11

System Services and SELinux

CERTIFICATION OBJECTIVES

This is a "big picture" chapter with respect to the RHCE objectives. Those objectives are focused on common tasks that you'll perform on the job. These tasks relate to the detailed configuration of RHCE-level services.

RHEL 6 incorporates basic system configuration files in the /etc/sysconfig directory. These files are called by different services in the /etc/init.d directory. The services then use custom configuration files in dedicated locations. Integral to this process is the configuration of SELinux, as it includes a substantial number of custom options for various services.

These tools will be tested on the one service that you might install on all bastion systems, SSH. As it is the common service for all such systems, crackers everywhere want to find a weakness in SSH. So this chapter also describes how you can make SSH services more secure. This is the first chapter where the three virtual machines created in Chapters 1 and 2 will be used.

In this chapter, you'll also the boolean options used by SELinux to secure those server services associated with the RHCE objectives. While SELinux is a common source of frustration, it is easier to handle when you know the options that support desired features.

In addition, this chapter covers the basic procedure that should be followed to make sure the service is operational, accessible from remote systems, and up the next time the system is rebooted.

INSIDE THE EXAM

This section includes tasks that will be repeated in the remainder of this book.

- Install the packages needed to provide the service

Whether you're installing the Samba file server or a DNS caching-only nameserver, the same tools are used. Yes, these are the same **rpm** and **yum** commands, along with the Package Management tools described in Chapter 7. To save time, you might use these commands to install the services described in Chapters 12 through 17.

- Configure the service to start when the system is booted
- Configure SELinux to support the service

While the detailed configuration of individual services is the province of each chapter, the steps required to configure a service to start during the boot process are based on common commands like **chkconfig**. In addition, the configuration of SELinux to support a service requires access to and the configuration of similar options. As suggested in the introduction, there's a special focus on the SSH service.

- Configure key-based authentication

With key-based authentication, you'll be able to connect to remote systems by using private/public key pairs. Password transmission over the network will no longer be required. The 1024 or more bits associated with such authentication are a lot harder to crack than a password transmitted over a network. Given the importance of SSH security, you'll also

- Configure additional options described in documentation

CERTIFICATION OBJECTIVE 11.01

Red Hat System Configuration

In this section, you'll review basic information on how services are configured on Red Hat systems. The actual process associated with a service is a daemon. Such daemons are executable files, normally stored in the /usr/sbin and /sbin directories. Red Hat configures custom parameters and more in the /etc/sysconfig directory.

Service Management

As discussed throughout the book, services are controlled with scripts in the /etc/init.d directory. But those scripts just put things together. As described in Chapter 4, they can be used to **start**, **stop**, or **restart** a service. In many cases, those scripts can be used to **reload** the service with modified configuration files, without kicking off currently connected users.

While the real daemons are in the /sbin and /usr/sbin directories, the /etc/init.d scripts do more. They call the daemons with parameters configured in the /etc/sysconfig directory. The daemons then call service-specific configuration files.

In any case, the scripts in the /etc/init.d directory are hard-linked to the scripts in the /etc/rc.d/init.d directory. And the **service** command in the /sbin directory is a front end to those scripts. In other words, the following commands are functionally identical:

```
# /etc/init.d/sshd restart
# /etc/rc.d/init.d/sshd restart
# service sshd restart
```

System Services

The files in the /etc/sysconfig directory are normally used with /etc/init.d scripts. They're as varied as the services included in the /etc/init.d directory. As they include basic configuration options for each daemon, they drive the basic operation of each service. These files are briefly described in Table 11-1. While the list goes beyond the services covered in the RHCSA and RHCE objectives, the list does not include files in /etc/sysconfig subdirectories. While the descriptions frequently refer to discussions in other chapters, most files in this directory are rarely edited by administrators. In many cases, these files aren't even described in the noted chapters. They're listed to help give you a feel for the depth and breadth of Red Hat services. Remember, the RHCSA and RHCE are separate exams, covered in Chapters 1 through 10 and 11 through 17, respectively.

TABLE 11-1 Files in the /etc/sysconfig Directory

File	Description
atd	Supports limits on the number of jobs and time interval between jobs, as described in Chapter 9.
auditd	Includes switches for the audit daemon.
authconfig	Specifies the options for the Authentication Configuration tool, described in several chapters.
autofs	Works with the automounter, discussed in Chapter 6.
cgconfig, cgred.conf	Controls the control group rules daemon.
clock	Notes the current time zone, discussed in Chapter 5.
cpuspeed	Supports changes in CPU speeds based on workloads.
crond	Includes options for the cron daemon, as described in Chapter 9.

TABLE 11-1 Files in the /etc/sysconfig Directory (*continued*)

File	Description
ebtables-config	Configures data-link-level firewalls.
firstboot	Specifies whether the First Boot process will be run; if you change this to yes, the First Boot process is run upon the next reboot.
grub	Sets the boot drive and can set other boot parameters, discussed in Chapter 5.
httpd	Configures options for the Apache web server daemon, as described in Chapter 14.
i18n	Specifies the currently configured language.
init	Sets options during the boot process before the scripts in the /etc/init directory, discussed in Chapter 5.
ip6tables	Sets firewall configuration rules for IPv6 networking.
ip6tables-config	Defines firewall service options for IPv6.
iptables	Sets firewall configuration rules for IPv4 networking, discussed in Chapters 4 and 10.
iptables-config	Defines firewall service options for IPv4.
irqbalance	Configures how interrupts are loaded.
kadmin	Sets up Kerberos administration.
kexec	Sets up the boot into a different kernel.
kernel	Configures how new kernels are updated, briefly described in Chapter 5.
keyboard	Notes parameters of the current keyboard.
krb5kdc	Specifies arguments for starting the Kerberos key distribution center.
ksm	Sets up kernel page swaps for virtual machines.
ldap	Includes daemon options for the LDAP server.
libvirtd	Sets up basic parameters for virtualization support.
libvirt-guests	Configures the startup or shutdown of virtual machine guests.
mcelog	Specifies options for the kernel machine check log.
named	Sets up options for the standard Red Hat DNS server, described in Chapter 17.
netconsole	Supports remote logging.
network	Configures basic network parameters, described in Chapter 4.
nfs	Sets up specific NFS parameters and port numbers, discussed in Chapter 16.
nspluginwrapper	Supports plugins for web browsers.
ntpd	Includes basic options for the NTP server, described in Chapter 17.

TABLE 11-1 Files in the /etc/sysconfig Directory (*continued*)

File	Description
ntpdate	Includes basic options for the NTP client, as discussed in Chapter 5.
openct	Specifies control options for smart card readers.
prelink	Configures library management.
raid-check	Options for the weekly job to check RAID arrays.
readahead	Specifies caching information for the boot process.
readonly-root	Sets read/write settings during the boot process.
rsyslog	Includes options for the rsyslog daemon, as discussed in Chapters 9 and 17.
samba	Supports options for three Samba daemons, smbd, nmbd, and winbindd, as discussed in Chapter 15.
sandbox	Works with limits on users, such as xguest discussed in Chapter 8.
saslauthd	Helps configure authentication support.
sa-update	Supports nightly system activity reports.
smartmontools	Configures support for hard disk monitoring.
spamd	Specifies options for the spamd daemon.
sysstat	Includes log and compression options for system status tools.
sysstat.ioconf	Lists available drive devices for sysstat.
system-config-firewall	Summarizes non-default firewall rules.
system-config-users	Configures basic options for the user manager.
tgtd	Sets up basic SCSI and iSCSI parameters.
udev	Supports caching of device files.
vncservers	Configures basic VNC server options, as discussed in Chapter 7.
wpa_supplicant	Supports options for Wi-Fi Protected Access (WPA).
xinetd	Sets up the Extended Internet Super-Server, described in Chapter 10.

In most cases, each of these files supports the use of switches as described in associated man pages. For example, the /etc/sysconfig/httpd file can be used to set up custom options for starting the Apache web server. In that file, the **OPTIONS** directive would pass switches to the /usr/sbin/httpd daemon, as defined in the httpd man page.

Bigger Picture Configuration Process

In general, when you configure a network server service on Linux, take the following steps. The following guidelines are general; for example, sometimes it's appropriate to modify SELinux options first. Sometimes, you'll want to test a service locally and remotely before making sure the service starts automatically upon the next reboot.

1. Install the service, normally with a command like **rpm** or **yum**. In some cases, the RHCE exam or real-life needs require the installation of additional packages.
2. Edit the service configuration files. In some cases, there are one or more configuration files; for example, you may need to modify and customize several configuration files to set up the Postfix e-mail server in the /etc/postfix directory.
3. Modify SELinux booleans. As discussed later in this chapter, different services have a variety of options controlled by SELinux. For example, SELinux changes are required to allow the Samba file server to share files in read/write or in read-only mode.
4. Start the service, as typically controlled by a script in the /etc/init.d directory. You'll also need to make sure the service starts the next time the system is booted, as discussed later in this chapter.
5. Test the service locally. Make sure it works from appropriate client(s) on the local system.
6. Open applicable firewall ports, based on **iptables**, TCP wrappers, and more. Configure access to desired users and systems.
7. Test the service remotely. If the right ports are open, the service should work as well as when you connect locally. The service should not work for undesired users.

Available Configuration Tools

In general, it's most efficient to configure various services from the command line interface. An administrator who knows a service well will be able to set it up at least for basic operation in just a few minutes. However, all but the most capable

administrators can't specialize in everything. To that end, Red Hat has developed a number of configuration tools. When used properly, these tools will modify the right configuration files. Some are installed with each service, others have to be installed separately. Most of these tools are accessible from a GUI command line interface with a **system-config-*** command.

The tools used in this book (and a couple more) are summarized in Table 11-2.

