www/html/help directory. You'll want several commands in the **<Directory>** container:

- To set up basic authentication, you'll need an **AuthType Basic** directive.
- To describe the site to requesting users, you can include an **AuthName "*some comment*"** directive.
- To refer to a web server password database named /etc/httpd/webpass, you'll need a **AuthUserFile /etc/httpd/webpass** directive.
- To limit the site to a single user named engineer1, you could add a **Require user engineer1** directive.
- Alternatively, to limit the site to a group as defined in /etc/httpd/webgroups, you'd add the **AuthGroupFile /etc/httpd/webgroups** directive. You would also need a directive such as **Require group *design***, where *design* is the name of the group specified in webgroups.

FIGURE 14-7 Password protection for a website



Here's an example of code that we've added after the **<Virtual Host>** container:

```
<Directory "/var/www/html/help">
     AuthType Basic
     AuthName "Password Protected Test"
     AuthUserFile /etc/httpd/webpass
     Require user engineer1
</Directory>
```

With this configuration in place, Figure 14-7 illustrates the username/password prompt that appears when you access the http://server1.example.com/help website in a regular web browser. To authenticate, you will also need to create a local password database for Apache. We'll cover this topic in the next section and in Exercise 14-4.

CERTIFICATION OBJECTIVE 14.03

Specialized Apache Directories

In this section, you'll explore several options for specialized Apache directories. It may be appropriate to set up specialized security for some of these directories with the .htaccess file. As suggested earlier, you can set up password protection based on users and groups, which corresponds to the "private directories" cited in the RHCE objectives. One example preconfigured for a private home directory is shown in the conf.d/userdir.conf file. With the right options, such directories can also be managed by members of a group.

Once any changes are made to the Apache configuration files, you may want to test the result. To do so you could run the **systemctl restart httpd** command. Alternatively, to make Apache reload the configuration file without kicking off any currently connected users, run the **systemctl reload httpd** command, which is functionally equivalent to **apachectl graceful**.

Control Through the .htaccess File

With all of the complexity associated with the httpd.conf file, you might look at the .htaccess file and think, "Great, one more complication." But used correctly, the .htaccess file can simplify the list of directives applied to a directory, or a virtual host, because it can be used to override inherited permissions. To do so, you'll need to include the following command in targeted **<Directory>** containers:

```
AllowOverride Options
```

Then you can configure .htaccess files to override previously set directory options. The .htaccess file can be stored in any web directory, labeled with the httpd_sys_content_t SELinux type.

Password-Protected Access

To configure passwords for a website, you need to create a separate database of usernames and passwords. Just as the **useradd** and **passwd** commands are used for regular users, the **htpasswd** command is used to set up usernames and passwords for Apache.

For example, to create a database file named webpass in the /etc/httpd directory, start with the following command:

```
# htpasswd -c /etc/httpd/webpass engineer1
```

The **-c** switch creates the specified file, and the first user is engineer1. You're prompted to enter a password for engineer1. Users in the webpass database do not need to have a regular Linux account. Note the use of the **ServerRoot** directory (/etc/httpd). It's also helpful when configuring virtual hosts.

If you want to add more users to this authentication database, leave out the **-c** switch. For example, the following command sets up a second account for user drafter1:

```
# htpasswd /etc/httpd/webpass drafter1
```

To set up access for more than one user, you may also want to create a group file. For example, to set up the engineer1 and drafter1 users as a group named design, you could add the following line to the /etc/httpd/grouppass file:

```
design: engineer1 drafter1
```

In this case, the AuthUserFile directive would be associated with the /etc/httpd/webpass authentication database, and the AuthGroupFile directive would be associated with the group database.

Home Directory Access

The default /etc/httpd/conf.d/userdir.conf file includes commented suggestions that can enable access to user home directories. One useful option is access to a user's home directory. You can start to set up access to user home directories by changing the following directives from

```
UserDir disabled
#UserDir public_html
```

to

```
#UserDir disabled
UserDir public_html
```

Then anyone will have access to web pages that a user puts in his or her ~/public_html directory. For example, a user named michael can create a /home/michael/public_html directory and add the web pages of his choice.

However, this requires a bit of a security compromise; you need to make michael's home directory executable for all users. This is also known as *701 permissions,* which can be configured with the following command:

```
# chmod 701 /home/michael
```

You'll also need to make the public_html subdirectory executable by all users in the same way with the following command:

```
# chmod 701 /home/michael/public_html
```

But that entails some security risks. Even though a malicious hacker might not be able to directly read the contents of the noted directories, if he sees a script through the resulting website, he'd be able to execute that script as any logged-in user.

There is one alternative for filesystems with Access Control List (ACL) support (see Chapter 4). You could create ACLs on the noted directories specifically for the user named apache. For user michael and his home directory, you could run the following commands:

```
# setfacl -m u:apache:x /home/michael
# setfacl -m u:apache:x /home/michael/public_html
```

Whether permissions are set directly or through ACLs, the logical next step as a web server is to add an index.html file to this directory. For our purposes, it can be a text file.

FIGURE 14-8

View the index .html file for user Michael.

The commented container that follows is one excellent way to help keep home directories thus shared a bit more secure.

In addition, SELinux must be configured to "Allow HTTPD To Read Home Directories," associated with the httpd_enable_homedirs boolean. You can activate that option either with the SELinux Administration tool or with the **setsebool -P httpd_enable_homedirs 1** command.

At that point, a web server that's directed to user michael's directory can read an index .html file in the public_html subdirectory. Figure 14-8 illustrates the result, where the noted text is the only content of index.html. Note that users' public_html directories are accessible at the URL http://servername/~*user*, where *user* is the corresponding username.

Of course, additional settings are included in the userdir.conf file. The container that starts with the following line supports additional levels of access to the public_html subdirectory of all users' home directories:

```
<Directory "/home/*/public_html">
```

The AllowOverride directive allows users to set an .htaccess file to override the default server settings related to document types (FileInfo); access associated with authorization directives (AuthConfig); access secured by directives such as Allow, Deny, and Order; and to override the default directory indexing settings.

```
    AllowOverride FileInfo AuthConfig Limit Indexes
```

The Options directive configures what can be seen in a specific directory, based on content negotiation (MultiViews), a list of files in the current directory (Indexes), an option that allows symbolic links only if associated with the same owner (SymLinksIfOwnerMatch), and also activates an option that does not allow scripts (IncludesNoExec). While it may be a bad security practice to allow a script in a user directory, it may be appropriate for users who are developers on test systems, and possibly during a Red Hat exam. In that case, you would remove the IncludesNoExec option:

```
Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
```

The Require directive limits access only to the listed HTTP methods:

```
Require method GET POST OPTIONS
```

You could combine these directives with password protection. One straightforward possibility is to require the username and password of the user whose home directory is being shared. But as noted earlier, the authentication database generated by **htpasswd** is unrelated to the shadow password suite. You can use the Apache module mod_authnz_ldap if you want to implement authentication and authorization against an LDAP directory. However, this is outside the scope of the RHCE exam.

