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 the Netcraft (www.netcraft.com) survey, which tracks the web servers associated with virtually every site on the Internet, Apache is currently used by more Internet web sites than all other web servers combined. Apache is included with RHEL 6.

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

INSIDE THE EXAM

This chapter directly addresses four 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 6. In general, regular and secure web sites are configured in different files. The objectives are to

- Configure a virtual host

Virtual hosts are the bread and butter of Apache, which supports the configuration of multiple web sites on the same server.

- Configure private directories

The private directory on an Apache web server is sort of halfway in between a regular and a secure web site. It's a directory that's accessible by one user.

- Configure group-managed content

Sometimes groups of users have to maintain the content of a web site 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 depend on scripts such as those associated with CGI. While you won't have to write a CGI script, you will have to set up Apache to support its deployment.

In addition, there are the standard requirements for all network services, discussed in Chapters 10 and 11. To review, 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

Apache is by far the most popular web server in use today. 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 (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 /var/www/manual directory.

Apache 2.2

As befits its reliability and stability, RHEL 6 includes an updated version of Apache 2.2. RHEL 5 also included a slightly older version of Apache 2.2. But no matter, the Apache 2.2 included with RHEL 6 has all of the updates needed to support the latest web pages, 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 integrated with other software components. The most common version of such is known as the LAMP stack, which refers to its components: Linux, Apache, MySQL, and one of three scripting languages (Perl, Python, or PHP).
There's no expectation from the RHCE objectives that you'll have to install a Structured Query Language database system such as MySQL (My Structured Query Language), or any scripting language. However, they do explain many of the modules available as part of the Apache configuration files. And of course, Apache works with other SQL databases along with other scripting languages.

Installation

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

```
# yum install httpd
```

But 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 groupinstall "Web Server"
```

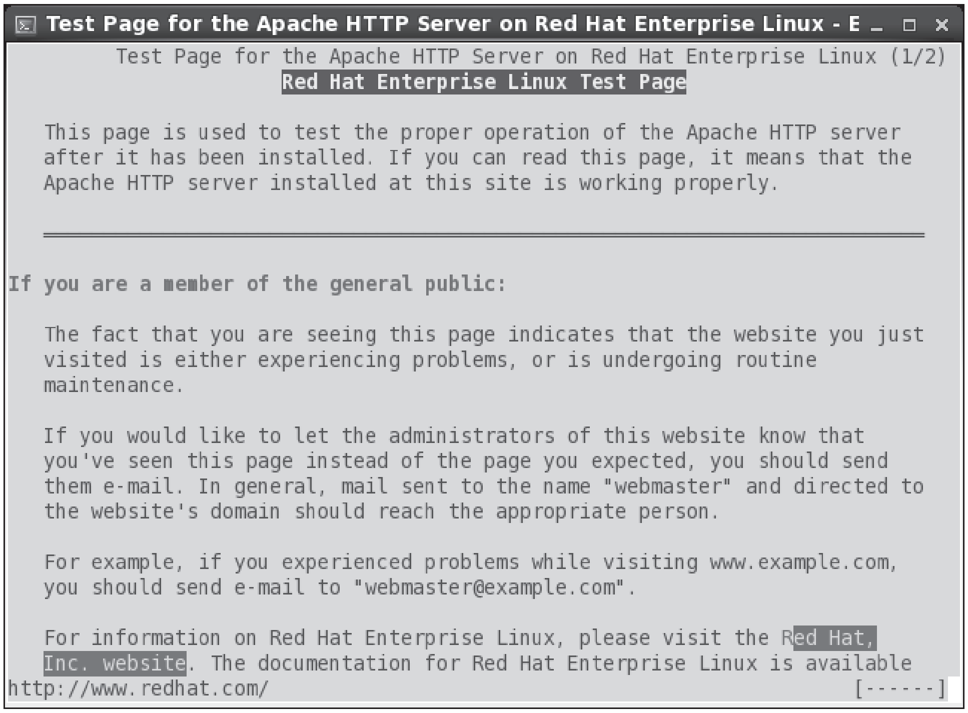
If you don't remember the names of available groups, run the **yum grouplist** command. The standard method to start Linux services is with a script in the /etc/init.d directory. It contains an **httpd** script. However, you can stop and start Apache, as well as reload the configuration file gracefully with the following commands:

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

No configuration is required for the most basic operation. 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.
The web page is based on the contents of the /etc/httpd/conf.d/welcome.conf file, which refers to the /var/www/error/noindex.html file for more information.

FIGURE 14-1

Default installed Apache home page



```
Test Page for the Apache HTTP Server on Red Hat Enterprise Linux - E
         Test Page for the Apache HTTP Server on Red Hat Enterprise Linux (1/2)
                    Red Hat Enterprise Linux Test Page

   This page is used to test the proper operation of the Apache HTTP server
   after it has been installed. If you can read this page, it means that the
   Apache HTTP server installed at this site is working properly.

   ___________________________________________________________________________

If you are a member of the general public:

   The fact that you are seeing this page indicates that the website you just
   visited is either experiencing problems, or is undergoing routine
   maintenance.

   If you would like to let the administrators of this website know that
   you've seen this page instead of the page you expected, you should send
   them e-mail. In general, mail sent to the name "webmaster" and directed to
   the website's domain should reach the appropriate person.

   For example, if you experienced problems while visiting www.example.com,
   you should send e-mail to "webmaster@example.com".

   For information on Red Hat Enterprise Linux, please visit the Red Hat,
   Inc. website. The documentation for Red Hat Enterprise Linux is available
http://www.redhat.com/                                                [------]
```

EXERCISE 14-1

Install the Apache Server

In this exercise, you'll install all of 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 Network or connect the system to the RHEL 6 (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 "Web Server" near the end of the list.

```
# yum groupinfo
```

3. You can install all default packages in the "Web Server" package group with the following command:

   ```
   # yum groupinstall "Web Server"
   ```

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

   ```
   # chkconfig --list httpd
   ```

5. Now use the following command to make sure Apache starts in runlevels 2, 3, 4, and 5 the next time Linux boots normally:

   ```
   # chkconfig httpd on
   ```

6. Start the Apache service with the following command:

   ```
   # apachectl start
   ```

7. If you haven't already done so in Chapter 2, install a text-based web browser. The RHEL 6 standard is **elinks**, which you can install with the following command:

   ```
   # yum install elinks
   ```

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

   ```
   # elinks 127.0.0.1
   ```

9. Review the result. Do you see the Apache test page?
10. Exit from the ELinks browser. Press `Q`, and when the Exit ELinks text menu appears, press `Y` to exit Elinks.
11. Back up the default httpd.conf configuration file; a logical location is your home directory.
12. 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.

But if you do get that message, install that package with the **yum httpd-manual** command.

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. There are other configuration files in two directories: /etc/httpd/conf and /etc/httpd/conf.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 directory.

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

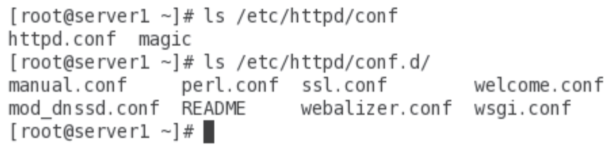
Apache comes with a well-commented set of default configuration files. In this section, you'll examine some key directives in the httpd.conf configuration file. Browse through this file in your favorite text editor or using a command pager such as **less**. Before beginning this analysis, remember that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

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

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

FIGURE 14-2

Apache configuration files



```
[root@server1 ~]# ls /etc/httpd/conf
httpd.conf  magic
[root@server1 ~]# ls /etc/httpd/conf.d/
manual.conf     perl.conf  ssl.conf        welcome.conf
mod_dnssd.conf  README     webalizer.conf  wsgi.conf
[root@server1 ~]#
```

the name of the directory, file, or module to be configured, contained in directional brackets (< >). Examples of this include