There are actually fewer GUI configuration tools for RHEL 6 when compared to RHEL 5. For example, RHEL 5 included GUI configuration tools to configure a DNS server and the GUI display. Personally, I find it easier to edit DNS server configuration files directly. To me, the RHEL 5 GUI tool was cumbersome.

TABLE 11-2

Red Hat Configuration Tools

Tool	Command	Function
Add/Remove Software	**gpk-application**	**Front end to yum** command; manages current software configuration
Authentication Configuration	**authconfig*, system-config-authentication**	**Connection between clients and authentication databases**
Date/Time Properties	**system-config-date**	**Management of the current time zone, NTP client**
Firewall Configuration	**system-config-firewall**	**Configuration of iptables** firewalls, masquerading, and forwarding
Language Selection	**system-config-language**	**Language selection within the GUI**
Network Management	**system-config-network**	**Network device / DNS configuration at the console**
Network Connections	**nm-connection-editor**	**Detailed network device configuration tool**
Printer Configuration	**system-config-printer**	**Management of the CUPS print server**
SELinux Management	**system-config-selinux**	**Configuration of SELinux booleans, labels, users, etc.**
Software Update	**gpk-update-viewer**	**Review and install available updates to installed packages**
User Manager	**system-config-users**	**Management and configuration of users and groups**

CERTIFICATION OBJECTIVE 11.02

Security-Enhanced Linux

Security-Enhanced Linux (SELinux) provides one more layer of security. Developed by the U.S. National Security Agency, SELinux makes it more difficult for crackers to use or access any file or service if they break in. SELinux assigns different contexts to each file, known as *subjects*, *objects*, and *actions*.

Basic SELinux options were covered in Chapter 4, as it is also a requirement for the RHCSA certification. For the RHCE, the focus of SELinux relates to various services. Specifically, you need to know how to configure SELinux to support the Apache web server, Domain Name System (DNS) services, an FTP server, Network File System (NFS) servers, Samba file servers, Simple Mail Transport Protocol (SMTP) servers, Secure Shell (SSH) servers, and Network Time Protocol (NTP) servers.

The requirements for each of these services are covered in this and later chapters of this book. As the SELinux configuration for each service requires the use of the same commands and tools, they're covered here.

Perhaps the key commands and tools discussed in this section are **getsebool**, **setsebool**, **chcon**, **ls -Z**, and the SELinux Management Tool. While these are the same tools used in Chapter 4, the focus is different. To review, the **getsebool** and **setsebool** commands set boolean options in the files of the /selinux/booleans directory. A boolean is a binary option, 1 or 0, which corresponds to yes and no.

Options in the SELinux Booleans Directory

When configuring SELinux for a service, you'll generally make changes to boolean settings in the **/selinux/booleans** directory. Take a look at the files in this directory. The file names are somewhat descriptive.

For example, the ftp_home_dir boolean either allows or denies access to user home directories via an FTP server. It is disabled by default. In other words, if you configure the vsFTP server in Chapter 16 to support logins to user home directories without changes to SELinux, users won't be able to log in to their home directories via FTP.

Problems like this are a common source of frustration for administrators of RHEL systems. They do all the work to configure a service, they test out the configuration, they check their documentation, they think they've done everything right, and yet

the service doesn't work as they want. The solution is to make SELinux a part of what you do to configure a service.

In other words, run the following command:

```
$ cat /selinux/booleans/ftp_home_dir
```

By default, the output should be

```
0 0
```

That's two 0s. Supposedly one boolean is for the current setting, the other is for the permanent setting. In practice, the numbers don't reflect the differences, at least for RHEL 6. But the differences are still there. Because of this issue, the best way to see the current state of a boolean is the **getsebool** command. For example, the command

```
$ getsebool ftp_home_dir
```

leads to the following output:

```
ftp_home_dir --> off
```

Bottom line, if the current setting is 0, either of the following commands would activate the ftp_home_dir boolean only until the system is rebooted:

```
# setsebool ftp_home_dir 1
# togglesebool ftp_home_dir
```

As suggested by the name, the **togglesebool** command changes the current boolean from 0 to 1 and back. To repeat from Chapter 4, the way to make the change permanent from the command line is with the **setsebool -P** command, in this case:

```
# setsebool -P ftp_home_dir 1
```

exam Watch

Many service-related SELinux booleans are described in local documentation; for a list of associated man pages, run the* man -k _selinux *command. If you don't have access to the GUI SELinux Management tool during the exam, that could be a lifesaver.

Service Categories of SELinux Booleans

There are about 170 files in the /selinux/booleans directory. As the filenames in the /selinux/booleans directory are descriptive, you can use database filter commands like grep to help classify those booleans. Based on some of the services discussed in this book, the following would be some appropriate filtering commands:

```
$ ls /selinux/booleans | grep http
$ ls /selinux/booleans | grep ftp
$ ls /selinux/booleans | grep nfs
```

You'll explore each of these categories of booleans in more detail shortly. For a brief description of available booleans with their current status, run the **semanage boolean -l** command. The **semanage** command is part of and can be installed through the policycoreutils-python package.

Boolean Configuration with the SELinux Management Tool

One of the benefits of GUI tools is a view of the "big picture." With the SELinux Management tool, you can review the active booleans and quickly get a sense of whether SELinux is set to allow few or many options associated with a service. As discussed in Chapter 4, you can start the SELinux Management tool in a GUI desktop environment with the **system-config-selinux** command. In the left-hand pane, click Boolean. It opens access to a group of booleans in the right side of the window. Note the **ftp** filter added in Figure 11-1. It filters the system for all booleans related to FTP.

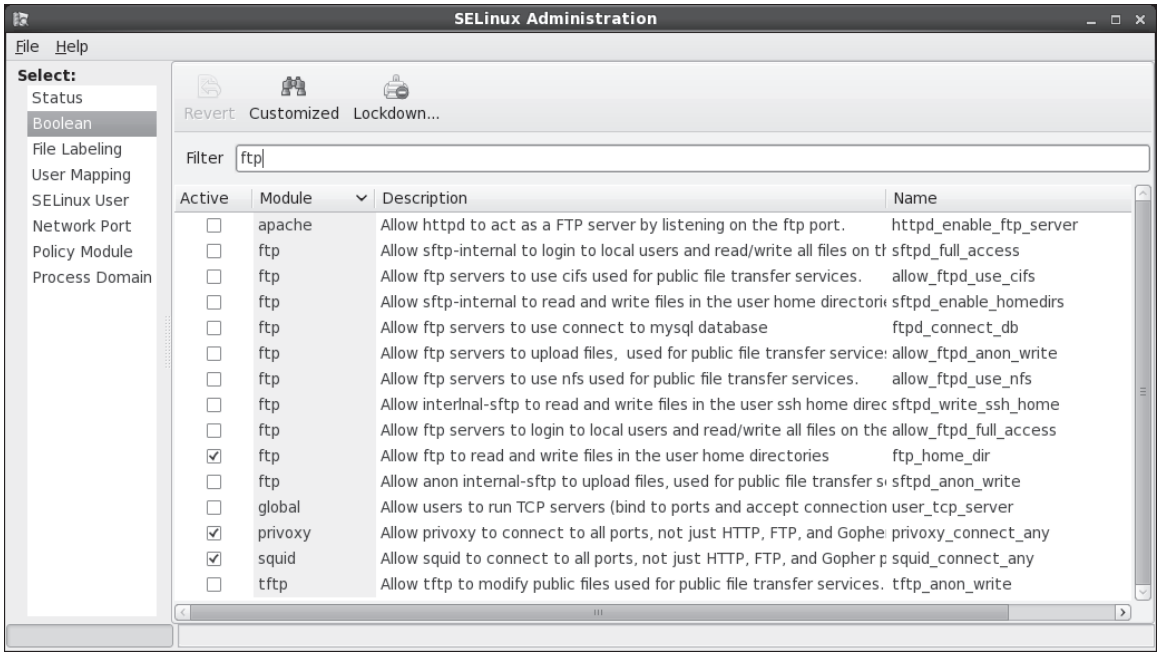
Compare the list to the output of the ls /selinux/booleans | grep ftp command described earlier. Note the differences. Some booleans shown through the SELinux Management tool don't show up directly with a grep of the file list in the /selinux/booleans directory. In general, booleans like user_tcp_server are only tangentially related to FTP services.

There are a number of categories shown in the left pane of the SELinux Management Tool window; they are described in the following sections. Most of the focus here will be in the Boolean category, where most of SELinux policies are customized.

In some cases, the boolean is associated with a requirement for a SELinux file context. For example, the allow_httpd_anon_write boolean works only if associated files and directories are labeled with the public_content_rw_type. To set that type on, say, the /var/www/html/files directory (and subdirectories), you would run the following command:

```
# chcon -R -t public_content_rw_type /var/www/html/files
```

FIGURE 11-1 Filter booleans with the SELinux Management tool.



Boolean Settings

The boolean settings discussed in the following sections fall into a number of categories. They're based on the services defined in the RHCE objectives. The SELinux settings do not stand alone. For example, if you enable the httpd_enable_homedirs boolean, you'll still have to configure the /etc/httpd/conf/httpd.conf file to support access to user home directories. Only after both SELinux and Apache are configured with such support can users connect to their home directories through that Apache server.

As there are no current SELinux booleans related to the Network Time Protocol (NTP) service, there is no separate section for NTP booleans in the following list.

Regular and Secure HTTP Services

There are a number of SELinux directives available to help secure the Apache web server, as summarized in the following bullets. Most are straightforward and self-

explanatory. They are ordered by the filename of the boolean as shown in the /selinux/booleans directory. While these booleans should apply to any web service, Red Hat assumes the use of the Apache web server. The descriptions specify the configuration if the boolean is active.