Group-Managed Directories

You can combine the features of group directories discussed in Chapter 8 with the public_html/ subdirectory just described. However, the steps required to set up a group to manage shared web content are somewhat different. Specifically, to set up a group-managed directory, it's best to start that group as a user. The standard Apache configuration directives for a private user can apply to private groups. Conceptually, you'd take the following steps:

1. Create a regular user.
2. Set up that user with a higher UID and GID number, beyond those associated with existing local and network users.
3. Configure the home directory of that user with the user nobody as the owner. Set up the login shell of that user as /sbin/nologin.
4. Create the public_html subdirectory.
5. Change permissions for the group home directory, with associated subdirectories, to be consistent with the group requirements described in Chapter 8, along with the requirements of the Apache web server. For example, if the new group directory is /home/design, you'd run the following command:

   ```
   # chmod -R 2771 /home/design
   ```

Of course, as discussed in Chapter 8, you could substitute an executable ACL restricted to the user named apache for the execute bit for all users. In that case, you'd run the following commands:

```
# chmod -R 2770 /home/design
# setfacl -m u:apache:x /home/design
# setfacl -m u:apache:x /home/design/public_html
```

6. Log in as a user member of the new group. Create a new file in the public_html subdirectory. Check the ownership of that file; with the Super Group ID (SGID) bit included in the **chmod** command, the group owner should be the owner of all files created in the public_html subdirectory.
7. Make the changes described earlier in this chapter in the httpd.conf file associated with the **UserDir** directive.
8. Make the Apache web server reread the file.

You will have a chance to set this up in one of the chapter labs.

EXERCISE 14-4

Password Protection for a Web Directory

In this exercise, you'll configure password protection for your regular user account on a subdirectory of **DocumentRoot**. This involves the use of the **AuthType Basic**, **AuthName**, and **AuthUserFile** directives. This will be done with the standard Apache website; virtual hosts are covered in the next major section.

1. Back up the main configuration file, httpd.conf, from the /etc/httpd/conf directory. Then open up that file in a text editor.
2. Navigate below the line **<Directory "/var/www/html">**. Create a new container for a **DocumentRoot** subdirectory. One option is the /var/www/html/chapter directory. The first and last directives in the stanza would look like this:

```
<Directory "/var/www/html/chapter">
</Directory>
```

3. Add the following directives: **AuthType Basic** to set up basic authentication, the **AuthName "Password Protected Test"** directive to configure a comment that you should see shortly, and the **AuthUserFile /etc/httpd/testpass** directive to point to a password file. Substitute your regular username for *testuser* in **Require user *testuser***.

```
<Directory "/var/www/html/chapter">
     AuthType Basic
     AuthName "Password Protected Test"
     AuthUserFile /etc/httpd/testpass
     Require user testuser
</Directory>
```

4. Check the syntax of your changes with either of the following commands:

```
# httpd -t
# httpd –S
```

5. Assuming the syntax checks out, make Apache reread the configuration files:

```
# systemctl reload httpd
```

6. Add an appropriate index.html file to the /var/www/html/chapter directory. It's okay to use a text editor to enter a simple line such as "test was successful." No HTML coding is required.
7. Create the /etc/httpd/testpass file with an appropriate password. On our systems, we created a web password for users michael and alex in the noted file with the following commands:

```
# htpasswd -c /etc/httpd/testpass michael
# htpasswd /etc/httpd/testpass alex
```

   If you're adding another user, leave out the **-c** switch.
8. Test the result, preferably from another system. (In other words, make sure the firewall allows access from at least one remote system.)
9. You should now see a request for a username and password, with the comment associated with the **AuthName** directive. Enter the username and password just added to /etc/httpd/testpass and observe the result.
10. Close the browser, and restore any earlier configuration.

CERTIFICATION OBJECTIVE 14.04

Regular and Secure Virtual Hosts

Perhaps the most useful feature of Apache is its ability to handle multiple websites on a single IP address. In a world where there are virtually no more new IPv4 addresses available, this can be useful. To do so, you can configure virtual hosts for regular websites as separate configuration files in the /etc/httpd/conf.d directory. In that way, you can configure multiple domain names such as www.example.com and www.mheducation.com on the same IP address on the same Apache server. This is referred as "name-based" virtual hosting.

Conversely, you can configure a different IP address for each virtual host. This is known as "IP-based" virtual hosting. Both approaches are valid, although name-based virtual hosting is usually preferred because it can significantly reduce your public IP requirements.

The example.com, example.org, and example.net domain names cannot be registered and are officially reserved by the Internet Engineering Task Force (IETF) for documentation. Many other example.* domains are also reserved by appropriate authorities.

In the same fashion, you can create multiple secure websites accessible through the HTTPS protocol. While the details vary, the basic directives associated with both regular and secure virtual hosts are the same.

If you use the ELinks text-based browser to test the connection to the regular and secure virtual websites created in this chapter, there are several things to keep in mind:

- Make sure the /etc/hosts file of the client systems includes the IP address with the specified fully qualified domain names (FQDNs). IP addresses with different FQDNs are normal. (If there's a DNS server for the local network, you can skip this step.)
- Open the /etc/elinks.conf configuration file, and set the first directive in that file to 0, to disable certificate verification.
- To access a regular website, make sure to include the protocol in front of the FQDN, such as http://vhost1.example.com or https://vhost2.example.com.

The beauty of **VirtualHost** is that you can copy virtually the same container to create as many websites on an Apache server, limited only by the capabilities of the hardware. All that's required is one IP address. The next virtual host can be set up with a copy of the original **VirtualHost** container. All that you absolutely have to change for name-based virtual hosts is the **ServerName**. Most administrators will also change the **DocumentRoot**, but even that's not absolutely necessary. You'll see how that works for regular and secure virtual hosts in the following sections.

Be prepared to create multiple websites on an Apache web server using virtual hosts. It's best to create separate VirtualHost **containers in different configuration files for this purpose.**

The Standard Virtual Host

In RHEL 6, the default httpd.conf included sample directives that could be used to create one or more virtual hosts. This is not the case anymore, so if you forget the syntax to create a new virtual host, you may look at the Apache documentation at http://localhost/manual/vhosts.

As noted earlier, the **IncludeOptional conf.d/*.conf** directive automatically includes information from *.conf files in that directory. With that in mind, create and edit a vhost-dummy.conf file in the /etc/httpd/conf.d directory.

Then, add a **<Directory>** container to grant access to the content files of the website. The following example assumes that the new host is named dummy-host.example.com and that the website content is located in the directory /srv/dummy-host/www:

```
<Directory "/srv/dummy-host/www">
    Require all granted
</Directory>
```

Next, add a container for the virtual host configuration:

```
<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host.example.com
    DocumentRoot /srv/dummy-host/www
    ServerName dummy-host.example.com
    ServerAlias www.dummy-host.example.com
    ErrorLog logs/dummy-host.example.com-error_log
    CustomLog logs/dummy-host.example.com-access_log common
</VirtualHost>
```

Port 80 is the default for serving web pages. You could also substitute **<VirtualHost 192.168.122.50:80>**, but in general you can leave that directive as is to support the use of the same IP address for different websites.