```
<Directory "/var/www/icons">
<Files ~ "^\.ht">
<IfModule mod_mime_magic.c>
```

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. I 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 is installed per Exercise 14-1, refer to http://localhost/manual/mod/quickreference.html. It'll provide detailed information on each directive. The default directives are summarized in the following three tables. Table 14-1 specifies directives associated with Section 1: Global Environment.

In all three tables, 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 **apachectl restart** to restart the Apache daemon or **apachectl graceful** to just reread the Apache configuration files.

Table 14-2 specifies directives associated with Section 2: Main Server Configuration.

TABLE 14-1

Global Environment Directives

Directiv	Description
ServerTokens	Specifies the response code at the bottom of error pages; options include OS, Prod, Major, Minor, Min, and Full.
ServerRoot	Sets the default directory; other directives are subdirectories.
PidFile	Names the file with the Process ID (and locks the service).
Timeout	Limits access time for both sent and received messages.
KeepAlive	Supports persistent connections.
MaxKeepAliveRequests	Limits requests during persistent connections (unless set to 0, which is no limit).
KeepAliveTimeout	Sets a time limit, in seconds, before a connection is closed.
StartServers	Adds child Apache processes; normally set to 8, which means 9 Apache processes run upon startup.
MinSpareServers	Specifies a minimum number of idle child servers.
MaxSpareServers	Specifies a maximum number of idle child servers; always at least +1 greater than MinSpareServers.
ServerLimit	Sets a limit on configurable processes; cannot exceed 20000.
MaxClients	Limits the number of simultaneous requests; other requests to the server just have to wait.
MaxRequestsPerChild	Limits the requests per child server process.
MinSpareThreads	Specifies the minimum number of spare threads to handle additional requests.
MaxSpareThreads	Specifies the maximum number of available idle threads to handle additional requests.
ThreadsPerChild	Sets the number of threads per child server process.
Listen	Specifies a port and possibly an IP address (for multihomed systems) to listen for requests.
LoadModule	Loads various modular components, such as authentication, user tracking, executable files, and more.
Include	Adds the content of other configuration files.
User	Specifies the username run by Apache on the local system.
Group	Specifies the group name run by Apache on the local system.

TABLE 14-2 Main Server Configuration Directives

Directiv	Description
ServerAdmin	Sets the administrative e-mail address; may be shown (or linked to) on default error pages.
UseCanonicalName	Supports the use of ServerName as the referenced URL.
DocumentRoot	Assigns the root directory for web site files.
Options	Specifies features associated with web directories, such as ExecCGI, FollowSymLinks, Includes, Indexes, MultiViews, and SymLinksIfOwnerMatch.
AllowOverride	Supports overriding of previous directives from .htaccess files.
Order	Sets the sequence for evaluating Allow and Deny directives.
Allow	Configures host computers that are allowed access.
Deny	Configures host computers that are denied access.
UserDir	Specifies location of user directories; can be set to enable or disable for all or specified users.
DirectoryIndex	Specifies files to look for when navigating to a directory; set to index.html by default.
AccessFileName	Sets a filename within a directory for more directives; normally looks for .htaccess.
Satisfy	Specifies result when both user and host restrictions are used; may be set to Any or All.
TypesConfig	Locates mime.types, which specifies file types associated with extensions.
DefaultType	Sets a default file type if not found in mime.types.
MIMEMagicFile	Normally looks to /etc/httpd/conf/magic to look inside a file for its MIME type.
HostNameLookups	Requires URL lookups for IP addresses; results are logged.
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, in a different format, with a location relative to ServerRoot.
ServerSignature	Adds a list with server version and possibly ServerAdmin e-mail address to error pages and file lists; can be set to On, OFF, or EMail.
Alias	Configures a directory location; similar to a soft link.
DAVLockDB	Specifies the path to the lock file for the WebDAV (Web-based Distributed Authoring and Versioning) database.
ScriptAlias	Similar to Alias; for scripts.
IndexOptions	Specifies how files are listed from a DirectoryIndex.

TABLE 14-2 Main Server Configuration Directives (*continued*)

Directiv	Description
AddIconByEncoding	Assigns an icon for a file by MIME encoding.
AddIconByType	Assigns an icon for a file by MIME type.
AddIcon	Assigns an icon for a file by extension.
DefaultIcon	Sets a default icon for files not otherwise configured.
ReadmeName	Configures a location for a README file to go with a directory list.
HeaderName	Configures a location for a HEADER file to go with a directory list.
IndexIgnore	Adds files that are not included in a directory list.
AddLanguage	Assigns a language for filename extensions.
LanguagePriority	Sets a priority of languages if not configured in client browsers.
ForceLanguagePriority	Specifies action if a web page in the preferred language is not found.
AddDefaultCharset	Sets a default character set; you may need to change it for different languages.
AddType	Maps filename extensions to a specified content type.
AddHandler	Maps filename extensions to a specified handler; commonly used for scripts or multiple languages.
AddOutputFilter	Maps filename extensions to a specified filter.
BrowserMatch	Customizes responses to different browser clients.

Table 14-3 specifies directives associated with Section 3: Virtual Hosts. While virtual host directives are disabled by default, I include those directives in the commented example near the end of the default httpd.conf file. While these directives were already used in other sections, you can—and generally should—customize them for individual virtual hosts to support different web sites on the same Apache server. In many cases, each virtual host stanza will include directives from the main part of the httpd.conf configuration file, customized for that virtual host.

Basic Apache Configuration for a Simple Web Server

As described earlier, 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.

TABLE 14-3

Virtual Host Configuration Directives

Directiv	Description
NameVirtualHost	Specifies an IP address and port number for multiple virtual hosts.
ServerAdmin	Assigns an e-mail address for the specified virtual host.
DocumentRoot	Sets a root directory for the virtual host.
ServerName	Names the URL for the virtual host.
ErrorLog	Creates an error log; the location is based on the DocumentRoot.
CustomLog	Creates an custom log; the location is based on the DocumentRoot.

The default **DirectoryIndex** directive looks for an index.html web page file in this directory. A standard RHEL 6 index.html page is available in the /usr/share/doc/HTML/en-US directory. Copy that file to the /var/www/html directory, and navigate to http://localhost with a browser such as ELinks.

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 14-1 confirms that the main Apache configuration files are stored in the conf/ and conf.d/ subdirectories of the **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 in the /etc/httpd/logs directory, they're actually stored in the /var/log/httpd directory. 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 configured, how secure web sites 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 set up different sets of log files for different web sites. To that end, you should set up different log files for the secure versions of a web site. The traffic on a web site is important when choosing a log rotation frequency.

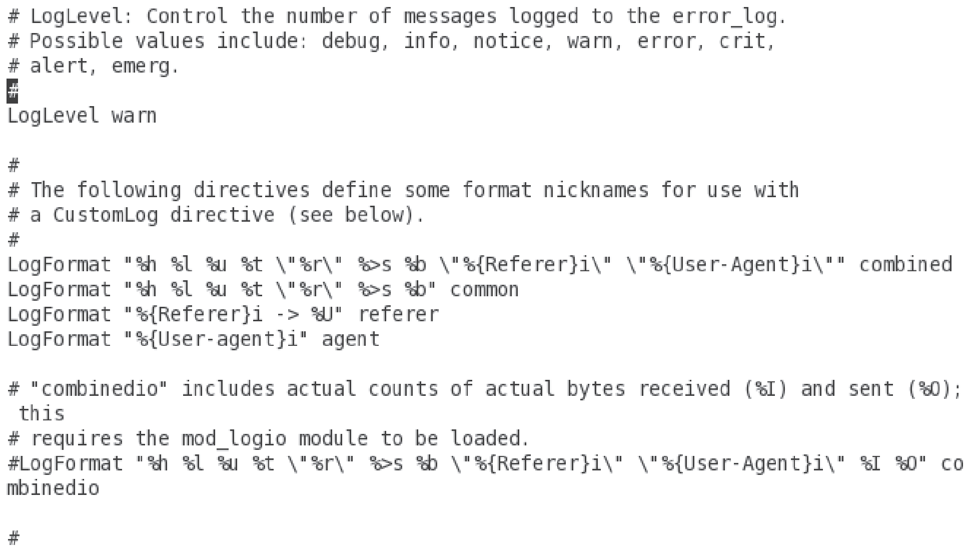
There are standard Apache log file formats. For more information, take a look at the **LogFormat** directive in Figure 14-3. Four different formats are shown: combined, common, the referrer (the web page with the link used to get to your site), and the agent (the user's web browser). The first two **LogFormat** lines include a number of percent signs followed by lowercase letters. These directives determine what goes into the log.

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 Stats) requires the combined log format. It will fail to run if you leave the default common format. Awstats is a great tool for graphically displaying site activity. You can download it from a site such as www.sourceforge.net.

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

#
# 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
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent

# "combinedio" includes actual counts of actual bytes received (%I) and sent (%O);
 this
# requires the mod_logio module to be loaded.
#LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" co
mbinedio

#
```

CERTIFICATION OBJECTIVE 14.02

Standard Apache Security Configuration

You can configure several layers of security for the Apache web server. Firewalls based on the **iptables** 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 web sites can encrypt communication. If there is a problem, SELinux can limit the risks.

Ports and Firewalls

With the **Listen** and **NameVirtualHost** 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 **iptables** rules accordingly.

If you just open these ports indiscriminately, it allows traffic from all systems. It may be appropriate to set up a custom rule to limit access to one or more systems or networks. For example, the following custom rules allows access to every system on the 192.168.122.0 network except the one with IP address 192.168.122.150, over port 80. To review, these rules are applied to the **iptables** command, in order.

```
-A INPUT -m state --state NEW -m tcp -p tcp -s 192.168.122.150
--dport 80 -j REJECT
-A INPUT -m state --state NEW -m tcp -p tcp -s 192.168.122.0/24
--dport 80 -j ACCEPT
```

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, as they relate to a service, 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 14-4.

The first five are just the default SELinux contexts for standard directories. For web sites 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.

Create a Special Web Directory

In many cases, you'll create dedicated directories for each virtual web site. It's better to segregate the files for each web site in its 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 directories, the user context is system_u and the type is http_sys_content_t. For a newly created /www directory, you could just change the SELinux contexts with the following commands. The **-R** applies the changes recursively, so the new contexts are applied to files and subdirectories.

```
# chcon -R -u system_u /www/
# chcon -R -t httpd_sys_content_t /www/
```

Of course, if scripts are required for the associated web site, you'll want to run the following command to make sure the SELinux changes survive a relabel:

```
# semanage fcontext -a -s system_u -t httpd_sys_content_t /www/
```

TABLE 14-4

SELinux File Contexts

Directory	SELinux Context Type
/etc/httpd, /etc/httpd/conf, /etc/httpd/conf.d, /var/run/httpd	httpd_config_t
/usr/lib64/httpd/modules	httpd_modules_t
/var/log/httpd	httpd_log_t
/var/www, /var/www/error, /var/www/html, /var/www/icons, /var/www/manual, /var/www/usage	httpd_sys_content_t
/var/www/cgi-bin	httpd_sys_script_exec_t
n/a	httpd_sys_content_rw_t
n/a	httpd_sys_content_ra_t

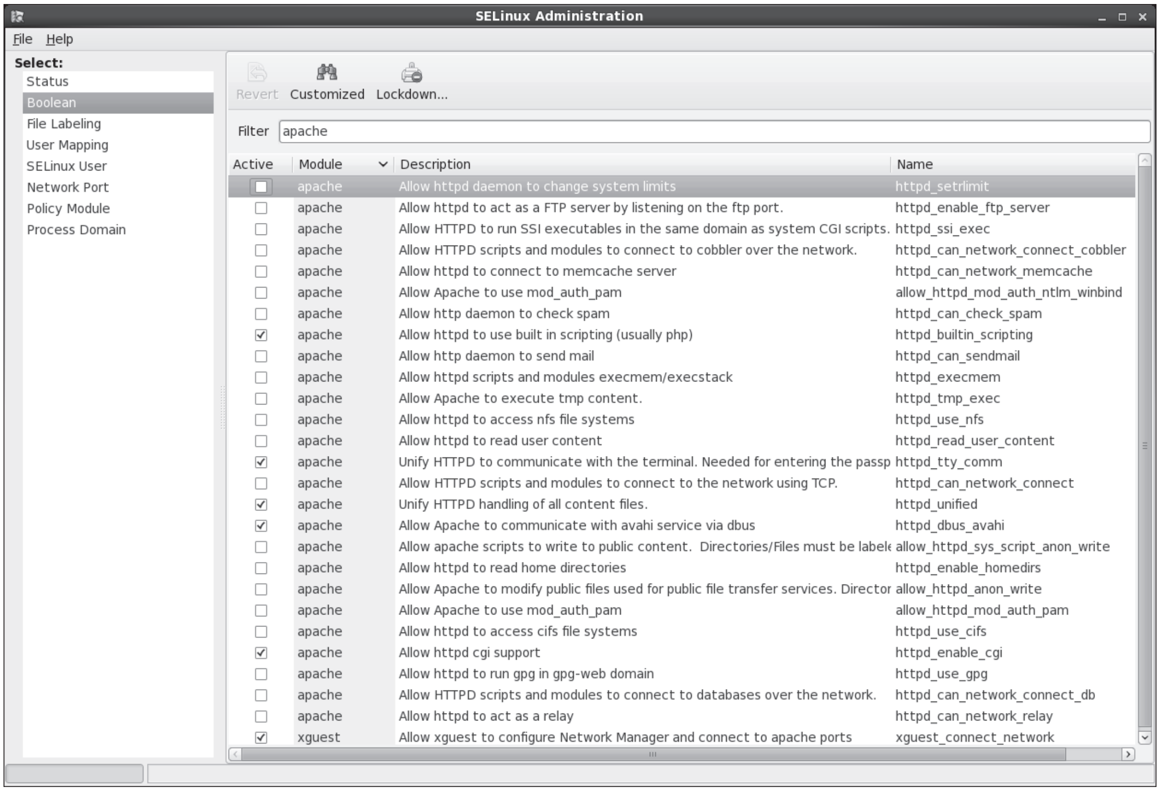
This command creates a file_contexts.local file in the /etc/selinux/targeted/contexts/files directory. If there's also a cgi-bin/ subdirectory, you'll want to set up appropriate contexts for that subdirectory as well with the following command:

```
# semanage fcontext -a -s system_u -t httpd_sys_script_exec_t \
/www/cgi-bin/
```