- **allow_httpd_anon_write** Allows the web service to write to files labeled with the public_content_rw_t type.
- **allow_httpd_mod_auth_ntlm_winbind** Permits access to the Microsoft NT LAN Manager (NTLM) and winbind authentication databases; requires an installed and active mod_auth_ntlm_winbind module for Apache.
- **allow_httpd_mod_auth_pam** Supports PAM access for user authentication; requires an installed and active mod_auth_pam module for Apache.
- **allow_httpd_sys_script_anon_write** Allows HTTP scripts to write to files labeled with the public_content_rw_t type.
- **httpd_builtin_scripting** Permits access to scripts, normally associated with PHP. Enabled by default.
- **httpd_can_check_spam** Supports the use of SpamAssassin for web-based e-mail applications.
- **httpd_can_network_connect** Supports access by scripts to external systems; normally disabled to minimize risks to other systems.
- **httpd_can_network_connect_cobbler** Supports access by scripts to an external Cobbler installation server; if this boolean is enabled, you should disabled the httpd_can_network_connect boolean.
- **httpd_can_network_connect_db** Allows connections to database server ports; more specific than httpd_can_network_connect.
- **httpd_can_network_memcache** Enables access to memory object caching; associated with the Pootle language translation project.
- **httpd_can_network_relay** Supports access to standard relay ports, such as those associated with HTTP and FTP; more specific than httpd_can_network_connect.
- **httpd_can_sendmail** Allows outgoing SMTP access from web-based e-mail applications.

- **httpd_dbus_avahi** Supports access to automated IP addressing, using the avahi service, via the D-bus message system. Allowed by default.
- **httpd_enable_cgi** Supports access to Common Gateway Interface (CGI) scripts. Allowed by default; requires scripts to be labeled with the httpd_sys_script_exec_t file type.
- **httpd_enable_ftp_server** Allows Apache to listen on the FTP port (normally 21) and work as an FTP server.
- **httpd_enable_homedirs** Enables configuration of Apache to read user home directories.
- **httpd_execmem** Supports programs such as those written in Java or Mono that require memory addresses that are executable and writable.
- **httpd_read_user_content** Allows the Apache web server to read all files in user home directories, not just those labeled with various httpd_*_t file types.
- **httpd_setrlimit** Allows changes to Apache file descriptor limits.
- **httpd_ssi_exec** Supports executable Server Side Includes (SSIs).
- **httpd_tmp_exec** Lets Apache run executable files from the /tmp directory.
- **httpd_tty_comm** Supports access by the apache user (lower case) for access to the files with secure certificates. Allowed by default.
- **httpd_unified** Enables access to all httpd_*_t labeled files, whether they be read-only, writable, or executable. Allowed by default.
- **httpd_use_cifs** Supports access from Apache to shared Samba files and directories labeled with the cifs_t file type.
- **httpd_use_gpg** Allows Apache to use GPG for encryption.
- **httpd_use_nfs** Supports access from Apache to shared NFS files and directories labeled with the nfs_t file type.
- **privoxy_connect_any** Enables access by the Privoxy web proxy server through standard Apache ports.
- **squid_connect_any** Enables access by the Squid web proxy server through standard Apache ports.
- **varnishd_connect_any** Enables access by the Varnish web proxy server through standard Apache ports.

Name Service

The name service daemon (**named**) is based on the Berkeley Internet Name Domain (BIND) software, which is the RHEL 6 DNS service. If you maintain an authoritative DNS zone, it's appropriate to activate the named_write_master_zones boolean. Then local DNS software can overwrite master zone files.

In general, this does not apply to the RHCE, as the objectives state that all you need to do is configure DNS as a caching or a forwarding name server. Such servers by definition cannot be authoritative for a specific domain. As such DNS servers do not have master zone files, the noted DNS boolean does not apply.

FTP

While there aren't quite as many configuration options for FTP servers when compared to those available for the Apache web server, they are varied and important. While these booleans should apply to any FTP server, Red Hat assumes the use of the vsFTP server. While there are a number of options associated with the FTP server associated with SSH, the boolean settings associated with the **sftp** commands are not active, and in any case they are not part of the configuration associated with the vsFTP server. In any case, changes to the sftp-booleans had no effect on my ability to perform the associated actions. Therefore, **sftp**-related SELinux settings are not covered in this book. None of these booleans are enabled by default.

- **allow_ftpd_anon_write** Supports uploads from anonymous users, to directories labeled with the public_content_rw_t type.
- **allow_ftpd_full_access** Supports full access to all directories shared via the FTP server, beyond those allowed by the allow_ftp_anon_write and the ftp_home_dir booleans.
- **allow_ftpd_use_cifs** Enables access to files labeled with the cifs_t file type, which is assigned to files from mounted Samba directories.
- **allow_ftpd_use_nfs** Enables access to files labeled with the nfs_t file type, which is assigned to files from mounted NFS directories.
- **ftpd_connect_db** Supports connections from an FTP server to a database.
- **ftp_home_dir** Allows access to user home directories.

NFS More of the basic SELinux booleans associated with the Network File Service (NFS) servers are enabled by default. So in some basic configurations, you'll be able to share certain directories with the NFS server without changes to SELinux booleans.

But alas, that's not always true. In fact, if you don't want to allow others to set up an NFS server, it may be safest to disable such booleans.

- **cdrecord_read_content** Supports access by the **cdrecord** command to shared files from network and locally mounted directories.
- **nfs_export_all_ro** Allows shared NFS directories to be exported with read-only permissions. Enabled by default.
- **nfs_export_all_rw** Allows shared NFS directories to be exported with read/write permissions. Enabled by default.
- **qemu_use_nfs** Enables access from KVM-based virtual machines of files from NFS mounted filesystems. Enabled by default.
- **use_nfs_home_dirs** Supports access of home directories from remote systems. Enabled by default.
- **virt_use_nfs** Enables access with the libvirt daemon of files from NFS mounted filesystems.

Samba Samba booleans are generally not enabled by default. So in most configurations, you'll need to make changes to SELinux to match changes to the Samba configuration files. These booleans include the following:

- **allow_smbd_anon_write** Supports writing to public directories, where access by specific users is not otherwise regulated. Requires directories labeled with the public_content_rw_t file type.
- **samba_create_home_dir** Allows Samba to create new home directories, such as for users who connect from other systems, using network authentication databases.
- **samba_domain_controller** Enables the configuration of the local Samba server as a local domain controller on a Microsoft Windows style network.
- **samba_enable_home_dirs** Supports the sharing of user home directories.
- **samba_export_all_ro** Allows files and directories to be shared in read-only mode.
- **samba_export_all_rw** Allows files and directories to be shared in read/write mode.

- **samba_run_unconfined** Allows Samba to run scripts stored in the /var/lib/samba/scripts directory; it has the samba_unconfined_script_exec_t tag.
- **samba_share_fusefs** Supports sharing of filesystems mounted under FUSE filesystems (fusefs).
- **samba_share_nfs** Supports sharing of filesystems mounted under NFS.
- **use_samba_home_dirs** Enables the use of applications shared from remote Samba servers on local home directories.
- **virt_use_samba** Allows virtual machines to use files shared from Samba.

SMTP

The two SELinux booleans associated with SMTP services both work with the default Postfix server. The httpd_can_sendmail boolean was previously described. The other Postfix boolean is enabled by default:

- **allow_postfix_local_write_mail_spool** Supports sharing of filesystems mounted under NFS.

Be aware, if you use the Exim SMTP service, a different set of SELinux booleans are available. Nevertheless, Exim is not included on the RHEL 6 DVD.

SSH

The two SELinux booleans associated with SSH connections, in my opinion, should never be enabled. Fortunately, neither option is enabled by default:

- **allow_ssh_keysign** Allows host-based authentication; would not require usernames or public/private passphrase based authentication.
- **ssh_sysadm_login** Supports access by users configured with the sysadm_r role. This does not include the root administrative user; in general, it's more secure to log in as a regular user, connecting with passphrases, before authenticating with administrative privileges.

SELinux File Contexts

Changes made with the **chcon** command are not permanent. While they do survive a reboot, they do not survive a *relabel*. SELinux relabels of a system can

happen when SELinux is disabled and then re-enabled. The **restorecon** command relabels a target directory. The configured SELinux contexts are stored in the /etc/selinux/targeted/contexts/files directory.

The default version of this directory includes three files:

- **file_contexts** Baseline file contexts for the entire system
- **file_contexts.homedirs** File contexts for the /home directory, and all subdirectories
- **media** File contexts for removable devices that may be mounted after installation

If you need a change to file system contexts to survive a reboot, the **semanage** command can help. Specifically, if you need to set up the /www directory for virtual web sites, the following command makes sure the file contexts are appropriate for that directory (and subdirectories) even after a relabel:

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

The noted command creates a file_contexts.local file in the /etc/selinux/targeted/contexts/files directory.

While the **semanage** command manages a variety of SELinux policies, the focus here is on file contexts, as represented by the **fcontext** option. The switches shown are described in Table 11-3.

TABLE 11-3 Command Switches for semanage fcontext

Switch	Description
-a	Add
-d	Delete
-D	Delete all
-f	File type
-l	List
-m	Modify
-n	No heading
-r	Range
-s	SELinux user name; used for user roles
-t	SELinux file type

EXERCISE 11-1

Configure a New Directory with Appropriate SELinux Contexts

In this exercise, you'll set up a new directory, /ftp, with SELinux contexts that match the standard directory for FTP servers. This exercise demonstrates how this is done with the **chcon** command, along with the effect of the **restorecon** and **semanage** commands.