If you've read the descriptions of the first two sections of the main part of the httpd .conf file, you should recognize most of these directives. However, each directive points to nonstandard files and directories. To review:

- The e-mail address defined by **ServerAdmin** is included in all error messages that are returned to clients.
- The web pages can be stored in the **DocumentRoot** directory. Make sure the SELinux security contexts of any **DocumentRoot** directory you create are consistent with the contexts of the default /var/www directory (and subdirectories). Apply the **restorecon** and **semanage fcontext -a** commands, as required, to make the security contexts match. Note that by default the SELinux policy already marks the files in /srv/*/www with the httpd_sys_content_t type.
- Based on the **ServerName** directive, Apache knows that requests to http://dummy-host.example.com must use the configuration declared in this <VirtualHost> block.
- **ServerAlias** specifies additional names that the virtual host can be reached as.
- The **ErrorLog** and **CustomLog** directives specify a *relative* log directory, relative to the **ServerRoot**. These files can be found in the /etc/httpd/logs directory. Normally, that directory is soft linked to /var/logs/httpd.

You can add more directives to each virtual host container, to customize the settings for the virtual host relative to the main configuration file. You'll set up a CGI script in a virtual host later in this chapter, with some custom directives.

It's easy to configure a virtual host website. Substitute the IP domain names, directories, files, and e-mail addresses of your choice. Create the **DocumentRoot** directory if it doesn't already exist. To that end, we've set up two virtual hosts with the following containers:

```
<Directory "/srv/vhost1.example.com/www">
    Require all granted
</Directory>
<VirtualHost *:80>
    ServerAdmin webmaster@vhost1.example.com
    DocumentRoot /srv/vhost1.example.com/www
    ServerName vhost1.example.com
    ErrorLog logs/vhost1.example.com-error_log
    CustomLog logs/vhost1.example.com-access_log common
</VirtualHost>

<Directory "/srv/vhost2.example.com/www">
    Require all granted
</Directory>
<VirtualHost *:80>
    ServerAdmin webmaster@vhost2.example.com
    DocumentRoot /srv/vhost2.example.com/www
    ServerName vhost2.example.com
    ErrorLog logs/vhost2.example.com-error_log
    CustomLog logs/vhost2.example.com-access_log common
</VirtualHost>
```

Don't forget to set up the /etc/hosts file, or a DNS server for the local network, with the IP addresses for the virtual host domain names described so far (dummy-host.example.com, vhost1.example.com, and vhost2.example.com).

You should also make sure the SELinux contexts are appropriate. You can test the syntax of any configuration changes with the following command:

```
# httpd –t
```

Apache will verify your configuration or identify specific problems. When you run this command on the default configuration, you'll get the following message:

```
Syntax OK
```

If you've created multiple virtual hosts, you can check them as well with either of the following commands:

```
# httpd -S
# httpd -D DUMP_VHOSTS
```

The output should list the default and individual virtual hosts. For example, we see the following output from our server1.example.com RHEL 7 system:

```
VirtualHost configuration:
*:443           is a NameVirtualHost
  default server server1.example.com (/etc/httpd/conf.d/ssl.conf:56)
  port 443 namevhost server1.example.com (/etc/httpd/conf.d/ssl.conf:56)
wildcard NameVirtualHosts and _default_ servers:
*:80            is a NameVirtualHost
  default server vhost1.example.com (/etc/httpd/conf.d/vhost1.conf:1)
  port 80 namevhost vhost1.example.com (/etc/httpd/conf.d/vhost1.conf:1)
  port 80 namevhost vhost2.example.com (/etc/httpd/conf.d/vhost2.conf:1)
```

Secure Virtual Hosts

If you're configuring a secure web server that conforms to the HTTPS protocol, Red Hat provides a different configuration file for this purpose: ssl.conf in the /etc/httpd/conf.d directory. If this file isn't available, you need to install the mod_ssl package. Before editing this file, back it up.

The first directive in ssl.conf ensures that the server listens on TCP port 443:

```
Listen 443 https
```

As suggested by the title, this configuration file includes a number of other SSL/TLS directives. Generally, no changes are required to the following lines:

```
SSLPassPhraseDialog exec:/usr/libexec/httpd-ssl-pass-dialog
SSLSessionCache         shmcb:/run/httpd/sslcache(512000)
SSLSessionCacheTimeout  300
SSLRandomSeed startup file:/dev/urandom  256
SSLRandomSeed connect builtin
SSLCryptoDevice builtin
```

Now you can set up virtual hosts with the directives that follow. The default ssl.conf file also has a default virtual host container, but it is a bit difficult to read with all of the comments. Therefore, a sample of the revised configuration file, focused on the virtual host container for the vhost1.example.com system, is shown in Figure 14-9. You can edit directly the ssl.conf file, although as a best practice it is recommended to use a separate configuration file in /etc/httpd/conf.d for each virtual host.

In the default version of the ssl.conf file, examine the **<VirtualHost _default_:443>** container. Compare it to the **<VirtualHost *:80>** container in the previous standard virtual hosts configuration. Some changes are required. First, you should replace **_default_** in the **VirtualHost** container with an asterisk (*):

```
<VirtualHost *:443>
```

FIGURE 14-9

Secure virtual host container for vhost1.example.com

```
<VirtualHost *:443>
    ServerAdmin webmaster@vhost1.example.com
    DocumentRoot /srv/vhost1.example.com/www
    ServerName vhost1.example.com

    ErrorLog logs/vhost1_ssl_error_log
    TransferLog logs/vhost1_ssl_access_log
    LogLevel warn

    SSLEngine on
    SSLProtocol all -SSLv2
    SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5
    SSLCertificateFile /etc/pki/tls/certs/localhost.crt
    SSLCertificateKeyFile /etc/pki/tls/private/localhost.key

    <Files ~ "\.(cgi|shtml|phtml|php3?)$">
        SSLOptions +StdEnvVars
    </Files>
    <Directory "/var/www/cgi-bin">
        SSLOptions +StdEnvVars
    </Directory>

    BrowserMatch "MSIE [2-5]" \
             nokeepalive ssl-unclean-shutdown \
             downgrade-1.0 force-response-1.0

    CustomLog logs/ssl_request_log \
              "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"

</VirtualHost>
```

Don't forget to add the https service to the default zone on the firewall:

```
# firewall-cmd --permanent --add-service=https
# firewall-cmd --reload
```

In the ssl.conf file, you should also include **ServerAdmin**, **DocumentRoot**, and **ServerName** directives. Examples of directives that would be consistent with the virtual hosts created in the preceding section include the following:

```
ServerAdmin webmaster@vhost1.example.com
DocumentRoot /srv/vhost1.example.com/www
ServerName vhost1.example.com
```

While the **DocumentRoot** directive can be set to any directory, it's appropriate for organizational purposes to keep the files associated with each virtual host in a dedicated directory.

The standard error log directives can be changed. In fact, if you want log information for each secure website to be set up in different files, they should be changed, as shown next.