Apache and SELinux Boolean Settings

Boolean settings are more extensive. For display purposes, I'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 14-5.

FIGURE 14-4 Apache-related SELinux boolean settings



Filter apache

Active	Module	Description	Name
☐	apache	Allow httpd daemon to change system limits	httpd_setrlimit
☐	apache	Allow httpd to act as a FTP server by listening on the ftp port.	httpd_enable_ftp_server
☐	apache	Allow HTTPD to run SSI executables in the same domain as system CGI scripts.	httpd_ssi_exec
☐	apache	Allow HTTPD scripts and modules to connect to cobbler over the network.	httpd_can_network_connect_cobbler
☐	apache	Allow httpd to connect to memcache server	httpd_can_network_memcache
☐	apache	Allow Apache to use mod_auth_pam	allow_httpd_mod_auth_ntlm_winbind
☐	apache	Allow http daemon to check spam	httpd_can_check_spam
✓	apache	Allow httpd to use built in scripting (usually php)	httpd_builtin_scripting
☐	apache	Allow http daemon to send mail	httpd_can_sendmail
☐	apache	Allow httpd scripts and modules execmem/execstack	httpd_execmem
☐	apache	Allow Apache to execute tmp content.	httpd_tmp_exec
☐	apache	Allow httpd to access nfs file systems	httpd_use_nfs
☐	apache	Allow httpd to read user content	httpd_read_user_content
✓	apache	Unify HTTPD to communicate with the terminal. Needed for entering the passp	httpd_tty_comm
☐	apache	Allow HTTPD scripts and modules to connect to the network using TCP.	httpd_can_network_connect
✓	apache	Unify HTTPD handling of all content files.	httpd_unified
✓	apache	Allow Apache to communicate with avahi service via dbus	httpd_dbus_avahi
☐	apache	Allow apache scripts to write to public content. Directories/Files must be labele	allow_httpd_sys_script_anon_write
☐	apache	Allow httpd to read home directories	httpd_enable_homedirs
☐	apache	Allow Apache to modify public files used for public file transfer services. Director	allow_httpd_anon_write
☐	apache	Allow Apache to use mod_auth_pam	allow_httpd_mod_auth_pam
☐	apache	Allow httpd to access cifs file systems	httpd_use_cifs
✓	apache	Allow httpd cgi support	httpd_enable_cgi
☐	apache	Allow httpd to run gpg in gpg-web domain	httpd_use_gpg
☐	apache	Allow HTTPD scripts and modules to connect to databases over the network.	httpd_can_network_connect_db
☐	apache	Allow httpd to act as a relay	httpd_can_network_relay
✓	xguest	Allow xguest to configure Network Manager and connect to apache ports	xguest_connect_network

TABLE 14-5

Default Active Apache-Related SELinux Boolean Settings

Active Boolean	Description
httpd_builtin_scripting	Provides permissions to scripts in httpd_t labeled directories; sometimes used for PHP content.
httpd_dbus_avahi	Supports access from HTTP services to automated IP address configuration.
httpd_enable_cgi	Allows HTTP services to execute GCI scripts, labeled with the httpd_sys_script_exec_t type.
httpd_tty_comm	Enables communication with controlling terminals; useful for SSL certificates.
httpd_unified	Supports full read/write/execute access by all httpd_t files.
xguest_connect_network	Allows access from secured guests.

Out of the many other SELinux options, the one of particular interest for this chapter is **httpd_enable_homedirs**, which supports access to files from 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 unenabled SELinux Apache-related options from Figure 14-4 are summarized in Table 14-6. 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 web sites 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 (for 32-bit systems, the directory is /usr/lib/httpd/modules). But available modules aren't used unless they're loaded with the LoadModule directive in appropriate Apache configuration files.

TABLE 14-6 Default Inactive Apache-Related SELinux Boolean Settings

Inactive Boolean	Description
allow_httpd_anon_write	Allows the web server to write to files labeled with the public_content_rw_t file type.
allow_httpd_mod_auth_ntlm_winbind	Supports access to Microsoft authentication databases, if the mod_auth_ntlm_winbind module is loaded.
allow_httpd_mod_auth_pam	Enables access to PAM authentication modules, if the mod_auth_pam module is loaded.
allow_httpd_sys_script_anon_write	Configures write access by scripts 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	Supports Apache access to connections on remote ports; normally disabled to minimize risk of attacks on other systems.
httpd_can_network_connect_cobbler	Allows Apache to connect to the Cobbler installation server; should not be activated simultaneously with httpd_can-network_connect.
httpd_can_network_connect_db	Allows Apache to connect to a database server.
httpd_can_network_memcache	Enables HTTP memory caching access; originally set up for a translation server.
httpd_can_network_relay	Supports the use of the HTTP service as a proxy.
httpd_can_sendmail	Allows the use of HTTP-based e-mail services; does not require the use of the sendmail SMTP server described in Chapter 13.
httpd_enable_homedirs	Configures access via HTTP to files in user home directories.
httpd_execmem	Supports operation of executable programs that require executable and writable memory addresses; normally disabled to minimize risk of buffer overflows.
httpd_read_user_content	Enables access to scripts from user home directories.
httpd_setrlimit	Allows Apache to modify maximum number of file descriptors.
httpd_ssi_exec	Allows Apache to access Server Side Include (SSI) scripts; similar to httpd_enable_cgi.
httpd_tmp_exec	Supports those Apache-based scripts that require access to the /tmp directory.
httpd_use_cifs	Enables Apache access to shared Samba directories, when labeled with the cifs_t file type.
httpd_use_gpg	Allows access to systems that require GPG encryption.
httpd_use_nfs	Enables Apache access to shared Samba directories, when labeled with the nfs_t file type.

Security Within Apache

You've read about (and hopefully tested) Apache security options related to **iptables**-based firewalls 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 web sites on the server. To explore the basics of Apache security, start with the first default active line in httpd.conf:

```
ServerTokens OS
```

This line looks deceptively simple; it limits the information displayed about a web server you navigate to a nonexistent page to the following message:

```
Apache/2.2.15 (Red Hat) Server at localhost Port 80
```

Contrast that output with what happens with a **ServerTokens Full** line:

```
Apache/2.2.15 (Red Hat) DAV/2 mod_ssl/2.2.15 OpenSSL/1.0.0-fips mod_wsgi/3.2
Python/2.6.5 mod_perl/2.0.4 Perl/v5.10.1 Server at localhost Port 80
```

In other words, with one option, outsiders can see whether modules such as Perl, Python, and PHP have been loaded, along with their version numbers. As not everyone updates their software in a perfectly timely manner, what happens when a cracker sees a version that has been compromised, your servers will face additional risks.

Next, you can restrict access to the directory defined by the **ServerRoot** directive as shown here:

```
<Directory />
     Options FollowSymLinks
     AllowOverride None
</Directory>
```

This configures a very restrictive set of permissions. The **Options FollowSymLinks** line supports the use of symbolic links for web pages. The **AllowOverride None** line disables any .htaccess files. The ServerRoot directive points to /etc/httpd, which contains Apache configuration files. Without the **AllowOverride None** line, a cracker who inserts a malicious .htaccess file can configure permissions that allows any user to change such configuration files.

However, there's an appropriate use for .htaccess files. For example, when placed in a subdirectory such as /www/html/project, then it can be used to permit access to a group, and such changes would apply only to that directory.

You can improve this by limiting access to all but explicitly allowed users, by adding the following commands to the desired **<Directory>** container:

```
Order deny,allow
Deny from all
```

The next **<Directory>** container example limits access to /var/www/html, which corresponds to the default **DocumentRoot** directive (while these directives are divided by numerous comments, they are all in the same stanza):

```
<Directory /var/www/html>
     Options Indexes FollowSymLinks
     AllowOverride None
     Order allow,deny
     Allow from all
</Directory>
```

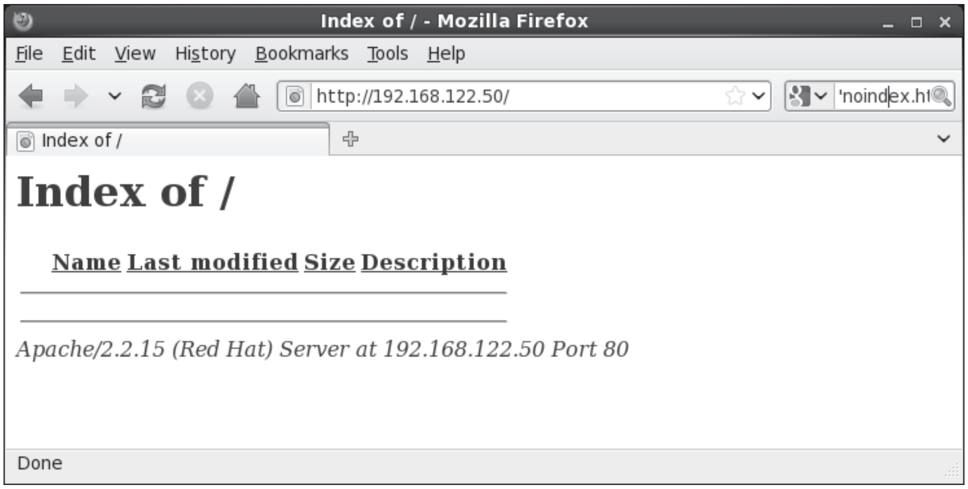
The **Options** directive is different; the **Indexes** setting allows readers to see a list of files on the web server if no index.html file is present in the specified directory. The **Order** and **Allow** lines allow all users to access the web pages on this server.