1. Create the /ftp directory. Use the **ls -Zd /ftp** command to identify the SELinux contexts on that directory. Contrast that with the contexts on the /var/ftp directory.
2. Change the contexts on the /ftp directory to match those on the /var/ftp directory. The most efficient method is with the following command:

   ```
   # chcon -R --reference /var/ftp /ftp
   ```

   While the **-R** switch is not required, I include it to help you get used to the idea of changing contexts recursively.
3. Run the **ls -Zd /ftp** command to review the changed contexts on that directory. It should now match the contexts on the /var/ftp directory.
4. Run the following command to see what happens when SELinux is relabeled.

   ```
   # restorecon -R /ftp
   ```

   What did this command do to the contexts of the /ftp directory? (If desired, include the **-v** switch; if successful, it specifies the changes that have been made.)
5. To make changes to the /ftp directory permanent, you need help from the **semanage** command, with the **fcontext** option. As there is no analogue to the **chcon --reference** command switch, the following command specifies the user role and file type, based on the default settings for the /var/ftp directory:

   ```
   # semanage fcontext -a -s system_u -t public_content_t /ftp/*
   ```

6. Review the results. First, the **semanage** command does not change the current SELinux contexts of the /ftp directory. Next, review the contents of file_contexts.local in the /etc/selinux/targeted/contexts/files directory. It should reflect the **semanage** command just executed.
7. Rerun the **restorecon** command from step 4. Does it change the SELinux contexts of the /ftp directory now?

CERTIFICATION OBJECTIVE 11.03

The Secure Shell Server

Red Hat Enterprise Linux installs the Secure Shell (SSH) server packages by default, using the openssh-server, openssh-clients, and openssh RPMs. Chapter 2 addressed SSH client programs including **ssh**, **scp**, and **sftp**. The focus of this section is on the SSH server. The secure daemon, **sshd**, listens for all inbound traffic on TCP port 22. The SSH server configuration files are located in the /etc/ssh directory.

As SSH is an important tool for administering systems remotely, it's important to understand the basics of how it encrypts communication between a client and the SSH server. Then you'll see how to create a public/private keypair, so connections won't even put passwords at risk. Finally, you'll examine how to configure the SSH server configuration file in detail. But first, it may be helpful to review some basic information about SSH configuration commands and files.

SSH Configuration Commands

There are a few SSH-oriented utilities you need to know about:

- **sshd** The daemon service; this must be running to receive inbound Secure Shell client requests.
- **ssh-agent** A program to hold private keys used for Digital Signature Algorithm (DSA) and Rivest, Shamir, Adelman (RSA) authentication. The idea is that the **ssh-agent** command is started in the beginning of an X session or a login session, and all other windows or programs are started as clients to the **ssh-agent** program.
- **ssh-add** Adds RSA identities to the authentication agent, **ssh-agent**.
- **ssh** The Secure Shell command, **ssh**, is a secure way to log in to a remote machine, similar to Telnet or **rlogin**. The basic use of this command was discussed in Chapter 2. To make this work securely, you need a private key on the server and a public key on the client. Take the public key file, identity.pub or id_dsa.pub, created later in this section. Copy it to the client. Place it in the home directory of an authorized user, in the ~/.ssh/authorized_keys file.

- **ssh-keygen** A utility that creates private/public keypairs for SSH authentication. The **ssh-keygen -t *keytype*** command will create either *keytype*, DSA or RSA.
- **ssh-copy-id** A command that transmits a public key to a target remote system.

All you need to do is transfer the public key, with the .pub extension, to an authorized user. You can do this directly, using a method such as a USB key. Alternatively, you can transmit that key over the network with the **ssh-copy-id** command. It's important to add a passphrase to protect that digital signature. It's important to protect that passphrase. In the worst case, a cracker could use the passphrase to steal your identity.

SSH Configuration Files

Systems configured with SSH include configuration files in two different directories. For the local system, basic SSH configuration files are stored in the /etc/ssh directory. The functionality of these files are summarized here:

- **moduli** Supports the Diffie-Hellman Group Exchange key method with prime numbers and random key generators.
- **ssh_config** Includes configuration for the local SSH client, discussed in Chapter 2.
- **sshd_config** Specifies the configuration of the SSH server; discussed in detail later in this chapter.
- **ssh_host_dsa_key** Includes the host private key for the local system, based on the DSA algorithm.
- **ssh_host_dsa_key.pub** Includes the host public key for the local system, based on the DSA algorithm.
- **ssh_host_key** Obsolete; private host key for SSH protocol version 1, which has been cracked.
- **ssh_host_key.pub** Obsolete; public host key for SSH protocol version 1, which has been cracked.
- **ssh_host_rsa_key** Includes the host private key for the local system, based on the RSA algorithm.
- **ssh_host_rsa_key.pub** Includes the host public key for the local system, based on the RSA algorithm.

But just as important are the configuration files in each user's home directory, in the .ssh/ subdirectory. Those files configure how the given user is allowed to connect to remote systems. When both DSA and RSA keys are included, the typical user .ssh/ subdirectory includes the following files, which should for the most part be familiar from the previous discussion of files in the /etc/ssh directory.

- **authorized_keys** Includes a list of public keys from remote systems. Users with public encryption keys in this file can connect to remote systems. The system users and names are listed at the end of each public key copied to this file.
- **id_dsa_key** Includes the local private key based on the DSA algorithm.
- **id_dsa_key.pub** Includes the local public key for the system based on the DSA algorithm.
- **id_rsa_key** Includes the local private key based on the RSA algorithm.
- **id_rsa_key.pub** Includes the local public key for the system based on the RSA algorithm.
- **known_hosts** Contains the public RSA keys from remote systems. The first time a user logs in to a system, he's prompted to accept the key. The contents of the remote /etc/ssh/ssh_host_rsa_key.pub file are added to the local known_hosts file at that time.

Older RHEL systems included id_dsa.keystore and id_rsa.keystore files with copies of both public and private keys. That is no longer necessary.

Basic Encrypted Communication

Basic encryption in computer networking normally requires a private key and a public key. The principle is the same as GPG communications discussed in Chapter 10. The private key is stored on the server, and the public key is sent to administrative workstations. When the pair is properly configured, administrators on those workstations can connect to the noted servers remotely. Data that is sent through an SSH connection is encrypted with the public key. The SSH server can unscramble the message with the private key.

Encryption keys are based on random numbers. The numbers are so large (typically 768 bits or more) that the chance that someone will break into the server system, at least with a PC, is quite small. Private and public encryption keys are based on a matched set of these random numbers.

Private Keys

The private key must be secure. It is accessible only to the user owner of that key, in the .ssh subdirectory of that user's home directory. Anything that is sent over the SSH connection is then digitally signed and encrypted with the public key. Only the recipient, in this case, the SSH server, will be able to decrypt the message.

Public Keys

The public key is just that, publicly available. Public keys created in this chapter are exclusive to the SSH server in use. They're designed to be copied to appropriate user's .ssh/ subdirectories, in files with .pub extensions. The public keys for SSH servers belong on administrative workstations, so systems managers like yourself can connect to those systems remotely.

The example shown in Figure 11-2 lists the directories and files associated with SSH usage as well as a public key that has been added to the local "keyring."

This key is like a password used to encrypt communications data. But it's not a standard password by any means. Imagine trying to remember the 1024-bit number expressed in hexadecimal format as shown here:

```
3081 8902 8181 00D4 596E 01DE A012 3CAD 51B7
7835 05A4 DEFC C70B 4382 A733 5D62 A51B B9D6
29EA 860B EC2B 7AB8 2E96 3A4C 71A2 D087 11D0
E149 4DD5 1E20 8382 FA58 C7DA D9B0 3865 FF6E
88C7 B672 51F5 5094 3B35 D8AA BC68 BBEB BFE3
9063 AE75 8B57 09F9 DCF8 FFA4 E32C A17F 82E9
7A4C 0E10 E62D 8A97 0845 007B 169A 0676 E7CF
5713 1423 96E0 8E6C 9502 0301 0001
```

That is why the applications save this value for you, on a "public keyring." You can add as many public keys from other users, sites, and services as needed.

FIGURE 11-2

Keys in a User's .ssh/ subdirectory

```
michael@Maui:~$ ls -l .ssh/
total 44
-rw-------. 1 michael michael  1822 Feb  8 16:55 authorized_keys
-rw-------. 1 michael michael   736 Jun  5  2008 id_dsa
-rw-r--r--. 1 michael michael  3576 Jun 17  2008 id_dsa.keystore
-rw-r--r--. 1 michael michael   606 Jun  5  2008 id_dsa.pub
-rw-------. 1 michael michael  1743 Feb  5  2010 id_rsa
-rw-r--r--. 1 michael michael  2506 Feb  5  2010 id_rsa.keystore
-rw-r--r--. 1 michael michael   398 Feb  5  2010 id_rsa.pub
-rw-r--r--. 1 michael michael 15162 Jan 21 09:05 known_hosts
michael@Maui:~$
```

exam Watch

The RHCE objectives for the first time specify that you need to know how to "configure key-based authentication" for SSH.

The private key is similar, *but you must keep it private*, or this whole system fails. Keeping it private means no one should have access to the server systems. If your PC is public, secure your system with a passphrase (password). The procedure to set up a passphrase is described next. Don't forget the passphrase, or you'll have to remove and reinstall the Secure Shell.

Set Up a Private/Public Pair for Key-Based Authentication

The **ssh-keygen** command is used to set up a public/private key pair. While it creates an RSA key by default, it also can be used to create DSA keys. For example, some users may need DSA keys to comply to certain U.S. government standards. An example of the command sequence is shown in Figure 11-3.

As shown in the figure, the command prompts for a passphrase, twice. When the identical passphrase is confirmed, the private key is saved in the id_rsa file, and the corresponding public key is stored in the id_rsa.pub file. Both files for user michael are stored in the /home/michael/.ssh directory.