Based on the **ServerRoot** directive from the httpd.conf file, these log files can be found in the /var/log/httpd directory.

```
ErrorLog logs/vhost1_ssl_error_log
TransferLog logs/vhost1_ssl_access_log
LogLevel warn
CustomLog logs/vhost1_ssl_request_log \
          "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
```

The TLS directives in the file are based on the default certificates for the localhost system. Shortly, you'll see how to configure a new TLS certificate. The following five directives, in order, activate SSL/TLS, disable the insecure SSL version 2, support a variety of encryption ciphers, and point to the default TLS certificate as well as the TLS key file:

```
SSLEngine on
SSLProtocol all -SSLv2
SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5
SSLCertificateFile /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
```

The container that follows relates to files with extensions associated with dynamic content. For such files, along with any files in the standard CGI directory, standard SSL environment variables are used:

```
<Files ~ "\.(cgi|shtml|phtml|php3?)$">
    SSLOptions +StdEnvVars
</Files>
<Directory "/var/www/cgi-bin">
    SSLOptions +StdEnvVars
</Directory>
```

The following container deals with situations associated with clients running legacy versions of the Microsoft Internet Explorer browser:

```
BrowserMatch "MSIE [2-5]" \
         nokeepalive ssl-unclean-shutdown \
         downgrade-1.0 force-response-1.0
```

Of course, the virtual host container ends with the following directive:

```
</VirtualHost>
```

You do not need to apply all the directives just listed to a new TLS-based virtual host. A minimal configuration is shown next. This includes **DocumentRoot**, **ServerName**, and the directives that enable TLS and configure the certificate path:

```
<VirtualHost *:443>
    DocumentRoot /srv/vhost1.example.com/www
    ServerName vhost1.example.com
```

```
    SSLEngine on
    SSLCertificateFile /etc/pki/tls/certs/vhost1.example.com.crt
    SSLCertificateKeyFile /etc/pki/tls/private/vhost1.example.com.key
</VirtualHost>
```

When Apache is configured with an untrusted certificate, regular clients that access that site get a warning about the secure web host, as shown in Figure 14-10. In the next session you will see how to generate a certificate request to be signed by a certificate authority.

Create a New TLS Certificate

While the default TLS certificate listed in the ssl.conf configuration file can work for basic configuration, you may want to either create a customized self-signed certificate or otherwise use an actual certificate signed from a reputable certificate authority (CA) such as VeriSign or Thawte. Navigate to the /etc/pki/tls/certs directory. Note the file named

FIGURE 14-10 A warning about secure hosts

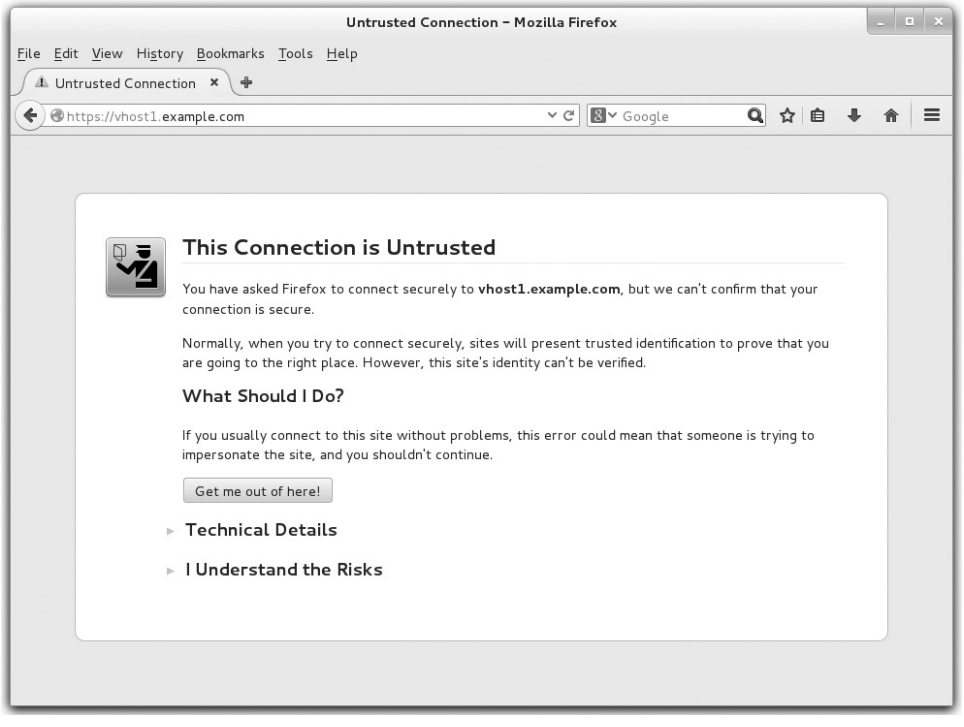
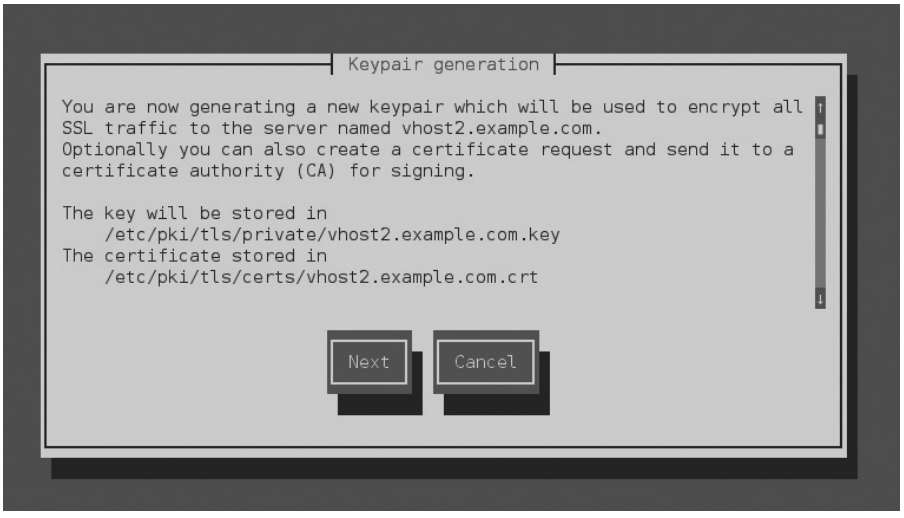


FIGURE 14-11

Generate a self-signed certificate.



Makefile in that directory. The code in that file can be used by the **make** command to create a new certificate for each virtual host. As an alternative, you can use the **genkey** command to automatically generate a private key and a "self-signed certificate" for the cited FQDN, as shown in Figure 14-11.

```
# genkey vhost2.example.com
```

The **genkey** command is convenient because when the process is complete, it automatically writes the key to the /etc/pki/tls/private directory and the certificate to the /etc/pki/tls/certs directory.

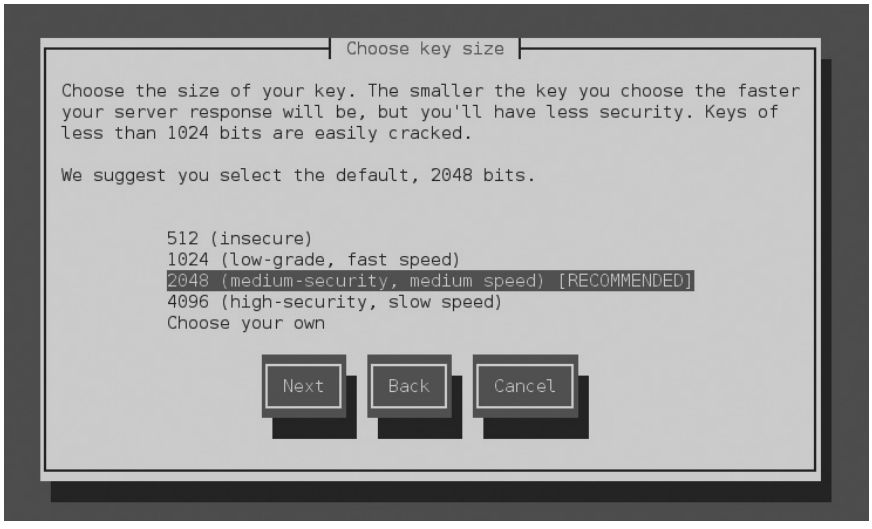
For the purpose of this section, select Next to continue. In the step shown in Figure 14-12, you'd select a key size. In a production environment, the default size of 2048 bits is usually appropriate. But in an exam context, you may want to select a smaller size to save time. The Linux random number generator may require additional activity; this may be an excellent time to put the process aside and do something else.