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 web site associated with the Apache server, the page shown in Figure 14-6 appears. For more information on how that worked, see Exercise 14-2.

FIGURE 14-5

Browse to an index of files


Index of / - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.122.50/
'noindex.ht
Index of /
Done

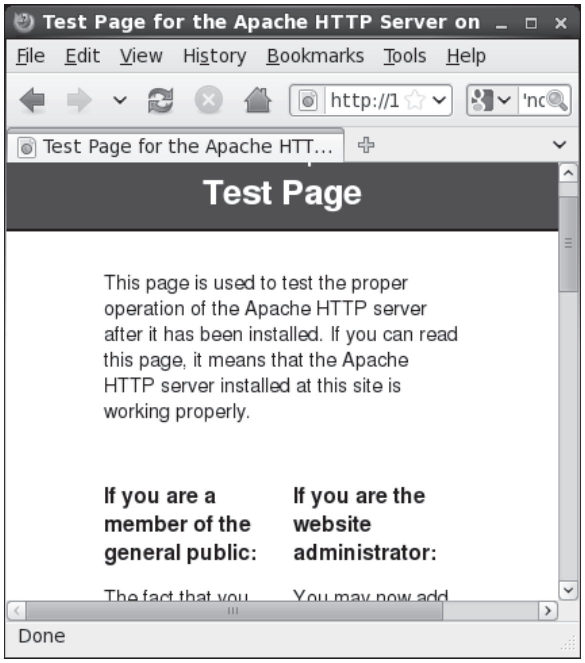
Index of /

Name Last modified Size Description

Apache/2.2.15 (Red Hat) Server at 192.168.122.50 Port 80

FIGURE 14-6

Browse to the default Apache test page


Test Page for the Apache HTTP Server on
File Edit View History Bookmarks Tools Help
http://1
Test Page for the Apache HTT...
Done

Test Page

This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page, it means that the Apache HTTP server installed at this site is working properly.

If you are a member of the general public:

If you are the website administrator:

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 computer that requests a web page from any of the IP addresses for your computer 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 uses to limit access to one specific IP address. For example, if a system has two network cards with IP addresses 192.168.0.200 and 192.168.122.1, the following directive can help limit access to systems on the 192.168.122.0 network:

```
Listen 192.168.122.1:80
```

For secure web sites, 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

exam watch

The RHCE objectives suggests that you need to be ready to configure regular HTTP and secure HTTPS web sites.

the following directive, which points to the default secure HTTP (HTTPS) port for TCP/IP, 443:

```
Listen 443
```

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:

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

2. Find the **ServerRoot** directive in the same file. It should read as follows:

   ```
   ServerRoot /etc/httpd
   ```

   In addition, make a note of the **Alias /error/ "/var/www/error/"** line. You'll need that information shortly.
3. Put the effect of the two directives together. In other words, the **Include 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.
4. Navigate to the /etc/httpd/conf.d directory. Open the welcome.conf file.
5. Note the **ErrorDocument** page. While it points to the /error/noindex.html file, that's based on the aforementioned **Alias** directive. In other words, you should be able to find the noindex.html file in the /var/www/error directory.
6. Take a look at the /var/www/error/noindex.html file. To open it up in the ELinks browser, run the **elinks /var/www/error/noindex.html** command. The web page that appears should now be familiar.
7. Exit from the browser. Move the welcome.conf file from the /etc/httpd/conf directory to a backup location.
8. Restart the Apache service with the **apachectl restart** command.

9. Navigate to the localhost system with **elinks http://127.0.0.1** command. What do you see?
10. Open a second terminal, navigate to the /var/www/html directory, and run the **touch test1 test2 test3 test4** command.
11. Reload the browser in the original terminal. What do you see?
12. 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, create a subdirectory of **DocumentRoot**, fill it with several files, set up appropriate security contexts, and activate Apache.

1. Make sure the firewall does not block access to port 80. One way to do so is with the Red Hat Firewall Configuration tool, which you can start with the **system-config-firewall** command. In the Trusted Services section, make sure to allow incoming WWW (HTTP) connections. Alternatively, you could use the directions specified earlier for a custom firewall.
2. Create a subdirectory of **DocumentRoot**. In the default /etc/httpd/conf/httpd.conf file, it's /var/www/html. For this exercise, I've created the /var/www/html/help directory.
3. Copy the files from the /var/www/manual directory:

   ```
   # cp -ar /var/www/manual/* /var/www/html/help/
   ```

4. Restart the Apache service with the following command:

   ```
   # apachectl restart
   ```

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

   ```
   # chkconfig httpd on
   ```

6. Use the **ls -Z /var/www/html** and **ls -Z /var/www/html/help** commands to review the security contexts for the /var/www/html/sharing directory and

copied files. If it doesn't already correspond to the contexts shown here, set them up with the following commands:

```
# chcon -R -u system_u /var/www/html/sharing/
# chcon -R -t httpd_sys_content_t /var/www/html/sharing/
```

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 host names or IP addresses. This following standard command sequence allows access by default. It reads the **deny** directive first:

```
Order deny,allow
```

exam watch

If you set Order allow,deny, access is denied by default. Only those host names or IP addresses associated with the allow directive are allowed access.

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

```
Deny from osborne.com
```

If DNS service is unreliable, you may prefer to use IP addresses. The following example directives use 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 web sites configured on the Apache server to authorized users with passwords. As described shortly, these passwords can be different from the regular authentication database.

For example, to configure user-based security for the web site 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:

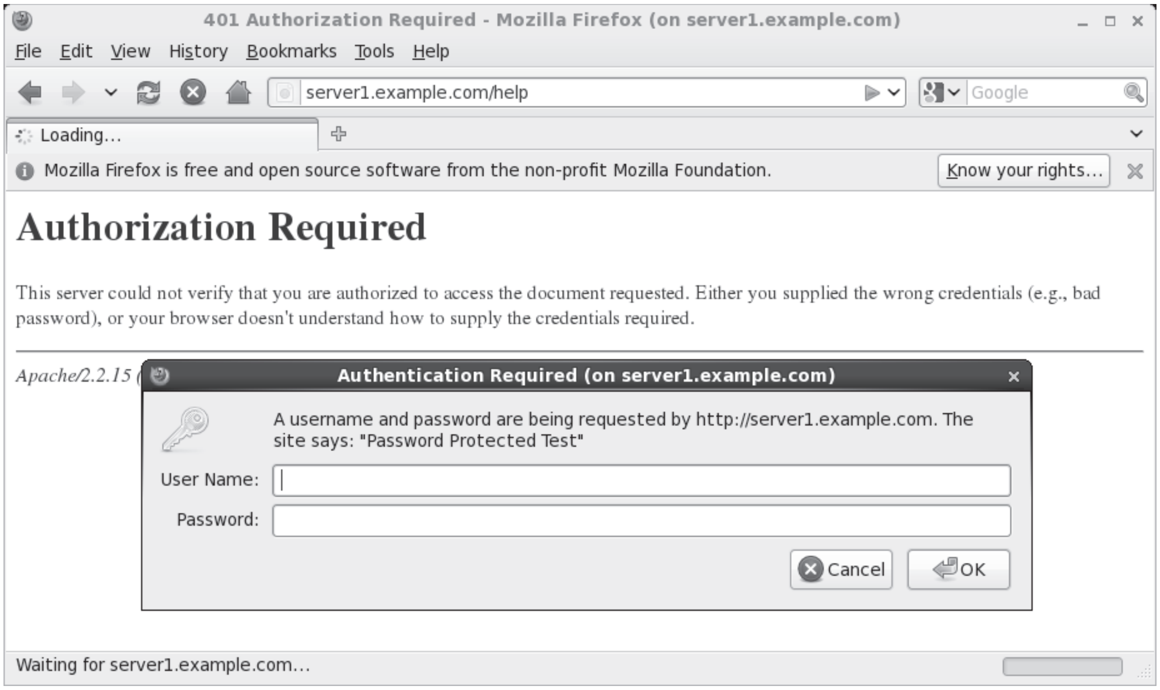
- To set up basic authentication, you'll need an **AuthType Basic** directive.
- To describe the site to requesting users, you can include an **AuthName "*some comment*"** directive.
- To refer to a web server password database named /etc/httpd/testpass, you'll need a **AuthUserFile /etc/httpd/testpass** 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.

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

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

When properly configured, the next time you try accessing the http://server1.example.com/help web site directory in the Firefox browser, you're prompted for a username and password, as shown in Figure 14-7.

FIGURE 14-7 Password protection for a web site


401 Authorization Required - Mozilla Firefox (on server1.example.com)
File Edit View History Bookmarks Tools Help
server1.example.com/help
Google
Loading...
Know your rights...

Mozilla Firefox is free and open source software from the non-profit Mozilla Foundation.

Authorization Required

This server could not verify that you are authorized to access the document requested. Either you supplied the wrong credentials (e.g., bad password), or your browser doesn't understand how to supply the credentials required.

Apache/2.2.15 (

Authentication Required (on server1.example.com)

A username and password are being requested by http://server1.example.com. The site says: "Password Protected Test"

User Name:

Password:

Waiting for server1.example.com...

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 appropriate for a private directory is the home directory example included in the httpd.conf file. With the right options, such directories can also be managed by members of a group.

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

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, as 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 permissions. The .htaccess file can be stored in any web directory, labeled with the httpd_config_t SELinux context.

Password-Protected Access

To configure passwords for a web site, 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'll also need 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 httpd.conf file includes commented suggestions that can enable access to user home directories. One useful option through Apache 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 cracker might not be able to directly read the contents of the noted directories, if he sees a script through the resulting web site, he'd be able to execute that script as any logged-in user.