FIGURE 11-3

Commands to generate encryption keys

```
[michael@server1 ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/michael/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/michael/.ssh/id_rsa.
Your public key has been saved in /home/michael/.ssh/id_rsa.pub.
The key fingerprint is:
c4:60:26:5a:38:53:e9:22:c8:97:a7:51:d6:16:f9:ff michael@server1.example.com
The key's randomart image is:
+--[ RSA 2048]----+
|    o+.=.o       |
|   +o.* *        |
|o .++ . +        |
|o..+.. . .       |
| ...+   S .      |
|   .       .     |
|            .    |
|             E   |
|                 |
+-----------------+
[michael@server1 ~]$
```

If desired, you can set up RSA keys with a larger number of bits. In my testing, I was able to set up key pairs with up to 8192 bits fairly quickly, even on a 768MB virtual machine system. The command which starts the process is

```
$ ssh-keygen -b 8192
```

Alternatively, if a DSA key is needed, the following command can help. Unfortunately, only 1024-bit DSA keys are allowed. The process after this command is the same as shown in Figure 11-3.

```
$ ssh-keygen -t dsa
```

The next step is to transmit the public key to a remote system. It might be one of the servers that you administer. If you're willing to transmit that public key over the network (once per connection), the following command can work:

```
$ ssh-copy-id -i .ssh/id_rsa.pub michael@tester1.example.com
```

Strictly speaking, the **-i** option defaults to transmitting the .ssh/id_rsa.pub key. It's included in the command for clarity. The command automatically appends the noted local RSA public key to the end of the *remote* .ssh/authorized_keys file. In this case, that file can be found in the /home/michael directory. Of course, you may choose to substitute the IP address for the hostname.

exam Watch

Sometimes, the public key is not immediately accessible on the remote system. If you get an "agent admitted failure to sign using the key" error followed by a password prompt, log out of the console or the GUI and log back in. In most cases, the ssh command will prompt for the passphrase.

You should then be able to immediately connect to that remote system. In the preceding case, the appropriate command is either one of the following:

```
$ ssh -l michael tester1.example.com
$ ssh michael@tester1.example.com
```

When run on a console, the **ssh** command uses the following prompt for the passphrase:

```
Enter passphrase for key '/home/michael/.ssh/id_rsa'
```

When run in a GUI-based command line, it prompts with a window similar to that shown in Figure 11-4.

If the public key is available to all, there should be no hesitation about sending it over the network. However, it does mean transmitting the public key over the network, once. As that's a "public" key, that shouldn't cause heartburn. However, you may want to limit access to that pubic key. To do so, you can use a USB key or other portable media to physically carry the public key to the target remote system. You'd then copy the contents of the public key to the end of the target user's .ssh/authorized_keys file.

Configure an SSH Server

You don't have to do much to configure an SSH server for basic operation. Install the packages described earlier, activate the service, make sure it's active the next time the system is rebooted. As discussed in Chapter 1, the standard SSH port 22 is open in the default RHEL 6 firewall.

However, the RHCE objectives specify that you should be prepared to "configure additional options described in documentation." Because of the general nature of that objective, this section will address every active and commented option in the default version of the SSH server configuration file.

The SSH server configuration file is /etc/ssh/sshd_config. The commands in comments are generally defaults. So if you want to set a nonstandard port for the SSH service, you could change the following commented directive

```
#Port 22
```

to something like

```
Port 2222
```

FIGURE 11-4

Prompt for a passphrase

Assuming the firewall allows access through this port, you'd then be able to connect from a remote system with the **ssh -p 2222 server1.example.com** command. If the SSH server is different, substitute for server1.example.com.

While the default shown here listens for both IPv4 and IPv6 addresses,

```
#AddressFamily any
```

it's possible to limit access to one of these types of addresses, where **inet** corresponds to IPv4 and **inet6** corresponds to IPv6.

```
AddressFamily inet
AddressFamily inet6
```

The default shown with the following ListenAddress directives is to listen for SSH communications on all local IPv4 and IPv6 addresses:

```
#ListenAddress 0.0.0.0
#ListenAddress ::
```

You can limit SSH to listening on the IPv4 or IPv6 addresses of certain network cards. That can help limit access to the SSH server to certain networks.

The first active directive configures SSH version 2. As noted earlier, SSH version 1 has been cracked and is therefore insecure.

```
Protocol 2
```

Since SSH version 1 is not used, you should not have to activate the following directive:

```
#HostKey /etc/ssh/ssh_host_key
```

The standard RSA and DSA keys are documented here; generally, there's no reason to change their locations:

```
#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_dsa_key
```

The commented directives that follow relate to an SSH version 1 key. Such keys would be regenerated every hour, with 1024 bits. But that would still be insecure.

```
#KeyRegenerationInterval 1h
#ServerKeyBits 1024
```

The first directive sends all logging attempts, successful and otherwise, to the appropriate log file; in this case, /var/log/secure. It's as defined in the /etc/rsyslog.conf file. The level of information is INFO and above.

```
SyslogFacility AUTHPRIV
#LogLevel INFO
```

To limit unauthorized password attempts, the default LoginGraceTime shown here is two minutes. In other words, if a connection has not been completed in that time, the SSH server automatically disconnects from the remote client.

```
#LoginGraceTime 2m
```

The directive that follows documents that the root administrative user can log in from a remote location.

```
#PermitRootLogin yes
```

Direct root logins over SSH can be inherently insecure. If you've set up private/public key-based passphrase authentication from an administrative account on a laptop system, that's a risk. A cracker who gets a hold of that laptop system might then be able to connect to the remote server with administrative privileges. For that reason, I always change that directive to

```
PermitRootLogin no
```

But if that's not a requirement when you take the RHCE exam, don't make that change. In fact, it could be counted as an error on the exam.

Administrators who log in as regular users can use the **su** or **sudo** command as appropriate to take administrative privileges with fewer risks. Next, it's more secure to retain the following directive, especially with respect to private and public keys:

```
#StrictModes yes
```

As noted with the following directive, the default number of authentication attempts per connection is six. You could reduce that number, for users who know their passphrases:

```
#MaxAuthTries 6
```

The following directive suggests that you could open up to ten SSH connections on different consoles:

```
#MaxSesssions 10
```

The following directive is used only with SSH version 1. Hopefully, you didn't activate that version of SSH.

```
#RSAAuthentication yes
```

On the other hand, the following directive is critical if you want to set up passphrase-based private/public key authentication on the standard SSH version 2:

```
#PubkeyAuthentication yes
```

The following directive confirms the use of the authorized_keys file on the remote system to confirm public keys for authentication:

```
#AuthorizedKeysFile .ssh/authorized_keys
```

The two directives that follow are typically ignored:

```
#AuthorizedKeysCommand none
#AuthorizedKeysCommandRunAs nobody
```

The following Rhosts directive is generally not used, as it applies to SSH version 1, and the less-secure Remote Shell (RSH):

```
#RhostsRSAAuthentication no
```

While the following directive could support the use of the /etc/hosts.equiv file to limit hosts that connect, that's not normally encouraged. Nevertheless, it is one method for SSH host-based security, beyond what's possible with an alternative such as TCP wrappers discussed in Chapter 10.

```
#HostbasedAuthentication no
```

As described earlier, the .ssh/known_hosts file stores RSA keys from remote systems, and is read because of the following default:

```
#IgnoreUserKnownHosts no
```

The following directive may help administrators who are converting from RSH to SSH, as they use .rhosts and .shosts files. But as it's not used by default, the following option is sensible:

```
#IgnoreRhosts yes
```

For systems and users where private/public passphrases aren't used, password-based authentication is needed, as enabled by this default:

```
#PasswordAuthentication yes
```

In general, you should never permit empty passwords, due to the security risks:

```
#PermitEmptyPasswords no
```

Challenge-response authentication is normally associated with one-time passwords common with remote terminals. While it can also work with PAM, it is normally disabled on SSH:

```
ChallengeResponseAuthentication = no
```

If you do set up a Kerberos system for the local network, as discussed in Chapter 12, you could set up some of the following options. The first two are almost self-explanatory, as they can enable Kerberos verification of a user and set up alternative Kerberos or local password authentication.

```
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
```

The next two options relate to Kerberos authentication tickets and associated Andrew File System (AFS) tokens:

```
#KerberosTicketCleanup yes
#KerberosGetAFSToken no
```

This is followed by a directive that supports authentication using the Generic Security Services Application Programming Interface (GSSAPI) for client/server authentication:

```
GSSAPIAuthentication = yes
```

The following directive destroys GSSAPI credentials upon logout.

```
GSSAPICleanupCredentials = yes
```

Normally, hostname checks are strict:

```
GSSAPIStrictAcceptorCheck = yes
```

But normally authentication doesn't rely on SSH keys, as described here:

```
GSSAPIKeyExchange = yes
```

Access to PAM modules is supported:

```
UsePAM yes
```

The following directives allow the client to set several environmental variables. The details are normally trivial between two Red Hat Enterprise Linux systems:

```
AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
AcceptEnv LC_IDENTIFICATION LC_ALL
AcceptEnv XMODIFIERS
```

With the following setting, the **ssh-agent** command can be used to forward private keys to other remote systems:

```
#AllowAgentForwarding yes
```

In a similar fashion, TCP communications can be forwarded over an SSH connection:

```
#AllowTCPForwarding yes
```

The GatewayPorts directive is normally disabled to keep remote hosts from connecting to forwarded ports:

```
#GatewayPorts no
```

The following directive is important for anyone who needs remote access to a GUI tool:

```
X11Forwarding yes
```