If you have nothing else to do and need to speed up the process, run some of the scripts in the /etc/cron.daily directory. Run some of the **find** commands described in Chapter 3. Click a bunch of times in an open terminal.

Once a key is generated, you're prompted with a question: whether to generate a Certificate Request (CSR) to send to a CA. Unless you run your own internal CA or you're actually preparing to purchase a signed certificate from a public CA, select No to continue. You're prompted to encrypt the key with a passphrase, as shown in Figure 14-13.

FIGURE 14-12

Select a key size for an SSL certificate.



If security is most important, you should select the Encrypt the Private Key option. If speed is important, avoid the option. Make a choice and select Next to continue. If you did not select the Encrypt the Private Key option, you'll be taken immediately to the certificate details shown in Figure 14-14. Make any appropriate changes and select Next to continue.

If successful, you'll see output similar to that shown in Figure 14-15.

FIGURE 14-13

Option to protect with a passphrase

FIGURE 14-14

SSL certificate details

FIGURE 14-15

SSL certificate command output

```
[root@server1 conf.d]# genkey vhost2.example.com
/usr/bin/keyutil -c makecert -g 2048 -s "CN=vhost2.example.com, O=My Company Ltd, L=Newb
ury, ST=Berkshire, C=GB" -v 1 -a -z /etc/pki/tls/.rand.23390 -o /etc/pki/tls/certs/vhost
2.example.com.crt -k /etc/pki/tls/private/vhost2.example.com.key
cmdstr: makecert

cmd_CreateNewCert
command:  makecert
keysize = 2048 bits
subject = CN=vhost2.example.com, O=My Company Ltd, L=Newbury, ST=Berkshire, C=GB
valid for 1 months
random seed from /etc/pki/tls/.rand.23390
output will be written to /etc/pki/tls/certs/vhost2.example.com.crt
output key written to /etc/pki/tls/private/vhost2.example.com.key


Generating key. This may take a few moments...

Made a key
Opened tmprequest for writing
/usr/bin/keyutil Copying the cert pointer
Created a certificate
Wrote 1682 bytes of encoded data to /etc/pki/tls/private/vhost2.example.com.key
Wrote the key to:
/etc/pki/tls/private/vhost2.example.com.key
[root@server1 conf.d]#
```

Test Pages

You may need to create some index.html files to test virtual hosts in various situations, in various pre-production configurations, or even during an exam. Fortunately, the Red Hat exams don't test knowledge of HTML. You could use Apache's default web page. You can change this or any other web page with a text- or HTML-specific editor.

You can even save a simple text file as index.html. For the purpose of this chapter, all we put into the index.html file for the regular vhost1.example.com website is the following text:

```
Test web page for Virtual Host 1
```

Once appropriate changes were made to Apache configuration files, we restarted the service. When we then ran the **elinks http://vhost1.example.com** command, the screen shown in Figure 14-16 appeared.

Syntax Checkers

In many cases, the **apachectl restart** and the **systemctl restart httpd** commands will reveal syntax problems. But that's just in many cases. In some cases, you might try to restart Apache, proceed to test the result with a client browser, and get frustrated, only to find that Apache did not start because of a syntax error. To minimize the risk of that issue, the following command checks the work that you've done to edit Apache configuration files:

```
# httpd -S
```

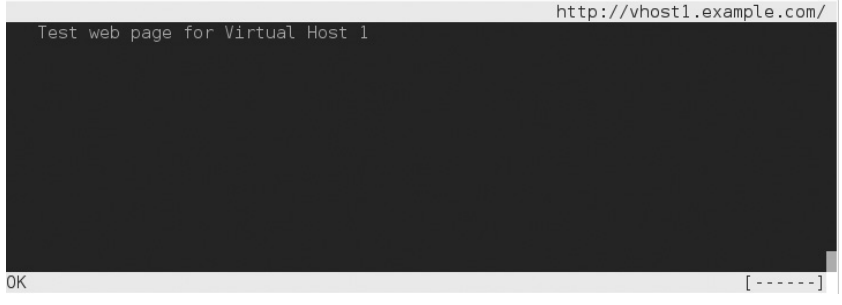
You can also check the log messages captured by the systemd journal:

```
# journalctl -u httpd
```

If no problems are found, you should be able to start the local web server and connect from a client with a browser request.

FIGURE 14-16

A test web page



Apache Troubleshooting

When the right Apache packages are installed, the default configuration normally creates a running system. You can check basic syntax with the **httpd -t** command. But if you're setting up a real website, you probably want more than just the test page. Before making changes, back up the Apache configuration files. If something goes wrong, you can always start over.

Some Apache errors fall into the following categories:

- **Error message about an inability to bind to an address** Another network process may already be using the default http port (80). Alternatively, Apache is configured to listen to the wrong IP address.
- **Network addressing or routing errors** Double-check network settings. For more information on network configuration, see Chapter 3's section on network configuration and troubleshooting.
- **Apache not running** Run **systemctl status httpd**. Check the error_log in the /var/log/httpd directory.
- **Apache not running after a reboot** Run **systemctl is-enabled httpd**. Make sure Apache (httpd) is set to start at the appropriate target during the boot process with the command

```
# systemctl enable httpd
```

Apache administration is a necessary skill for any Linux system engineer. You should develop the ability to install, configure, and troubleshoot Apache quickly. You should also be able to set up and customize virtual websites.

EXERCISE 14-5

Set Up a Virtual Web Server

In this exercise, you'll set up a web server with a virtual website. You can use this technique with different directories to set up additional virtual websites on the same Apache server.

1. Add a virtual website for the fictional company LuvLinex, with a URL of www.example.com. Use the sample configurations from http://localhost/manual/vhosts for hints as needed. Save the configuration in the vhost-luvlinex.conf file inside the /etc/httpd/conf.d directory.
2. Assign the **DocumentRoot** directive to the /luvlinex directory. (Don't forget to create this directory on your system as well.)

3. Grant access to the files to be served using a **Require all granted** directive inside a **<Directory>** block.
4. Open the /luvlinex/index.html file in a text editor. Add a simple line in text format such as

```
This is the placeholder for the LuvLinex Website.
```

5. Save this file.
6. If you've enabled SELinux on this system, you'll have to modify the context type and apply the **restorecon** command to the DocumentRoot directory:

```
# semanage fcontext -a -t httpd_sys_content_t "/luvlinex(/.*)?"
# restorecon -R /luvlinex
```

7. If you're running a DNS service, update the associated database. Otherwise, update /etc/hosts with www.example.com and the appropriate IP address.
8. If you want to check the syntax, run the **httpd -t** and **httpd -D DUMP_VHOSTS** commands.
9. Remember to restart the Apache service; the proper way is with the **systemctl restart httpd** command.
10. Ensure that the local firewall is configured to grant access to connections to the HTTP service:

```
# firewall-cmd --list-all
# firewall-cmd --permanent --add-service=http
# firewall-cmd --reload
```

11. Navigate to a remote system. Update the remote /etc/hosts if appropriate. Open the browser of your choice. Test the access to the configured website (www.example.com).
12. Close the browser on the remote system. Restore the original httpd.conf configuration file.