There's an alternative, if the filesystem with the /home directory has been mounted with the Access Control Lists discussed in Chapter 4. You could create ACLs on the noted directories specifically for the user named apache. For user michael and his home directory, that would work with 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. The commented stanza that follows is one excellent way to help keep home directories so 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 htttpd_enable_homedirs 1** command.

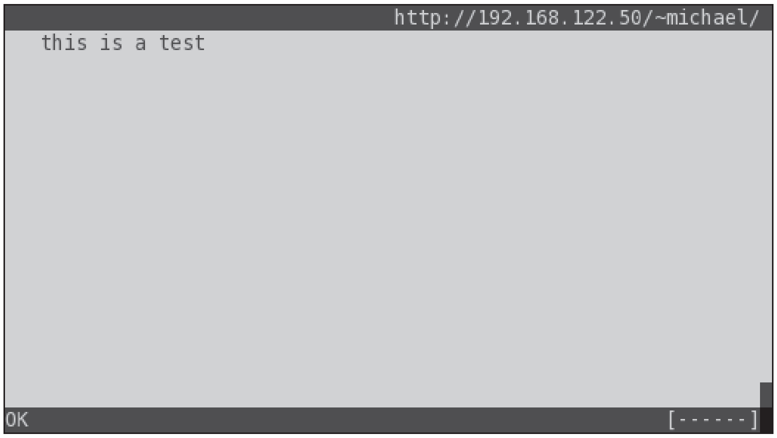
At that point, a web server that's directed to user michael's directory can read an index.html file in the public.html subdirectory. Figure 14-8 illustrates the result, where the noted text is the only contents of index.html.

Of course, additional changes are suggested in the httpd.conf file. If you activate the stanza that starts with the <Directory /home/*/public_html> container, it supports additional levels of access to the public_html subdirectory of all user's home directories.

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

FIGURE 14-8

View the index.html file for user michael



```
                                        http://192.168.122.50/~michael/
   this is a test
OK                                                            [------]
```

The AllowOverride directive supports information access by document type (FileInfo), access associated with authorization directives (AuthConfig), and access secured by directives such as Allow, Deny, and Order.

```
#    AllowOverride FileInfo AuthConfig Limit
```

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 activates symbolic links 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 <Limit> container limits access and write options as shown; however, the default is to allow from all users.

```
#    <Limit GET POST OPTIONS>
#        Order allow,deny
#        Allow from all
#    </Limit>
```

In contrast, the <LimitExcept> container shown here denies nothing.

```
#    <LimitExcept GET POST OPTIONS>
#        Order deny,allow
#        Deny from all
#    </LimitExcept>
#</Directory>
```

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 for a shared Apache directory is unrelated to the shadow password suite.

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 for the user named apache for the execute bit for all users. That assumes the filesystem with the /home directory has been mounted with ACLs. 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 re-read the file.

You will have a chance to set this up in one of the chapter labs, and perhaps more (hint, hint!).

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 use of the **AuthType Basic**, **AuthName**, and **AuthUserFile** directives. This will be done with the standard Apache web site; 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 stanza **<Directory "/var/www/html">**.Create a new stanza for a **DocumentRoot** subdirectory. One option is the /var/www/html/chapter directory. In the default version of httpd.conf, it's just before the commented options for the **UserDir** directive. The first and last directives in the stanza would look like

```
<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:

```
# apachectl restart
```

If you're concerned about currently connected users, make Apache reread the configuration file, without disconnections, with the **service httpd reload** command.

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 my system, I created a web password for user michael in the noted file with the following command:

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

   Additional users can be added without 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 web sites on a single IP address. In a world where there are no more new IPv4 addresses available, that can be useful. To do so, you can configure virtual hosts for regular web sites in the main Apache configuration file, /etc/httpd/conf/httpd.conf. 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.

on the job

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 also create multiple secure web sites accessible through the HTTPS protocol in the /etc/httpd/conf.d/ssl.conf configuration file. 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 web sites created in this chapter, there are several things to keep in mind.

- Make sure the /etc/hosts file of the client system includes the IP address with the specified fully qualified domain names (FQDNs). Duplicate IP addresses with different FQDNs are normal.
- Open the /etc/elinks.conf configuration file, and comment out the two standard directives in that file.
- To access a regular web site, make sure to include the protocol in front of the FQDN, such as http://vhost1.example.com or https://vhost2.example.com.

exam watch

Be prepared to create multiple web sites on an Apache web server using virtual hosts. It's best to create a separate *VirtualHost* ***containers for this purpose.***

The beauty of **VirtualHost** containers is that you can copy virtually the same stanza to create as many web sites 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 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.

The Standard Virtual Host

As described earlier, Section 3 of the default httpd.conf includes sample commands that can be used to create one or more virtual hosts. To activate the virtual host feature, the first step is to activate this directive:

```
#NameVirtualHost *:80
```

To use a name-based host, leave the asterisk after this directive. Otherwise, set the IP address for the local network interface. It's often more reliable to substitute the IP address, as it avoids the delays sometimes associated with name resolution through a DNS server. However, you may need to create multiple name-based virtual hosts as well. Normally, I leave the asterisk in place.

exam watch

To create a virtual host for a secure web site, you'll need a second *NameVirtualHost* ***directive for the HTTPS port, 443. I normally add that directive to the ssl.conf file in the /etc/httpd/conf.d directory, which is incorporated into Apache configuration through the*** *Include conf.d/*.conf* ***directive. Options for creating a secure virtual host are described later in this chapter.***

You should already know that TCP/IP port 80 is the default for serving web pages. To direct all requests on this server via IP address 192.168.122.50 on port 80, you could substitute **<VirtualHost 192.168.122.50:80>** for the first line. But in most cases, you should leave that directive as is, to preserve the use of the same IP address for different web sites. (It also makes it possible for DHCP to work on that web server, but that's a more complex issue.)

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

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

- Error messages are sent to the e-mail address defined by **ServerAdmin**.
- The web pages can be stored in the **DocumentRoot** directory. Make sure the SELinux security contexts of any **DocumentRoot** directory you create is consistent with the contexts of the default /var/www directory (and subdirectories). Apply the **chcon** and **semanage fcontext -a** commands as required to make the security contexts match.
- Clients can call this web site through the **ServerName**.

- The **ErrorLog** and **CustomLog** directives specify a *relative* log directory, relative to the **ServerRoot**. Unless you've created a different **ServerRoot** for this virtual host, 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 stanza, 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 web site. 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, I've set up two virtual hosts with the following stanzas:

```
<VirtualHost *:80>
    ServerAdmin webmaster@vhost1.example.com
    DocumentRoot /www/docs/vhost1.example.com
    ServerName vhost1.example.com
    ErrorLog logs/vhost1.example.com-error_log
    CustomLog logs/vhost1.example.com-access_log common
</VirtualHost>
<VirtualHost *:80>
    ServerAdmin webmaster@vhost2.example.com
    DocumentRoot /www/docs/vhost2.example.com
    ServerName vhost2.example.com
    ErrorLog logs/vhost2.example.com-error_log
    CustomLog logs/vhost2.example.com-access_log common
</VirtualHost>
```

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, I see the following output from one of my server1.example.com RHEL 6 systems:

```
VirtualHost configuration:
wildcard NameVirtualHosts and _default_ servers:
*:80            is a NameVirtualHost
  default server vhost1.example.com (/etc/httpd/conf/httpd.conf:1010)
  port 80 namevhost vhost1.example.com(/etc/httpd/conf/httpd.conf:1010)
  port 80 namevhost vhost2.example.com(/etc/httpd/conf/httpd.conf:1017)
*:443            is a NameVirtualHost
  default server vhost1.example.com (/etc/httpd/conf.d/ssl.conf:75)
  port 80 namevhost vhost1.example.com(/etc/httpd/conf.d/ssl.conf:75)
  port 80 namevhost vhost2.example.com(/etc/httpd/conf.d/ssl.conf:105)
Syntax OK
```

If a "using 127.0.0.1 for ServerName" error appears, you haven't assigned a value for the **ServerName** directive. In addition, you may want to set up the /etc/hosts file, or a DNS server for the local network, with the IP addresses for domain names like vhost1.example.com and vhost2.example.com.

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 RPM. Before editing this file, back it up. The first active directive loads the SSL module:

```
LoadModule ssl_module modules/mod_ssl.so
```

Make sure the following **Listen** directive is active:

```
Listen 443
```

As suggested by the title, this configuration file includes a number of passphrase dialogues. Generally, no changes are required to these directives:

```
SSLPassPhraseDialog builtin
SSLSessionCache         shmcb:/var/cache/mod_ssl/scache(512000)
SSLSessionCacheTimeout  300
SSLMutex default
SSLRandomSeed startup file:/dev/urandom  256
SSLRandomSeed connect builtin
SSLCryptoDevice builtin
```

Before the virtual host containers, you'll need to include a **NameVirtualHost** directive for Port 443 before you can configure multiple virtual hosts in this file. It's the same directive used in the main Apache configuration file, just pointed to the standard HTTPS port:

```
NameVirtualHost *:443
```

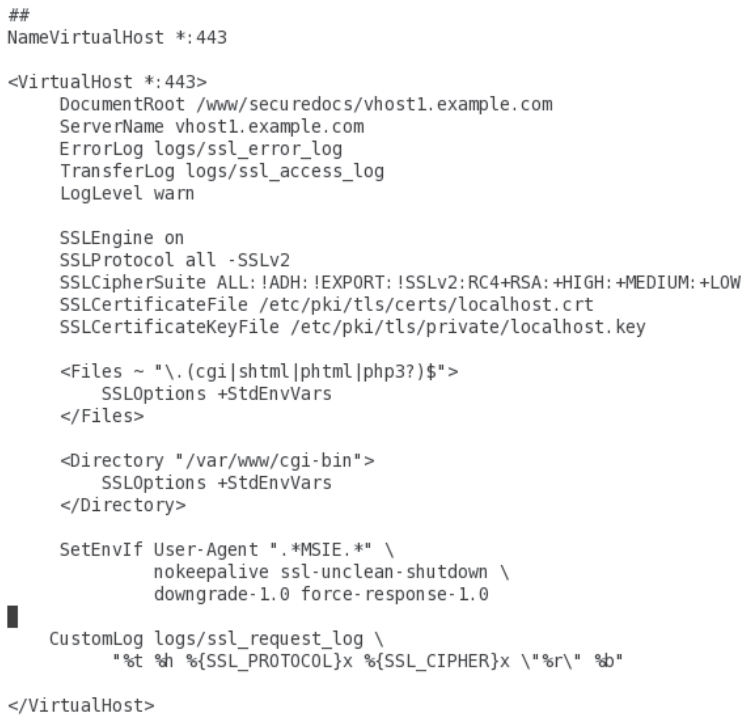
Now you can set up virtual hosts with the directives that follow. The default ssl.conf file also has a model virtual host container. But it is incomplete and a bit difficult to read with all of the comments. So a sample of the revised ssl.conf configuration file, focused on the virtual host container for the vhost1.example.com system, is shown in Figure 14-9.

In the default version of the ssl.conf file, examine the **<VirtualHost _default_:443>** container. Compare it to the **<VirtualHost *:80>** container in httpd.conf. 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