For example, when I'm working from a remote location, I can connect to and open GUI tools from Red Hat systems in my office when I use SSH to connect with the following command:

```
# ssh -X michael@Maui.example.com
```

The next directive helps avoid conflicts between local and remote GUI displays. The default should be adequate, unless there are more than ten GUI desktop environments open on the SSH server system.

```
#X11DisplayOffset 10
```

Normally, no changes are required to the following default, related to how the GUI display is bound on the SSH server:

```
#X11UseLocalhost yes
```

When SSH users log in remotely, the following setting means they see the contents of the /etc/motd file. Different messages are possible, based on the cron script configured in Chapter 9.

```
#PrintMotd yes
```

This is one useful setting for administrators, as it documents the date and time of the last login to the noted system:

```
#PrintLastLog yes
```

The **TCPKeepAlive** directive can keep a system from crashing if a network connection, the SSH server, or any connected SSH client goes down:

```
#TCPKeepAlive yes
```

Generally, you should not enable this option, as it is incompatible with **X11Forwarding**:

```
#UseLogin no
```

The privilege separation associated with the following directive sets up separate processes for the user and for network communication:

```
#UsePrivilegeSeparation yes
```

The following directive does not supersede the default **AuthorizedKeysFile** setting earlier in the file:

```
#PermitUserEnvironment no
```

Compression often helps communications over an SSH connection. The default is to delay compression until the password is accepted or the private/public key pair is matched to authenticate the user:

```
#Compression delayed
```

Sometimes, it's important to have the SSH server make sure the user still wants to transmit data. It's how clients are disconnected from sensitive systems such as bank accounts. But for an administrative connection, the following option disables such checks:

```
#ClientAliveInterval 0
```

If the **ClientAliveInterval** is set to some number, the following directive specifies the number of messages that may be sent before that client is automatically disconnected:

```
#ClientAliveCountMax 3
```

The following option for a patch level applies only to SSH version 1:

```
#ShowPatchLevel no
```

To minimize the risks of spoofing, the following option checks remote hostnames against a DNS server or an /etc/hosts file:

```
#UseDNS yes
```

The PID file listed here contains the process ID number of the running SSH server process:

```
#PidFile /var/run/sshd.pid
```

When a cracker tries to break into a SSH server, he may try to set up a bunch of terminals, all attempting to log in simultaneously. The following directive limits the number of terminals that the SSH server will work with. For a SSH server on an administrative system, it's something that you might consider reducing.

```
#MaxStartups 10
```

The following directive, if activated, would support device forwarding:

```
#PermitTunnel no
```

The following directive may seem like good idea but could be difficult to put into practice. Any directory specified should contain all of the commands, devices, and configuration files within that directory tree:

```
#ChrootDirectory none
```

The final directive supports the use of SSH encryption for secure FTP file transfers:

```
Subsystem  sftp   /usr/libexec/openssh/sftp-server
```

User-Based Security for SSH

User-based security can be configured in the /etc/ssh/sshd_config file. To that end, I like to add directives that limit the users allowed to access a system via SSH. The key is the **AllowUsers** directive. You can limit by user with a directive such as

```
AllowUsers michael donna
```

Alternatively, you can limit access by each user to certain hosts with a directive such as the following, which combines aspects of both user- and host-based security.

```
AllowUsers michael@192.168.122.50 donna@192.168.122.150
```

Just be aware, if access is coming from a remote network, an **iptables**-based masquerading configuration may assign the IP address of the router to the remote system. In that case, the IP address of the router serves as a proxy to limit access to all systems you won't be able to specify a single system (except the router) on a remote network.

Related directives that can be included in the /etc/ssh/sshd_config file are **AllowGroups**, **DenyUsers**, and **DenyGroups**. Such directives can be used in similar ways to limit the users who connect. But if you want to limit access to SSH to a very few users, the **AllowUsers** directive is all that's needed. For the first **AllowUsers** directive just shown, only users michael and donna can connect to this SSH server. A corresponding **DenyUsers** or **DenyGroups** directive is not required. Even the root user can't connect via SSH under those circumstances.

While the SSH server would prompt other users for a password, access is denied even when the remote user enters the correct password. The /var/log/secure log file would reflect that with something similar to the following message:

```
User dickens from 192.168.122.150 not allowed because not listed in AllowUsers.
```

Host-Based Security for SSH

While there are methods for configuring host-based security through the SSH configuration files, the process is needlessly complex. It involves changes to both servers and clients, and involves risks that I believe are not necessary. It's also possible to set up host-based security through **iptables**-based firewalls.

The simplest method for host-based SSH security is based on TCP Wrappers, as discussed in Chapter 10. For the purpose of this chapter, I've included the following

directive in /etc/hosts.allow, which accepts SSH connections from the noted network addresses.

```
sshd : 127. 192.168.122.
```

To make sure access is limited to systems on the noted networks, it's also important to include the following line in /etc/hosts.deny:

```
sshd : ALL
```

Of course, it would be more secure to include **ALL : ALL** in /etc/hosts.deny. But that may block communications with services that you've worked hard to configure in other ways. In addition, other ports should already be protected by an appropriate **iptables**-based firewall. So it may be an option to avoid during a Red Hat exam.

CERTIFICATION OBJECTIVE 11.04

A Security and Configuration Checklist

A number of steps required to install, configure, and secure a service are repetitive. I therefore summarize them in this section. If desired, you can use this section to help prepare for Chapters 12 through 17. It will help you install required services, make sure those services are active, and make sure those services are accessible through a firewall configured with appropriate open ports.

Installation of Server Services

The RHCE objectives directly address eight different services. This section addresses some of the different ways that you can install these services. If you've read Chapter 7, this should be mostly review. But it will also give you an opportunity to prepare a system such as the server1.example.com virtual machine for testing in Chapters 12 through 17.

In this section, you'll review commands like **rpm** and **yum**, in the context of the server services needed for upcoming chapters. If you prefer to use the GUI Add/Remove Software application, refer to Chapter 7. Generally, you can use any of these options to install desired services.

Install the vsFTP Server with the rpmCommand

In general, the installation of most services require the installation of more than one RPM package. One exception is the RPM package associated with the vsFTP server. To that end, if you've mounted the RHEL 6 DVD on the /media directory, you can install the vsFTP server with the following command (the version number may vary):

```
# rpm -ivh /media/Packages/vsftpd-2.2.2-6.el6.x86_64.rpm
```

Install Server Services with the yumCommand

As discussed in Chapter 7, the **yum** command can be used to install packages with dependencies. Sometimes, dependencies are simple. For example, for the e-mail services configured in Chapter 13, you may be more familiar with open-source sendmail, as opposed to the default Postfix SMTP service.

Of course, you'd also have to make sure the focus is on sendmail, but that's a detailed configuration topic discussed in Chapter 13. One way to install the sendmail package with dependencies is with the following command:

```
# yum install sendmail
```

As needed, you can use the **yum install** command to install a package in a way that automatically identifies and also installs all dependent packages.

Install Server Package Groups with the yumCommand

Also noted in Chapter 7 is the way RHEL 6 packages are organized in groups. Each of those groups have names, which can be identified with the **yum grouplist** command. The relevant groups for the RHCE exam are listed in Table 11-4. The capitalization of each package group is as shown in in the output to the **yum grouplist** command.

You can identify different packages in each group with the **grouplist** switch; for example, the following command lists the different packages that are part of the Web Server package group:

```
# yum grouplist "Web Server"
```

The output for RHEL 6 is shown in Figure 11-5.

Packages are classified in three categories: mandatory, default, and optional. If you run the following command:

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

TABLE 11-4

RHCE-Related Server Package Groups

Package Group	Description
CIFS file server	Package group for the Samba file server.
E-mail server	Support packages for SMTP and Internet Message Access Protocol (IMAP) services; the default services are Postfix and Dovecot. The sendmail server is an optional package in this group.
FTP server	Package group for the vsFTP server.
iSCSI Storage Client	Client packages for connections to remote iSCSI systems
NFS file server	Setup packages for the NFS server
Network Infrastructure Server	Blanket package group for the DNS and rsyslog servers; all packages in this group are optional.
Network Storage Server	Associated with iSCSI connections, discussed in Chapter 12.
Network file system client	Includes clients for the automounter, Samba, and NFS.
Web Server	Includes basic Apache web server packages; however, packages for CGI applications required in the objectives are listed as optional.

FIGURE 11-5

Packages in the Web Server package group

```
Group: Web Server
 Description: Allows the system to act as a web server, and run Perl and Pytho
web applications.
 Mandatory Packages:
   httpd
 Default Packages:
   crypto-utils
   httpd-manual
   mod_perl
   mod_ssl
   mod_wsgi
   webalizer
 Optional Packages:
   certmonger
   libmemcached
   memcached
   mod_auth_kerb
   mod_auth_mysql
   mod_auth_pgsql
   mod_authz_ldap
   mod_nss
   perl-CGI
   perl-CGI-Session
   perl-Cache-Memcached
   python-memcached
   squid
[root@Maui ~]#
```

Only packages in the mandatory and default categories are installed. In most cases, that's not a problem. But the RHCE objectives related to web servers suggest the need to "deploy a basic CGI application." And the CGI-related packages are listed as optional. While there are ways to set up the installation of optional packages with the **groupinstall** switch, it's easier for our purposes to just install needed packages separately by name. For web servers, the appropriate package for handling CGI scripts is covered in Chapter 14.