CERTIFICATION OBJECTIVE 14.05

Deploy a Basic CGI Application

When you see the RHCE objective to "deploy a basic CGI application," the requirement is easier than it looks. In fact, the steps required can be read from the Apache documentation, available from the httpd-manual package. When the application is installed, navigate to http://localhost/manual page. Apache documentation should appear. Select CGI: Dynamic Content for detailed directions, as explained in the following sections.

Apache Configuration Changes for CGI Files

To allow Apache to read CGI files, the conf.modules.d/00-cgi.conf file includes the **LoadModule cgi_module** directive. To control which directories include scripts, Apache includes the **ScriptAlias** directive. For example, the following **ScriptAlias** directive links the /cgi-bin/ URL path to the default /var/www/cgi-bin directory:

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

With this **ScriptAlias** directive, if the website is server1.example.com, scripts can be found in the http://server1.example.com/cgi-bin/ URL.

Alternatively, you can set up CGI scripts in a directory other than /var/www/cgi-bin and change the reference accordingly. The default **<Directory>** block configuration for /var/www/cgi-bin is shown next:

```
<Directory "/var/www/cgi-bin">
    AllowOverride None
    Options None
    Require all granted
</Directory>
```

As suggested in the Apache web server documentation available from the httpd-manual package, you'd need to make changes to allow CGI scripts outside a **ScriptAlias** directory to actually be executable by the Apache server:

```
<Directory "/home/*/public_html">
    AllowOverride None
    Options ExecCGI
    AddHandler cgi-script .pl
    Require all granted
</Directory>
```

As a security measure, the **AllowOverride None** command prevents regular users from changing configuration settings in that directory using an .htaccess file. The **Options ExecCGI** line supports executable scripts in the noted directory. The **AddHandler** directive associates CGI scripts with files with the .pl extension. The **Require all granted** line grants access to the directory from all users.

If CGI scripts are required for one of the previously configured virtual hosts, you can set up a different **ScriptAlias** and a corresponding **<Directory>** container. For the vhost1.example .com site described previously, we add the following directives:

```
ScriptAlias /cgi-bin/ /srv/vhost1.example.com/cgi-bin/
<Directory "/srv/vhost1.example.com/cgi-bin">
    Options none
    Require all granted
</Directory>
```

Set Up a Simple CGI Script in Perl

The Apache documentation includes instructions on how to set up a simple CGI script in the Perl programming language. Make sure that the httpd-manual package is installed and the local httpd service is active. In a browser, navigate to http://localhost/manual. Under the "How-To / Tutorials" section, click CGI: Dynamic Content. Scroll down to the "Writing a CGI Program" section.

In this section, the Apache documentation suggests a simple Perl script, called first.pl, based on the following code:

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
print "Hello, World!";
```

Create this file in the /srv/vhost1.example.com/cgi-bin directory. The first line is similar to the **#!/bin/bash** shebang; in this case, **perl** is the command interpreter. The content type is declared, followed by two newlines (as symbolized by the **\n**). The final line prints the expression commonly used for introductory program scripts. CGI scripts need to be executable by the apache user, and are usually assigned 755 permissions. In other words, once the first.pl file is saved, you'd apply the noted permissions with the following command:

```
# chmod 755 first.pl
```

Run the **ls -Z** command on the script. In the /var/www/cgi-bin directory, it should inherit the httpd_sys_script_exec_t SELinux file type associated with the directory. If necessary, you can apply the context type to the file and directory with the **restorecon** command. If the script directory is set up in a custom directory other than /var/www/cgi-bin, make sure the file type stays applied after a SELinux relabel with the **semanage fcontext -a** command:

```
# semanage fcontext -a -t httpd_sys_script_exec_t \
'/srv/vhost1.example.com/cgi-bin(/.*)?'
```

Connections to a Website

Once a CGI script is configured, you should be able to access that script from a client browser. For the purpose of this exercise, assume the first.pl Perl script has been configured on the server1.example.com system. You should then be able to review the result from a remote system with the **elinks http://vhost1.example.com/cgi-bin/first.pl** command. If successful, the following words should show up in the body of the browser:

```
Hello, world!
```

On occasion, you may see an error message such as "Internal Server Error." The most likely cause is a Perl script that does not have executable permissions for the user named apache. To repeat, that's normally addressed by giving that Perl script 755 permissions.

SCENARIO & SOLUTION	
You need to configure one website.	Install Apache; configure appropriate files in the /var/www/html directory.
You need to configure multiple websites.	With Apache, use **<VirtualHost>** containers configured in separate files within the /etc/httpd/conf.d directory.
You need to configure a secure website.	Configure a virtual host in the ssl.conf file in the /etc/httpd/conf.d directory.
You need a dedicated SSL certificate for the www.example.org website.	Run the **genkey www.example.org** command.
The Apache service is not running after a reboot.	Make sure the httpd service starts in the default target with the **systemctl enable httpd** command. If that's okay, check the contents of the error_log in the /var/log/httpd directory.
CGI scripts in Apache are not running.	In the Apache configuration file, make sure the **ScriptAlias** is pointing to the appropriate directory; if no **ScriptAlias** is configured, ensure that the **ExecCGI** option is active for the script directory and that a **AddHandler** directive specifies the proper script extension; make sure the script is executable by Apache and matches default SELinux contexts in the /var/www/cgi-bin directory.

CERTIFICATION SUMMARY

Apache is the most popular web server in use today. Key packages can be installed from the "Web Server" package group. The httpd-manual package includes a locally browsable manual that can help with other Apache configuration tasks, even during an exam. Key configuration files include httpd.conf in the /etc/httpd/conf directory and ssl.conf in the /etc/httpd/conf.d directory. With the help of sample containers in both noted configuration files, you can create regular and secure virtual hosts for multiple websites on one system, even if only one IP address is available. Related log files are stored in the /var/log/httpd directory.

You can allow access to Apache through ports 80 and 443 to some or all systems with **firewall-cmd**. Apache files and directories are associated with several different SELinux contexts. Different Apache functions may be regulated by a variety of different SELinux boolean settings.

The **Listen VirtualHost** directives direct traffic to the Apache web server to ports such as 80 and 443, along with specified virtual hosts. Host- and user-based security can also be set up within Apache configuration files with commands such as **htpasswd** and directives such as **Require, Allow**, and **Deny**.

With the right security options, user- and group-managed directories are possible. In fact, there's a commented container that can enable content in user home directories. Group-managed directories are somewhat more complex, combining aspects of Apache-based user directories and shared group directories discussed in Chapter 8. Also in security, new certificates can be created for a specific host such as www.example.org with a command like **genkey www.example.org**.

The configuration of CGI content on an Apache website is easier than it looks. In fact, detailed information on the process is provided with Apache documentation, including a Perl script that you can use to confirm that the resulting configuration works.

TWO-MINUTE DRILL

Here are some of the key points from the certification objectives in Chapter 14.

The Apache Web Server

- ❑ Red Hat Enterprise Linux includes the Apache web server, which is currently used by more Internet websites than all other web servers combined.
- ❑ You can install Apache and associated packages as part of the "Web Server" package group.
- ❑ Apache configuration files include httpd.conf in the /etc/httpd/conf directory and ssl.conf in the /etc/httpd/conf.d directory.
- ❑ Log information for Apache is available in the /var/log/httpd directory.

Standard Apache Security Configuration

- ❑ Apache can be secured through **firewall-cmd** rules and various SELinux booleans and contexts.

- ❑ Apache supports security by specifying active ports through the **Listen** and **VirtualHost** directives.
- ❑ Apache supports host-based security by IP address or domain name.
- ❑ Apache supports user-based security by password, with the help of the **htpasswd** command.

Specialized Apache Directories

- ❑ Apache makes it easy to set up access to user home directories in their public_html/ subdirectories.
- ❑ Group-managed directories can be configured in a fashion similar to user home directories.

Regular and Secure Virtual Hosts

- ❑ You can configure multiple websites on your server, even with only one IP address. This is possible through the use of virtual hosts.
- ❑ The RHEL configuration supports virtual hosts for regular and secure websites, usually set up in their own configuration files within the /etc/httpd/conf.d directory.
- ❑ The RHEL configuration includes a default secure virtual host in the /etc/httpd/conf.d/ssl.conf file.
- ❑ SSL certificates can be created with the **genkey** command.

Deploy a Basic CGI Application

- ❑ The use of CGI content depends on configuration options such as **ScriptAlias**, **ExecCGI**, and **AddHandler cgi-script**.
- ❑ Standard CGI scripts need to be executable by the Apache user. If needed, sample instructions are provided in the Apache manual available from the httpd-manual package.

SELF TEST

The following questions will help measure your understanding of the material presented in this chapter. As no multiple choice questions appear on the Red Hat exams, no multiple choice questions appear in this book. These questions exclusively test your understanding of the chapter. It is okay if you have another way of performing a task. Getting results, not memorizing trivia, is what counts on the Red Hat exams. There may be more than one answer to many of these questions.

The Apache Web Server

1. What is the Apache directive that specifies the base directory for configuration and log files?

   _____

2. Once you've modified httpd.conf, what command would make Apache reread this file, without kicking off currently connected users?

   _____

3. What directive specifies the TCP/IP port associated with Apache?

   _____

Standard Apache Security Configuration

4. What command creates the /etc/httpd/passwords file and configures a password for user elizabeth?

   _____

5. If you see the following directives limiting access within the container for a virtual host, what computers are allowed access?

   ```
   Order Allow,Deny
   Allow from 192.168.0.0/24
   ```

   _____

6. What standard services do you need to open in FirewallD to allow access to a regular website and a secure one?

   _____

Specialized Apache Directories

7. What regular permissions would work with a home directory that's shared via Apache?

   _____

8. What regular permissions would work with a shared group directory that's also shared via Apache?

   _____

Regular and Secure Virtual Hosts

9. What file does RHEL provide to help configure a virtual host as a secure server?

   _____

10. If you're creating a name-based virtual host, how many IP addresses would be required for three virtual servers?

    _____

11. To verify the configuration of one or more virtual hosts, what switch can you use with the **httpd** command?

    _____

Deploy a Basic CGI Application

12. What option with the **Options** directive supports dynamic CGI content in an Apache configuration file?

    _____

LAB QUESTIONS

Several of these labs involve configuration exercises. You should do these exercises on test machines only. It's assumed that you're running these exercises on virtual machines such as KVM. For this chapter, it's also assumed that you may be changing the configuration of a physical host system for such virtual machines.

Red Hat presents its exams electronically. For that reason, the labs in this and future chapters are available from the media that accompanies the book, in the Chapter14/ subdirectory. In case you haven't yet set up RHEL 7 on a system, refer to Chapter 1 for installation instructions.

The answers for each lab follow the Self Test answers for the fill-in-the-blank questions.

A SELF TEST ANSWERS

The Apache Web Server

1. The **ServerRoot** directive sets the default directory for the Apache server. Any files and directories not otherwise configured—or configured as a relative directory—are set relative to **ServerRoot**.
2. There are two basic ways to make Apache reread the configuration file without restarting the service. You can keep Apache running and make it reread the file with a command such as **apachectl graceful** or **systemctl reload httpd**. The **kill -HUP $(cat /run/httpd/httpd.pid)** command is also an acceptable answer.
3. The **Listen** directive specifies the TCP port associated with Apache.

Standard Apache Security Configuration

4. The command that creates the /etc/httpd/passwords file and configures a password for user elizabeth is **htpasswd -c /etc/httpd/passwords elizabeth**. If /etc/httpd/passwords already exists, all that's required is **htpasswd /etc/httpd/passwords elizabeth**.
5. As described in the chapter, the **Order Allow,Deny** directive denies access to all systems by default, except those explicitly allowed access. Therefore, access is limited to computers on the 192.168.0.0/24 network.
6. The standard services you need to open in FirewallD to allow access to regular and secure websites are http and https.

Specialized Apache Directories

7. The associated permissions are 701, executable permissions for other users. As "regular permissions" are specified, ACLs are not an option.
8. The associated permissions are 2771, which combine SGID permissions, rwx permissions for a shared group directory, and executable permissions for other users. As "regular permissions" are specified, ACLs are not an option.

Regular and Secure Virtual Hosts

9. The file associated with secure servers for virtual hosts is ssl.conf in the /etc/httpd/conf.d directory.
10. One IP address is required for a name-based virtual server, no matter how many virtual sites are configured.
11. To check your configuration of virtual hosts, you can use one of two switches with the **httpd** command: **httpd -S** checks the configuration file, including virtual host settings. Alternatively, **httpd -D DUMP_VHOSTS** focuses on the virtual host configuration, and is therefore also an acceptable answer.

Deploy a Basic CGI Application

12. The **Options ExecCGI** directive is commonly used in Apache-configured directories that contain CGI scripts such as Perl programs.

LAB ANSWERS

Lab 1

First, make sure the Apache web server is installed. If an **rpm -q httpd** command tells you that it is missing, the Web Server package group has not yet been installed. The most efficient way to do so is with the **yum group install "Web Server"** command. (To find appropriate package group names, run the **yum group list hidden** command.) This assumes a proper connection to a repository, as discussed in Chapter 7.

To start Apache, run the **systemctl start httpd** command. To make sure it starts the next time the system is booted, run the **systemctl enable httpd** command.

Once Apache is installed, you should be able to access it from a browser via http://localhost. From the default Apache configuration file, you can verify that the **DocumentRoot** points to the /var/www/html directory. You can then copy the index.html file from the /usr/share/doc/HTML/en-US directory to the /var/www/html directory. Then you can test the result by navigating once again to http://localhost. If you did not copy the other files associated with the default home page, the display will be missing some icons, but that's not an issue for this lab.

Lab 2

This is an informational lab. When it is complete, you should be able to refer to these Apache configuration hints in situations where this book and the Internet are not available, such as during a Red Hat exam.

Of course, you should study these tips in advance. If you forget the syntax of one or two commands, these files can be a lifesaver.

Lab 3

This lab requires that you create two virtual hosts. While there are certainly other methods to set up different virtual hosts, the description in this lab answer is one method—and it is important that you know at least one method to create a virtual host. One way to make this happen is with the following steps:

1. The **ServerRoot** directive for the system sets the default directory for the Apache server. Any files and directories not otherwise configured—or configured as relative paths—are set relative to **ServerRoot**. Don't change this setting.
2. In the /etc/httpd/conf.d directory, create two files named vhost-big.conf and vhost-small.conf for the configuration of the two virtual hosts.
3. Add a separate **VirtualHost** containers with settings appropriate for the big.example.com virtual host.
4. Assign the **ServerAdmin** to the e-mail address of this website's administrator.
5. Configure a unique **DocumentRoot** directory for big.example.com.
6. Set the first **ServerName** to big.example.com.
7. Add **ErrorLog** and **CustomLog** directives and set them to unique filenames in the /etc/httpd/logs directory (which is linked to the /var/logs/httpd directory). With the default **ServerRoot**, you can use a relative path such as the following:

   ```
   ErrorLog logs/big.example.com-error.log
   CustomLog logs/big.example.com-access.log combined
   ```

8. Make sure to close the **VirtualHost** container (with a **</VirtualHost>** directive at the end of the container).
9. Add a **<Directory>** container to grant access to the directory that you have configured as the **DocumentRoot**:

   ```
   <Directory "/srv/vhost-big/www">
       Require all granted
   </Directory>
   ```

10. Repeat the process for the second website, making sure to set the second **ServerName** to **small.example.com**.
11. Close and save the configuration files with your changes.
12. Create any new directories that you configured with the **DocumentRoot** directives.
13. Create index.html text files in each directory defined by the associated new **DocumentRoot** directives. Don't worry about HTML code; a text file is fine for the purpose of this lab.
14. Make sure these domain names are configured in a local DNS server or in the /etc/hosts file. For example, if the Apache server is on a system with IP address 192.168.122.150 (such as tester1 .example.com), you could add the following lines to /etc/hosts:

    ```
    192.168.122.150 big.example.com
    192.168.122.150 small.example.com
    ```

    The same data should be included in the /etc/hosts file of a remote client system.

15. Use the **firewall-cmd** utility to allow HTTP data through the firewall:

```
# firewall-cmd --permanent --add-service=http
# firewall-cmd --reload
```

16. You may need to configure appropriate SELinux file contexts on the directory associated with the **DocumentRoot**. For example, if that directory is /vhost-big, one way to do so is with the following commands:

```
# semanage fcontext -a -t httpd_sys_content_t '/vhost-big(/.*)?'
# restorecon -R /vhost-big
```

    Note that the first command is not required if you have set the **DocumentRoot** to a directory that the SELinux policy marks with the httpd_sys_content_t type by default, such as /srv/vhost-big/www. In fact, the /srv/*/www directories have their default context type already set to httpd_sys_content_t. If in doubt, check the settings of the SELinux policy by running **semanage fcontext -l | grep httpd_sys_content_t**.

17. Make sure to run the **systemctl reload httpd** command to make Apache reread its configuration files, with the changes you've made.
18. Now you can test the results. Navigate to a remote system and try to access the newly created websites in the browser of your choice. If it works, the big.example.com and small.example.com domain names should display the index.html files created for each website.
19. If there are problems, check the syntax with the **httpd -t** and **httpd -S** commands. Check the log files in the /var/log/httpd directory.

Lab 4

This lab should be straightforward; when it is complete, you should find the following two files, which can be used to support a virtual host for a secure version of the big.example.com website:

```
/etc/pki/tls/certs/big.example.com.crt
/etc/pki/tls/private/big.example.com.key
```

Corresponding files for the small.example.com system should also now exist in these directories. The process is based on standard responses to the questions generated by the **genkey big.example.com** and **genkey small.example.com** commands.

Lab 5

The basics of this lab are straightforward. You'll need to repeat the same steps that you performed in Lab 3 and use the certificate and key files created in Lab 4. You can use the contents of the /etc/httpd/conf.d/ssl.conf file as a template. In addition, you should be concerned about the following:

1. While not absolutely required, you may want to set up the **DocumentRoot** in a directory different from a regular web server. Otherwise, the same web page will appear for both the regular and secure versions of a website.

2. It's a good practice to configure the **ErrorLog** and **CustomLog** with appropriate filenames to help identify that information is from the secure version of a given website.
3. It's helpful to copy the SSL directives from the template SSL virtual host in the ssl.conf file. All directives can apply to the secure versions of the big.example.com and small.example.com websites. You need at the minimum to set a **ServerName** directive, turn on **SSLEngine**, and set proper paths to **SSLCertificateFile** and **SSLCertificateKeyFile**:

```
ServerName big.example.com
SSLEngine On
SSLCertificateFile /etc/pki/tls/certs/big.example.com.crt
SSLCertificateKeyFile /etc/pki/tls/private/big.example.com.key
```

Of course, you'd substitute small.example.com for big.example.com for the noted directives in the secure virtual host container for that website. Don't forget to add the https service to the default zone on the firewall:

```
# firewall-cmd --permanent --add-service=https
# firewall-cmd --reload
```

Lab 6

In the default conf.d/userdir.conf file, the configuration of user home directories requires that you enable the **UserDir** directive. You can then customize the commented container associated with user home directories. If successful, only one user is allowed access to his home directory through the Apache web server from a client browser. In general, you may see directives such as the following within the container for the given home page:

```
AuthType Basic
AuthName "Just for one user"
AuthUserFile /etc/httpd/oneuser
Require user michael
```

As suggested in the chapter, the home directory should have regular executable permissions for other users, or at least for the user named apache, through ACLs. In addition, access won't be allowed unless you've set the httpd_enable_homedirs SELinux boolean. You'll also need to set up user michael in the authentication database for this directory with the **htpasswd -c /etc/httpd/oneuser michael** command.

Lab 7

The process required to set up a group-managed directory is a hybrid. The overall basic steps are as follows:

1. Create a regular user and group named techsupport. While not required, it can be helpful to configure that user with a higher UID and GID to avoid interfering with other future users and groups.
2. Make the other users a member of that group named techsupport.

3. Create a public_html/ subdirectory of the new user's home directory.
4. Set up appropriate permissions to support access by members of the techsupport group, normally 2770 permissions. The new techsupport's home and public_html directories should have either regular executable permissions by other users or executable permissions by the user named apache configured in an ACL.
5. Set up an index.html file in the public_html directory. It should be set with ownership by the techsupport group.
6. You will need to configure basic authentication for the group in Apache. Don't forget to set up the group in the authentication database for this directory with the **htpasswd** command.

Lab 8

The specified hello.pl script should include something like the following entries:

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
print "Hello World";
```

That script should be located in the directory specified by a **ScriptAlias /cgi-bin/** directive in the big.example.com virtual host container.

As an alternative configuration, you can create a new **<Directory>** block for the CGI directory. That container should include the **Options ExecCGI** and **AddHandler cgi-script .pl** directives. You may also need to disable the default **ScriptAlias** directive in httpd.conf. Although it's normally best to have scripts in a different directory than the **DirectoryRoot** tree configured for a virtual host, it's not required.

In addition, the permissions on the hello.pl file should be set to 755, and the SELinux contexts on the file (and directory) should be of the httpd_sys_script_exec_t file type. Of course, you'll have run an appropriate **semanage fcontext -a** command to make the change permanent. In any case, a successful result is as suggested in the lab question.