```
##
NameVirtualHost *:443

<VirtualHost *:443>
     DocumentRoot /www/securedocs/vhost1.example.com
     ServerName vhost1.example.com
     ErrorLog logs/ssl_error_log
     TransferLog logs/ssl_access_log
     LogLevel warn

     SSLEngine on
     SSLProtocol all -SSLv2
     SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM:+LOW
     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>

     SetEnvIf User-Agent ".*MSIE.*" \
              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>
```

You should also include **ServerAdmin**, **DocumentRoot**, and **ServerName** directives. Example directives that would be consistent with the virtual hosts created in the preceding section include

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

While the **DocumentRoot** directive can be set to any directory, it's appropriate if only 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 web site to be set up in different files, they should be changed. But for our purposes, you can stick with the default options shown here. Based on the **ServerRoot** directive from the httpd.conf file, these log files can be found in the /var/log/httpd directory.

```
ErrorLog logs/ssl_error_log
TransferLog logs/ssl_access_log
LogLevel warn
CustomLog logs/ssl_request_log \
          "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
```

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

```
SSLEngine on
SSLProtocol all -SSLv2
SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM:+LOW
SSLCertificateFile /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
```

The stanza 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 stanza deals with situations associated with the Microsoft Internet Explorer as a browser client:

```
SetEnvIf User-Agent ".*MSIE.*" \
         nokeepalive ssl-unclean-shutdown \
         downgrade-1.0 force-response-1.0
```

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

```
</VirtualHost>
```

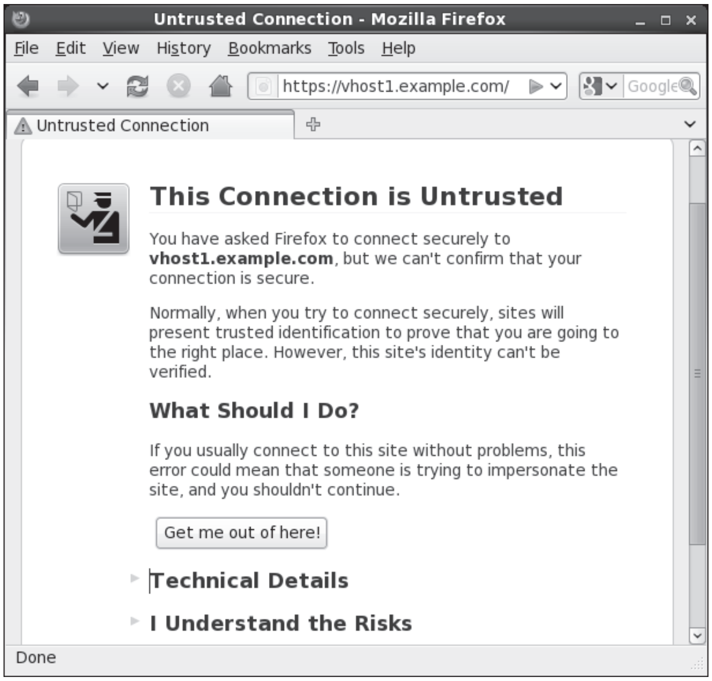
When Apache is configured with a secure site, regular GUI clients that access that site get a warning about the secure web host like that seen for so many web sites on the Internet, as shown in Figure 14-10.

Create a New SSL Certificate

While the default SSL certificate listed in the ssl.conf configuration file can work for basic configuration, you may want to either create a specialized local certificate or otherwise use an actual certificate purchased from a certificate authority (CA) such as VeriSign and Thawte.

FIGURE 14-10

A warning about secure hosts


Untrusted Connection - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://vhost1.example.com/
Google
Untrusted Connection

This Connection is Untrusted

You have asked Firefox to connect securely to **vhost1.example.com**, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Navigate to the /etc/pki/tls/certs directory. Note the file named 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. Unless you purchase an actual certificate from a CA, clients will still see screens such as that shown in Figure 14-10.

But you may need to know how to set up a "self-signed certificate." To that end, the following command, when run from the /etc/pki/tls/certs directory, automatically generates a private key and a certificate for the cited FQDN, as shown in Figure 14-11.

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

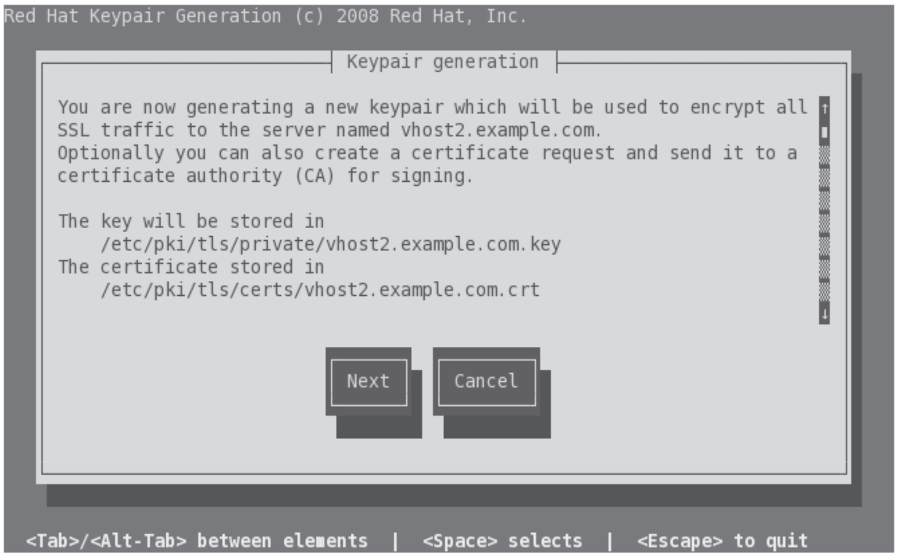
The **genkey** command is convenient, as when the process is complete, it automatically writes the key to the /etc/pki/tls/private directory and writes the certificate to the /etc/pki/tls/certs directory.

exam watch

The published outline for the Red Hat prep course for the RHCE (RH254) specifies the deployment of "an SSL-encapsulated web service." SSL certificates are part of that deployment process. If you're asked to create a certificate during an exam, keep the size of the key to the minimum allowed. The process of generating even 512-bit keys can take several minutes.

FIGURE 14-11

Prompts for a self-signed certificate



```
Red Hat Keypair Generation (c) 2008 Red Hat, Inc.

                    Keypair generation

You are now generating a new keypair which will be used to encrypt all
SSL traffic to the server named vhost2.example.com.
Optionally you can also create a certificate request and send it to a
certificate authority (CA) for signing.

The key will be stored in
    /etc/pki/tls/private/vhost2.example.com.key
The certificate stored in
    /etc/pki/tls/certs/vhost2.example.com.crt

                    Next      Cancel

 <Tab>/<Alt-Tab> between elements  |  <Space> selects  |  <Escape> to quit
```

For the purpose of this section, select Next to continue. In the step shown in Figure 14-12, you'd select a key size. What you select depends on the time available and the need for security. The Linux random number generator may require additional activity; this may be an excellent time to put the process aside and do something else.

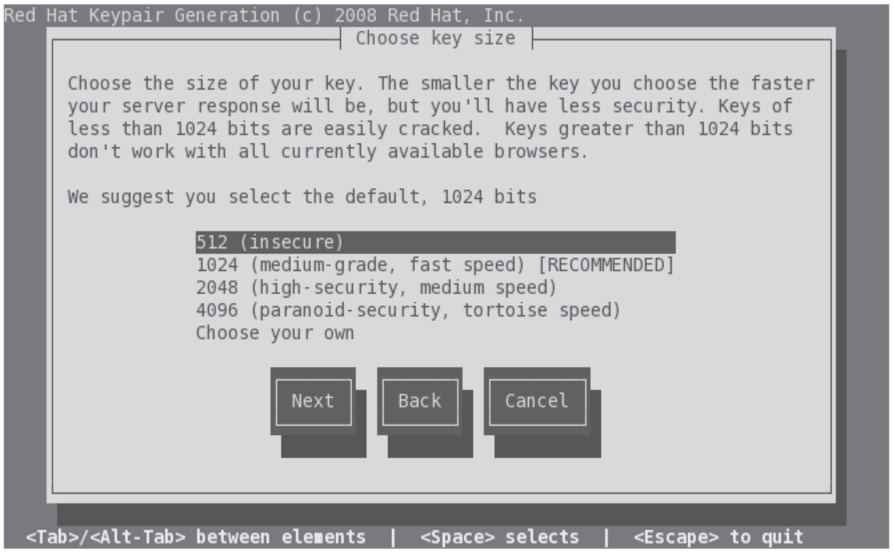
If you have nothing else to do and need to speed up the process, run some of the scripts in the /etc/cron.daily directory. Run some of the **find** commands described in Chapter 3. Click a bunch of times in an open terminal.

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're actually preparing to purchase a valid certificate, select No to continue. You're prompted to encrypt the key with a passphrase, as shown in Figure 14-13.

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 14-12

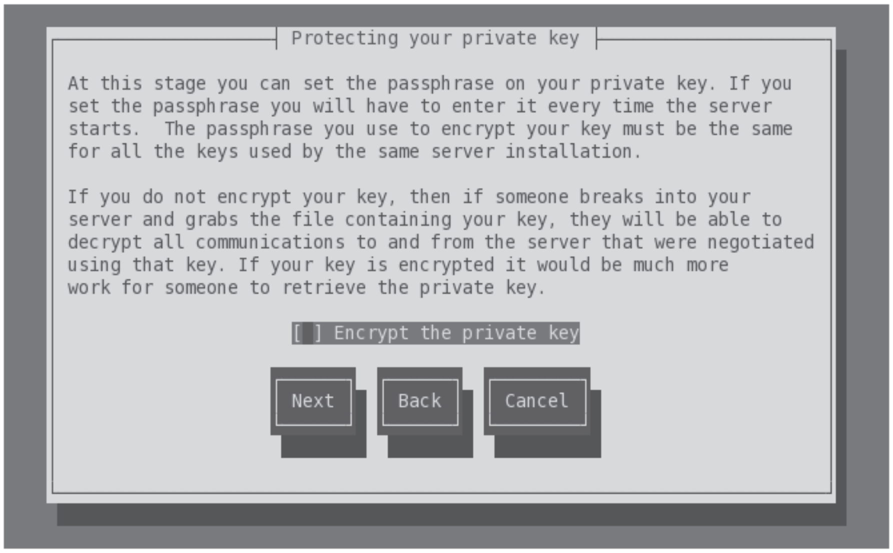


```
Red Hat Keypair Generation (c) 2008 Red Hat, Inc.
                         Choose key size

   Choose the size of your key. The smaller the key you choose the faster
   your server response will be, but you'll have less security. Keys of
   less than 1024 bits are easily cracked.  Keys greater than 1024 bits
   don't work with all currently available browsers.