In a similar fashion, you can install the Samba File Server covered in Chapter 15 with the following command:

```
# yum groupinstall "CIFS file server"
```

There's actually just one package in this group, samba. For that reason, the following command would work equally well:

```
# yum install samba
```

For Chapter 17, the Network Infrastructure Server package group includes packages associated with logging and DNS. However, as all packages in this group are optional, the **yum groupinstall** command would not install any packages from that group. Fortunately, the rsyslog package is already installed by default, even in a minimal RHEL 6 installation. But you will want to install DNS to address one of the RHCE objectives. One way to set up the DNS service for Chapter 17 is with the following command:

```
# yum install bind
```

For a number of server services, you should make sure that appropriate client packages are installed. The Network file system client package group can help in that respect; the following command would install clients for the automounter, Samba, and NFS:

```
# yum groupinstall "Network file system client"
```

A different kind of network server relates to iSCSI storage. There are two package groups of interest: the Network Storage Server and the iSCSI Storage Client. They each are associated with a single package, which will be discussed in Chapter 12.

Finally, there are a couple of packages of interest that are not included in standard package groups. They set up the NTP server and authentication to remote databases. If not already installed, you'll need to install them. One method is with the following command:

```
# yum install ntp sssd
```

I focus on command line installation methods, as they are generally fastest. Of course, you could install packages with the GUI Add/Remove Software tool discussed in Chapter 7.

Basic Configuration

While the current RHCE objectives are more specific than ever, it's best to keep what you change as simple as possible. As noted in the objectives, you'll be asked to "configure the service for basic operation." Basic operation is simpler. It is frequently more secure. If you do less to configure a service, it takes less time. You'll have a better chance to finish the exam. You'll be able to do more on the job.

Of course, the details associated with basic configuration are covered in upcoming chapters.

Make Sure the Service Survives a Reboot

In Chapter 5, you looked at when a service starts or does not start during the boot process. The simplest method is associated with the **chkconfig** command. To review, the **chkconfig *--list*** command lists whether a service controlled by scripts in the /etc/init.d directory. For the services discussed in the following chapters, once the appropriate packages have been installed, you'll want to make sure they start during the boot process with the following commands:

```
# chkconfig httpd on
# chkconfig iscsi on
# chkconfig iscsid on
# chkconfig named on
# chkconfig nfs on
# chkconfig nmb on
# chkconfig ntpd on
# chkconfig ntpdate on
# chkconfig rpcbind on
# chkconfig rsyslog on
# chkconfig smb on
# chkconfig sshd on
# chkconfig vsftpd on
```

This list is not complete. On the other hand, on an actual exam, while all of these services are "fair game," you may not be required to install them all. For example, if there's no requirement to install an NTP server (or client), you need not

install the ntp package. Without that package, the **chkconfig ntpd on** command would lead to an error message.

Sometimes, there are mutually exclusive alternatives. For example, for SMTP services, RHEL 6 makes it possible to configure either Postfix or sendmail. In Chapter 13, you'll examine a different way to make sure only one service is set to run.

Of course, it is always possible that during an exam, you'll be told to make sure a service does *not* start during the boot process. And in a production environment, the installation of so many services on a single system is rare, because of the security risks.

Review Access Through Layers of Security

The first place to check a service is from the local system. For example, if you can connect to an FTP server from that system, that confirms a good configuration of that service. Some issues relate to SELinux; others relate to various user and host-based firewalls. For issues beyond SELinux, the network command tools installed in Chapter 2 can be quite helpful.

Troubleshoot SELinux Issues

If the configuration is good but still does not work, that suggests an SELinux issue. Not all SELinux issues are readily visible. However, you can check two basic issues with respect to SELinux:

- The list of boolean settings related to the configuration option in question. For example, users who try to connect to remote home directories through the FTP server can be configured through the main vsFTP configuration file. However, problems don't always appear in the SELinux logs.
- The SELinux file contexts. The contexts of the target directories should match those of default directories. In other words, when you run the **ls -Z** command on a shared FTP directory, the file contexts should match that of the default shared FTP directory.

The next step is to test the connection from a remote system. It's appropriate to test the connection in a number of ways:

1. Can the remote client system communicate with the remote server? That's most easily confirmed with the **ping** command.

2. Can the remote client communicate over the assigned port? That can be confirmed with the **telnet** command. For example, the following command attempts to establish a connection on TCP/IP port 21:

   ```
   $ telnet 192.168.122.50 21
   ```

   If the connection is successful, the vsFTP server is identified by name. Type in the **quit** command and press ENTER to exit from the connection.
3. If the **telnet** connection to the appropriate port does not work, then there may be a firewall problem. There are actually subtle differences
   a. Firewalls exist on several levels. The **iptables** command firewall discussed in Chapters 4 and 10, as configured in /etc/sysconfig/iptables, must support access through the noted port number and can't limit it from desired hosts.
   b. The TCP Wrappers system works with daemons associated with the libwrap.so.0 library, allowing and limiting access in /etc/hosts.allow and /etc/hosts.deny.

If all firewalls are clear, and the service still does not work as intended, then there may be a configuration problem with the server. Some servers include their own limits, based on hosts, IP addresses, and user names.

Troubleshoot iptables-Based Firewall Issues

In general, **iptables**-based firewalls are straightforward. If a system allows access for server communications, you'll see it in the port numbers documented in the /etc/sysconfig/iptables file and the output to the **iptables -L** command.

Specifically, you should expect to see a line like the following in /etc/sysconfig/iptables to allow communications through the standard port for the SSH server:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
```

When the iptables service is started with the **/etc/init.d/iptables start** command, the file is read into memory; you should be able to see the following excerpt from the **iptables -L** command for the noted port:

```
ACCEPT  tcp  --  anywhere     anywhere     state NEW tcp dpt:ssh
```

Let's translate that line into something closer to English. This rule accepts packets. It checks the packet for the use of the TCP protocol. It then checks the IP addresses of the header but allows packets that come from any address, with a destination of

any address. The packet is new, with a TCP destination port (**dpt**) associated with **ssh**. The /etc/services file translates **ssh** to the standard SSH port of 22.

Of course, you could just use the Firewall Configuration tool described in Chapters 4 and 10, but what if those tools are flawed? The bottom line for iptables-based firewalls is based on the noted file and the way it's implemented with the **iptables -L** command.

Unless this port is open in the firewall, an attempt to connect to that server is rejected with a message like the following:

```
ssh: connect to host server1.example.com port 22: No route to host
```

In contrast, the **telnet server1.example.com 22** command leads to a similar message:

```
telnet: connect to address 192.168.122.50: No route to host
```

Troubleshoot TCP Wrappers Firewall Issues

In contrast, if the service is protected by TCP wrappers, the error message behavior is different. For this section, I configured the /etc/hosts.allow and /etc/hosts.deny files on the server1.example.com system to allow access only from .example.com systems on the 192.168.122.0/24 network. That means access is not allowed from systems such as outsider1.example.org on IP address 192.168.100.100.

In that case, when I tried accessing the server1.example.com system with the ssh command, I received the following error message:

```
ssh_exchange_identification: Connection closed by remote host
```

In contrast, the **telnet server1.example.com 22** command from the same system returns the following messages, which stops for a moment:

```
Trying 192.168.122.50
Connected to server1.example.com.
Escape character is '^]'
```

For a few moments, it appears the system is about to connect. But then the block from TCP Wrappers results in the following message:

```
Connection closed by foreign host.
```

EXERCISE 11-2

Review the Different Effects of iptables and TCP Wrappers

This exercise assumes an operational vsFTP server, similar to the one configured in Chapter 1 for installations. It is assumed that you'll configure that vsFTP server on the server1.example.com system. You'll then make sure the firewall blocks traffic on the standard vsFTP port, 21, before checking the effect from a remote blocked system, outsider1.example.org. To review, these systems as configured in Chapters 1 and 2 are on IP addresses 192.168.122.50 and 192.168.100.100, respectively.

Then you'll open up the firewall on port 21 and then limit access using TCP Wrappers. There are a lot of steps in this exercise; in fact, each numbered step requires several commands or actions. In some cases, the required command is implied. For example, since there are two versions of the Firewall Configuration tool, the exact steps required would differ.

1. If it is not already installed, install the vsFTP server, as discussed in the chapter. Make sure that server is active with the **/etc/init.d/vsftpd start** command.
2. Start the Firewall Management tool with the **system-config-firewall** command. Make sure FTP is not activated in the list of Trusted Services. Make sure the changes are applied and then exit from the Firewall Management tool.
3. Try connecting to the vsFTP server from the local system with a command like **lftp localhost**. It should work, which you can confirm from the lftp localhost:/> prompt with the **ls** command. Exit from the vsFTP server with the **quit** command.
4. Move to the outsider1.example.org system. It's acceptable to connect to it via SSH; in fact, that may be the only method available to connect to that system on an exam (and in real life).
5. Try pinging the system with the vsFTP server with the **ping 192.168.122.50** command. Remember to press CTRL-C to stop the process. Try connecting to the vsFTP server with the **lftp 192.168.122.50** command. What happens? Try to connect to the system with the **telnet 192.168.122.50 21** command. What happens?
6. Return to the server1.example.com system. Open the Firewall Configuration tool again, and this time, make FTP a trusted service. Don't forget to apply the change, before exiting from the Firewall Configuration tool.

7. Open the /etc/hosts.allow file, and include the following entry:

```
vsftpd : localhost 127. 192.168.122.150
```

8. Open the /etc/hosts.deny file, and include the following entry:

```
vsftpd : ALL
```

9. Return to the outsider1.example.com system as discussed in Step 4. Repeat Step 5. What happens after each attempt to connect?
10. Go back to the server1.example.com system. Open the /etc/hosts.allow and /etc/hosts.deny files, and delete the lines created in Steps 7 and 8.
11. Once again, move to the outsider1.example.org system. Repeat Step 5. Both commands should result in a successful connection. The **quit** command should exit in both cases.
12. BONUS: review connections via the contents of the /var/log/secure file. Review the originating IP addresses in that file. Use that information to configure **iptables** to deny access to all but one IP address.

SCENARIO & SOLUTION

You want to limit SSH access to two users.	Specify the desired usernames in the SSH server configuration file, /etc/ssh/sshd_config, with the AllowUsers directive.
You're told to limit SSH access to systems on the 192.168.122.0/24 network	Use TCP Wrappers. Configure /etc/hosts.allow to allow access to the **sshd** daemon from systems on the noted network. Configure /etc/hosts.deny to restrict access to **sshd** from **ALL** systems.
U.S. Government contracts require SSH passphrase compliance	Set up passphrases with the **ssh-keygen -t dsa** command to set up a private/public key pair that uses DSA encryption.
You need to make sure SELinux user and file types survive a relabel	Use the **semanage fcontext -a** command to specify the desired user and file types for desired directories.
A server is accessible only locally	Check security options for **iptables** command firewall rules, TCP Wrappers; make sure the service allows remote access.
A server is properly configured but still is not accessible	Check for SELinux booleans and file label types

CERTIFICATION SUMMARY

This chapter focused on the general steps required to configure, secure, and access various services. Daemons are controlled by scripts in the /etc/init.d directory, which are started based on parameters in various /etc/sysconfig files. Access to various aspects of server services may be controlled by different SELinux booleans.

The information was tested on the SSH server service. With the **ssh-keygen** command, you can create private/public key passphrase-protected pairs that can keep users from having to transmit passwords over a network. The sshd_config configuration file includes a substantial number of options for configuring that service.

To configure a service, you'll need to install the right packages and make sure the services are active after the next reboot. You'll also need to navigate through the variety of available security options, including SELinux, **iptables**-based firewalls, and TCP Wrappers–based security in the /etc/hosts.allow and /etc/hosts.deny files.

TWO-MINUTE DRILL

The following are some of the key points from the certification objectives in Chapter 11.

Red Hat System Configuration

- ❑ System services can be started with scripts in the /etc/init.d directory or with the **service** command.
- ❑ System services use basic configuration files in the /etc/sysconfig directory. Such files often include basic parameters for service daemons.
- ❑ When configuring a network server, you'll need to be concerned about SELinux booleans, **iptables**-based firewalls, TCP Wrappers, and more.
- ❑ Services should be tested locally and remotely.

Security-Enhanced Linux

- ❑ Individual services are frequently protected by multiple SELinux booleans.
- ❑ SELinux booleans are stored in the /selinux/booleans directory, with descriptive filenames.
- ❑ SELinux booleans can be changed with the **setsebool -P** command or the SELinux Management tool. From the command line, make sure to use the **-P** switch, or the change won't survive a reboot.
- ❑ SELinux file contexts can be changed with the **chcon** command. However, the change does not survive a relabel unless documented with the **semanage fcontext -a** command. Changes are documented in the file_contexts.local file, in the /etc/selinux/targeted/contexts/files directory.

The Secure Shell Package

- SSH server configuration commands include **sshd**, **ssh-agent**, **ssh-add**, **ssh-keygen**, **ssh-copy-id**, and **ssh**.
- SSH server configuration files in the /etc/ssh directory include client and server configuration files, along with public and private host keys, encrypted with RSA and DSA formats.
- User home directories include their own .ssh subdirectory of configuration files, with private and public identification keys, suitable for passphrases.
- Private/public keypairs can be configured with passphrases with the **ssh-keygen** command.
- Public keys can be transmitted to the home directory on remote systems with the **ssh-copy-id** command.
- The SSH server configuration file, sshd_config, can be set up with user-based security.
- The easiest way to set up host-based SSH security is through TCP Wrappers.

A Security and Configuration Checklist

- You'll need to install a number of services to prepare for the RHCE exam, with commands like **rpm** and **yum**.
- One way to make sure services survive a reboot is with the **chkconfig** command; a full list of such commands related to RHCE services are listed in the chapter.
- You'll need to configure access to a service through layers of security, including SELinux, **iptables**-based firewalls, and TCP Wrappers.

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.

Red Hat System Configuration Files

1. What is the name of the configuration file for iptables-based firewalls in the /etc/sysconfig directory?

   _____

2. What command is equivalent to the **/etc/init.d/smb reload** command?

   _____

Security-Enhanced Linux

3. What directory contains boolean options associated with SELinux? Specify the full path.

   _____

4. What man page contains SELinux options associated with NFS daemons?

   _____

5. What command restores the default settings on a given directory?

   _____

6. If the file_contexts file contains SELinux labels for different directories, what file is created with the help of **semanage fcontext -a** command? It's in the /etc/selinux/targeted/contexts/files directory.

   _____

The Secure Shell Server

7. What command configures a private/public key pair using DSA?

   _____

8. What subdirectory of a user home directory contains the authorized_keys file?

_____

9. What directive specifies the port number of the local SSH server in the associated configuration file?

_____

10. What directive specifies a list of allowed users in the SSH server configuration file?

_____

A Security and Configuration Checklist

11. What command displays a list of all available package groups?

_____

12. What command can help the abcd service survive a reboot?

_____

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.

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 Chapter11/ subdirectory. In case you haven't yet set up RHEL 6 on a system, refer to Chapter 1 for installation instructions.

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

SELF TEST ANSWERS

Red Hat System Configuration Files

1. Slight trick question: the file in the /etc/sysconfig directory where **iptables**-command firewalls are configured is iptables.
2. Two commands equivalent to **/etc/init.d/smb reload** are

```
/etc/rc.d/init.d/smb reload
service smb reload
```

Security-Enhanced Linux

3. The directory with SELinux booleans is /selinux/booleans.
4. The nfs_selinux man page contains some SELinux booleans for that service.
5. The command that restores file contexts to a given directory is **restorecon**.
6. The name of the file that is created by the noted command is file_contexts.local.

The Secure Shell Server

7. The command is **ssh-keygen -t dsa**.
8. Every user with public keys stored in the authorized_keys file can find that file in the .ssh subdirectory of her home directories.
9. The directive is **Port**.
10. The directive is **AllowUsers**.

A Security and Configuration Checklist

11. The command that lists all available package groups is **yum grouplist**.
12. Assuming the abcd service is also a script in the /etc/init.d directory, the command that would help it survive a reboot is **chkconfig abcd on**. Variations on this command such as **chkconfig --level 35 on** are acceptable.

LAB ANSWERS

Lab 1

This lab should give you an idea of what can be done with /etc/sysconfig files, and how it changes the way a daemon is started. It should also demonstrate the risks; the wrong change, such as that demonstrated in the lab, means that the service won't work.

Lab 2

There are three measures of success in this lab.

1. There will be an id_rsa file and an id_rsa.pub file in the client /home/hawaii/.ssh directory.
2. You'll be able to connect to the remote system without a password. Just enter the "I love Linux!" passphrase when prompted.
3. You'll find the contents of the client's id_rsa.pub file in the remote authorized_keys file in the /home/hawaii/.ssh directory.

Lab 3

Much as in Lab 2, there are three measures of success in this lab.

1. There will be an id_dsa and an id_dsa.pub file in the client /home/tonga/.ssh directory.
2. You'll be able to connect to the remote system without a password. Just enter the "I love Linux!" passphrase when prompted.
3. You'll find the contents of the client's id_dsa.pub file in the remote authorized_keys file in the /home/hawaii/.ssh directory.

Lab 4

The simplest way to implement this lab is to add the following directive to the /etc/ssh/sshd_config file:

```
AllowUsers hawaii
```

Just don't forget to reload or restart the SSH service after making the change; otherwise, other users will still have access.

In case you're curious, user tonga on the client is still able to access the hawaii account on the SSH server with the passphrase, as it is connections to the user hawaii account that are being allowed. The identity of the remote account does not matter to the AllowUsers directive.

If you've made too many changes to the /etc/ssh/sshd_config file and want to start fresh, move that file and run the **yum reinstall openssh-server** command. It'll set up a fresh copy of that configuration file. If you want to connect from other accounts in the future, make sure the **AllowUsers hawaii** directive is disabled.

Oh yes, did you need to activate the **PermitRootLogin no** directive to prevent SSH logins to the root account?

Lab 5

Success in this lab is confirmed by a good SSH connection from client to server. If you just want to make sure, use the **ssh -p 8022** command from the client. If you haven't disabled the AllowUsers directive on the server, that connection would have to be to the hawaii account.

In addition, this lab should give you a sense of the effort required to set up obscure ports. However, while the **nmap** command would detect an opening on port 8022, it would be obscure; the relevant output would be

```
PORT      STATE     SERVICE
8022/tcp  open      unknown
```

Go to the client system. Change the client configuration file (/etc/ssh/ssh_config) to point to port 8022. Try connecting to the SSH server. Remember, you'll also need to open port 8022 in the firewall of the SSH server.

When this lab is complete, restore the original port numbers on the SSH client and server.

Lab 6

Confirmation of success in this lab is straightforward. Run the **ls -Zd** commands on the noted directories. The SELinux contexts for the /virtual/web and /var/www directories should match with the following contexts:

```
system_u:object_r:httpd_sys_content_t:s0
```

The contexts for the /virtual/web/cgi-bin and /var/www/cgi-bin directories should also match:

```
system_u:object_r:httpd_sys_script_exec_t:s0
```

It should go without saying that any changes that you make should survive a SELinux relabel. Otherwise, how do you expect to get credit for your work? If you've run the **semanage fcontext -a** command on the correct directories, you'll see these contexts listed in the file_contexts.local file, in the /etc/selinux/targeted/contexts/files directory:

```
/virtual/web    system_u:object_r:httpd_sys_content_t:s0
/virtual/web/cgi-bin   system_u:object_r:httpd_sys_script_exec_t:s0
```