   We suggest you select the default, 1024 bits

               512 (insecure)
               1024 (medium-grade, fast speed) [RECOMMENDED]
               2048 (high-security, medium speed)
               4096 (paranoid-security, tortoise speed)
               Choose your own

                    Next      Back      Cancel

 <Tab>/<Alt-Tab> between elements  |  <Space> selects  |  <Escape> to quit
```

Select a key size for an SSL certificate

FIGURE 14-13

Option to protect with a passphrase



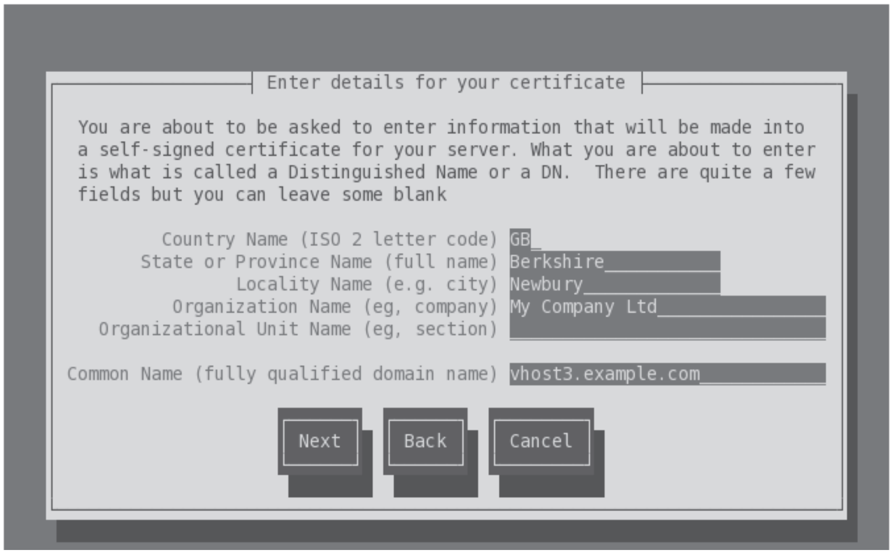
```
Protecting your private key

At this stage you can set the passphrase on your private key. If you
set the passphrase you will have to enter it every time the server
starts.  The passphrase you use to encrypt your key must be the same
for all the keys used by the same server installation.

If you do not encrypt your key, then if someone breaks into your
server and grabs the file containing your key, they will be able to
decrypt all communications to and from the server that were negotiated
using that key. If your key is encrypted it would be much more
work for someone to retrieve the private key.

                 [ ] Encrypt the private key

              Next      Back      Cancel
```

Test Pages

You may need to create some index.html files to test virtual hosts in various situations, to test various pre-production configurations, or even during an exam. Fortunately, the Red Hat exams don't test knowledge of HTML. You could use

FIGURE 14-14

SSL certificate details



```
Enter details for your certificate

 You are about to be asked to enter information that will be made into
 a self-signed certificate for your server. What you are about to enter
 is what is called a Distinguished Name or a DN.  There are quite a few
 fields but you can leave some blank

         Country Name (ISO 2 letter code) GB_
       State or Province Name (full name) Berkshire__________
              Locality Name (e.g. city) Newbury____________
          Organization Name (eg, company) My Company Ltd______________
   Organizational Unit Name (eg, section) ____________________________

Common Name (fully qualified domain name) vhost3.example.com__________

              Next      Back      Cancel
```

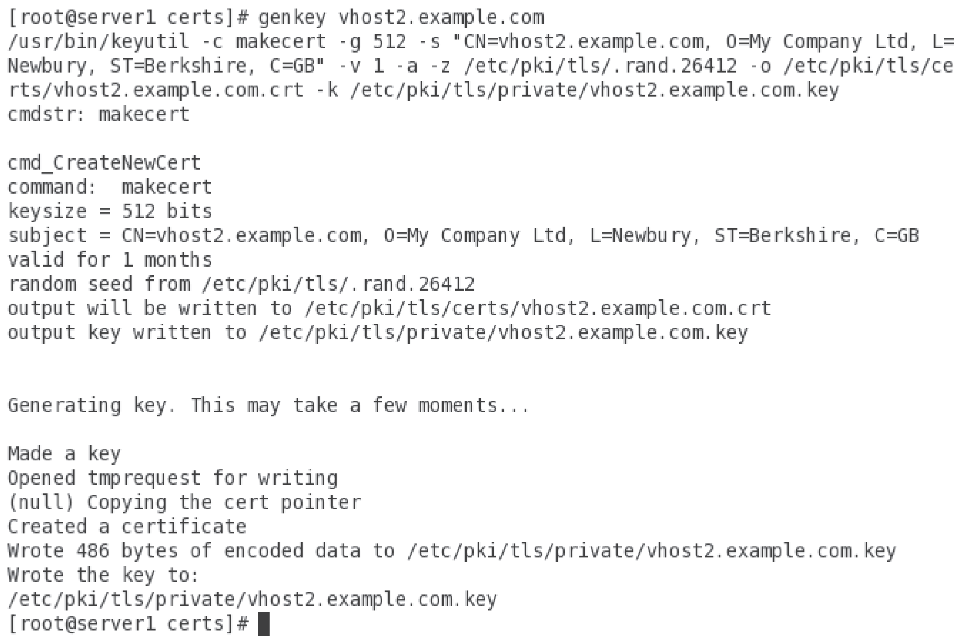


FIGURE 14-15

SSL certificate command output

```
[root@server1 certs]# genkey vhost2.example.com
/usr/bin/keyutil -c makecert -g 512 -s "CN=vhost2.example.com, O=My Company Ltd, L=
Newbury, ST=Berkshire, C=GB" -v 1 -a -z /etc/pki/tls/.rand.26412 -o /etc/pki/tls/ce
rts/vhost2.example.com.crt -k /etc/pki/tls/private/vhost2.example.com.key
cmdstr: makecert

cmd_CreateNewCert
command:  makecert
keysize = 512 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.26412
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
(null) Copying the cert pointer
Created a certificate
Wrote 486 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 certs]# 
```

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 I put into the index.html file for the regular vhost1.example.com web site is the following text:

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

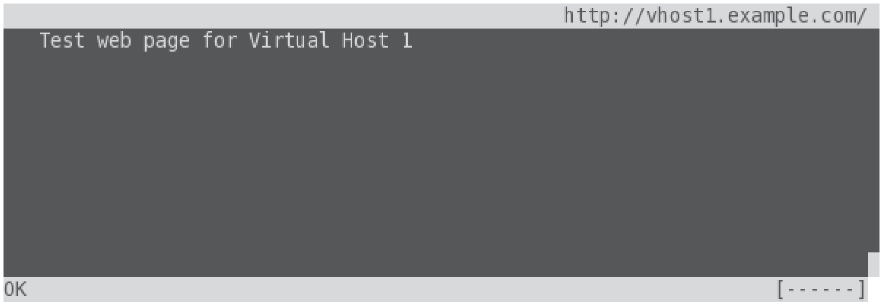
Once appropriate changes were made to Apache configuration files, I restarted the service. When I then ran the **elinks http://vhost1.example.com** command, the screen shown in Figure 14-16 appeared.

Syntax Checkers

In many cases, the **apachectl restart** and the **/etc/init.d/httpd restart** 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,

FIGURE 14-16

A test web page



```
                                              http://vhost1.example.com/
  Test web page for Virtual Host 1

OK                                                               [------]
```

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
```

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

Apache Troubleshooting

When the right Apache packages are installed, the default configuration normally creates a running system. Basic syntax can be checked with the **httpd -t** command. But if you're setting up a real web site, you probably want more than just the test page. Before making changes, back up the httpd.conf Apache configuration file. 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, your computer is running httpd as a normal user (not the user apache) with a port below 1024.
- **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 isn't running** Check the error_log in the /var/log/httpd directory.

- **Apache isn't running after a reboot** Run **chkconfig --list httpd**. Make sure Apache (httpd) is set to start at appropriate runlevels during the boot process with the command

  ```
  # chkconfig httpd on
  ```

- **You need to stop Apache** Send the parent process a **TERM** signal, based on its PID. By default, this is located in /var/run/httpd.pid. You kill Apache with a command such as

  ```
  #kill -TERM `cat /var/run/httpd.pid`
  ```

- Alternatively, you can use the **apachectl stop** command.

on the job

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

EXERCISE 14-5

Set Up a Virtual Web Server

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

1. Back up the httpd.conf file from the /etc/httpd/conf directory.
2. Add a virtual web site for the fictional company LuvLinex, with a URL of www.example.com. Don't forget to modify the **NameVirtualHost** directive. Use the sample comments at the end of the httpd.conf file for hints as needed.
3. Assign the **DocumentRoot** directive to the /luvlinex directory. (Don't forget to create this directory on your system as well.)
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 Web site.
   ```

5. Save this file.

6. If you've enabled SELinux on this system, you'll have to apply the **chcon** command to this directory:

```
# chcon -R -u system_u /luvlinex/
# chcon -R -t httpd_sys_content_t /luvlinex/
```

7. To make sure the changes survive a SELinux relabel, the following command should document the change in the file_contexts.local file in the /etc/selinux/targeted/contexts/files directory.

```
# semanage fcontext -a -s system_u -t httpd_sys_content_t /luvlinux/
```

8. If you've created a DNS service, as discussed in Chapter 11, update the associated database. Otherwise, update /etc/hosts with www.example.com and the appropriate IP address.
9. If you want to check the syntax, run the **httpd -t** and **httpd -D DUMP_VHOSTS** commands.
10. Remember to restart the Apache service; the proper way is with the **apachectl restart** command.
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 web site (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

No programming experience is required. When you see the RHCE objective to "Deploy a basic CGI application", the requirement is easier than it looks. In fact, the directions are accessible from the Apache documentation, available from the httpd-manual package. When installed, navigate to http://localhost/manual page. Apache documentation should appear. Select CGI: Dynamic content for detailed directions, explained in the following sections.

Apache Configuration Changes for CGI Files

To allow Apache to read CGI files, the httpd.conf file includes the **LoadModule cgi_module** directive. To make it easier for clients to find CGI files through their browsers, Apache includes the **ScriptAlias** directive. For example, the following **ScriptAlias** directive links the cgi-bin subdirectory to the default /var/www/cgi-bin directory:

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

With this **ScriptAlias** directive, if the web site 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. However, the associated <Directory> container does not allow executable scripts:

```
<Directory /var/www/cgi-bin>
     AllowOverride None
     Options None
     Order allow,deny
     Allow from all
</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 to actually be executable by the Apache server:

```
<Directory /var/www/cgi-bin>
     AllowOverride None
     Options ExecCGI
     AddHandler cgi-script .pl
     Order allow,deny
     Allow from all
</Directory>
```

The **AllowOverride None** command prevents regular users from changing permissions/settings in that directory. Otherwise, smarter users could read the CGI files in the /var/www/cgi-bin directory, a security risk. The **Options ExecCGI** line supports executable scripts in the noted directory. The **AddHandler** directive associates CGI scripts with files with the .pl extension. The **Order allow,deny** command sets up authorization checks; **Allow from all** lets all users run scripts in this directory.

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

```
ScriptAlias /cgi-bin/ /www/docs/vhost1.example.com/cgi-bin/
```

The Apache documentation continues with instructions on how you can write a simple CGI script.

Set Up a Simple CGI Script

The Apache documentation includes instructions on how to set up a simple CGI script, in the Perl programming language. Make sure 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 hello.pl, based on the following code:

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

The first line is similar to the **#!/bin/bash** found in many scripts; in this case, **perl** is the command interpreter. The content type is declared, followed by two carriage returns (as symbolized by the **\n**). The final line prints the expression commonly used for introductory program scripts. Such scripts need to be executable, with 755 permissions. In other words, once the hello.pl file is saved, you'd apply the noted permissions with the following command:

```
# chmod 755 hello.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 file type to the file and directory with the **chcon** command, and then make sure the file type stays applied after a SELinux relabel with the **semanage fcontext -a** command. Both commands were described earlier in this chapter and in Chapter 11.

But in most cases, you'll be setting up multiple web sites as virtual hosts in Apache configuration files. If you didn't already add the **Options ExecCGI** and **AddHandler cgi-script .pl** directives described earlier, they should be added to the virtual host container, along with an appropriate **ScriptAlias** directive:

```
Options ExecCGI
AddHandler cgi-script .pl
ScriptAlias /cgi-bin/ /www/docs/vhost1.example.com/cgi-bin/
```

Then you'd copy the hello.pl script to the noted directory and make it executable by all users. Don't forget to make sure the script and directory has the same SELinux contexts (httpd_sys_script_exec_t) as scripts in the /var/www/html/cgi-bin directory. To make sure the change survives a relabel, you also should apply the following command to that directory:

```
# semanage fcontext -a -s system_u -t httpd_sys_script_exec_t
/www/docs/vhost1.example.com/cgi-bin/
```

Connections to a Web Site

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 hello.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/hello.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 web site	Install Apache, configure appropriate files in the /var/www/html directory.
You need to configure multiple web sites	With Apache, use the <VirtualHost> containers at the end of the httpd.conf file as a template for as many web sites as needed.
You need to configure a secure web site	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 web site	From the /etc/pki/tls/certs directory, run the **genkey www.example.org** command.
The Apache service is not running after a reboot	Make sure the httpd service starts in appropriate runlevels with the **chkconfig --list 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 ExecCGI option is active; the AddHandler directive specifies **cgi-script .pl**, the ScriptAlias is pointed to the appropriate directory; the script is executable by all users and matches 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. The httpd.conf and ssl.conf files are organized in <VirtualHost> containers. With the help of sample stanzas in both noted configuration files, you can create virtual regular and secure hosts for multiple web sites 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 **iptables**-command-based firewalls. 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** and **NameVirtualHost** directives direct traffic to the Apache web server to ports like 80 and 443, along with specified virtual hosts. Host- and user-based security can also be set up within Apache configuration files with commands like **htpasswd** and directives such as **Allow** and **Deny**.

With the right security options, user- and group-managed directories are possible. In fact, there's a commented stanza that can enable directories 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 like www.example.org from the /etc/pki/tls/certs directory with a command like **genkey www.example.org**.

The configuration of CGI content on an Apache web site is easier than it looks. In fact, detailed information on the process is included 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 than twice as many Internet web sites as 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 **iptables** rules and various SELinux booleans and contexts.
- ❑ Apache supports security by specifying active ports through the **Listen** and **NameVirtualHost** 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 web sites on your server, even with only one IP address. This is possible through the use of virtual hosts.
- ❑ The RHEL configuration supports the configuration of virtual hosts for regular web sites at the end of the /etc/httpd/conf/httpd.conf file.
- ❑ The RHEL configuration supports the configuration of secure virtual hosts for regular web sites at the end of the /etc/httpd/conf.d/ssl.conf file.
- ❑ SSL certificates can be created with the **genkey** command, when run from the /etc/pki/tls/certs directory.

Deploy a Basic CGI Application

- ❑ The use of CGI content depends on configuration options like ScriptAlias, ExecCGI, and AddHandler cgi-script.
- ❑ Standard CGI scripts require 755 permissions. If needed, sample instructions are available 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 stanza for a virtual host, what computers are allowed access?

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

   ____

6. What standard ports do you need to open in a firewall to allow access to a regular web site 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 CD that accompanies the book, in the Chapter14/ subdirectory. In case you haven't yet set up RHEL 6 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.

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 the **/etc/init.d/httpd reload** command. The **service httpd reload** 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 elizabeth**.

5. As described in the chapter, the **Order Allow,Deny** directive denies access to all systems by default, except those explicitly allowed access. So access is limited to computers on the 192.168.0.0/24 network.

6. The standard ports you need to open in a firewall to allow access to regular and secure web sites are 80 and 443.

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 combines SGID permissions, standard 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 groupinstall "Web Server"** command. (To find appropriate package group names, run the **yum grouplist** command.) This assumes a proper connection to a repository, as discussed in Chapter 7.

To configure Apache to start, run the **apachectl start** command. To make sure it starts the next time the system is booted, run the **chkconfig httpd on** 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** is located in /var/www/html. You can 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 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 in the main Apache configuration file, /etc/httpd/conf/httpd.conf. 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 a relative directory—are set relative to **ServerRoot**. Don't change this unless you're ready to adjust the SELinux contexts of the new directory accordingly.
2. Set the **NameVirtualHost** directive to the port (80) serving your intended network audience. Don't assign any IP addresses.
3. Add separate **VirtualHost** containers with settings appropriate for the big.example.com and small.example.com systems.
4. Assign the **ServerAdmin** to the e-mail address of this web site's administrator.
5. Configure a unique **DocumentRoot** directory for each virtual host.
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 logs directory, such as the following:

   ```
   ErrorLog logs/big.example.com-error_log
   ```

8. Make sure to close the **VirtualHost** container (with a **</VirtualHost>** directive at the end of the stanza).
9. Repeat the process for the second web site, making sure to set the second **ServerName** to **small.example.com**.
10. Close and save the httpd.conf file with your changes.
11. Create any new directories that you configured with the **DocumentRoot** directives.

12. 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.
13. Make sure these domain names are configured in the authoritative 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.
14. Use the Security Level Configuration tool (**system-config-securitylevel**) utility to allow HTTP data through the firewall, as discussed in Chapter 10.
15. You'll need to configure appropriate SELinux file types on the directory associated with the **DocumentRoot**. For example, if that directory is /virt1, one way to do so is with the following commands:

    ```
    # chcon -R -u system_u /virt1/
    # chcon -R -t httpd_sys_content_t /virt1
    ```

    In addition, you'll need to set up the file_contexts.local file in /etc/selinux/targeted/contexts/files directory with a command such as

    ```
    # semanage fcontext -a -s system_u -t httpd_sys_script_exec_t /virt1
    ```

16. Make sure to run the **apachectl graceful** command (or something similar) to make Apache reread the httpd.conf configuration file, with the changes you've made.
17. Now you can test the results. Navigate to a remote system, and try to access the newly created web sites 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 web site.

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 web site:

```
/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 basic steps that performed in Lab 3 and use the certificate and key files created in Lab 4. One difference is that secure web sites are generally configured in the /etc/httpd/conf.d/ssl.conf file. In addition, you should be concerned about the following:

1. While not absolutely required, it's a good practice 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 web site.
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 web site.
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 web sites. The only difference is in the SSLCertificateFile and SSLCertificateKeyFile directives:

```
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.big for the noted directives in the secure virtual host stanza for that web site.

Lab 6

In the default httpd.conf file, the configuration of user home directories requires that you enable the UserDir directive. You can then customize the commented stanza 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. It is understood that the authentication database file for that directory may lead to a different password for that user.

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.

Lab 7

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

- 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.
- Make the other users a member of that group named techsupport.
- Set up appropriate permissions to support access by members of the techsupport group, normally 2770 permissions. It should also include either regular executable permissions by other users, or executable permissions by the user named apache.
- Create a public_html/ subdirectory of the new user's home directory.
- Set up an index.html file in that subdirectory. It should already be set with ownership by the techsupport group.

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. That container should also include the **Options ExecCGI**, and **AddHandler cgi-script .pl** directives. While it's normally best to have scripts in the same DirectoryRoot as that 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, with the SELinux file type documented in the file_contexts.local file in the /etc/selinux/targeted/contexts/files directory. In any case, a successful result is as suggested in the lab question.