10

A Security Primer

CERTIFICATION OBJECTIVES

As you start the first chapter of the RHCE section of this book, you'll start with security. Many administrators and enterprises move toward Linux because they believe it's more secure. Since most Linux software is released under open-source licenses, the source code is available to all. Some believe that provides advantages for crackers who want to break into a system.

However, Linux developers are believers in collaboration. "Linus' Law," according to the open-source luminary Eric Raymond is that "given enough eyeballs, all bugs are shallow." Some of those eyes are from the U.S. National Security Agency (NSA), which has contributed a lot of code to Linux, including the foundations of SELinux.

The NSA has also contributed a number of other concepts adapted by Red Hat, which have been integrated into a layered security strategy. It includes system firewalls, wrappers on packets, and security by service. It includes both user- and host-based security. It includes access controls such as ownership, permissions, and SELinux. (A number of these layers were covered in earlier chapters.) The fundamentals of these layers of security, as they apply to RHCE objectives, are also covered here.

RHEL comes with a large and varied assortment of tools for handling security. These include tools for managing the security on individual Linux computers and tools for managing security for an entire network of systems, both Linux and otherwise. In this chapter, you'll examine some of the tools provided by RHEL for managing security. You'll start with some fundamentals, and continue with detailed analysis of firewalls, Pluggable Authentication Modules (PAM), TCP Wrappers, and more.

This is not the only chapter to focus on security. Strictly speaking, it covers only two of the RHCE objectives. However, this chapter covers the themes associated with security on Linux systems. And those themes can help you understand the security options associated with every service in this book.

CERTIFICATION OBJECTIVE 10.01

The Layers of Linux Security

The best computer security comes in layers. If there's a breach in one layer, such as penetration through a firewall, a compromised user account, or a buffer overflow that messes up a service, there's almost always some other security measure that prevents or at least minimizes further damage.

INSIDE THE EXAM

This chapter is the first one in this book focused on RHCE requirements. As described in the RHCE objectives, security starts with firewalls developed with the **iptables** command. The related objective is

- Use iptables to implement packet filtering and configure Network Address Translation (NAT)

But as suggested in the introduction, security is an issue for all services covered in the RHCE objectives. This chapter provides a foundation for a discussion of security, including several methods to

- Configure host-based and user-based security for the service

While host-based security can start with **iptables**-based firewalls, host and user-based security measures can involve the Extended Internet Super-Server, TCP Wrappers, and Pluggable Authentication Modules.

These options start with bastion hosts, which minimizes the functionality associated with an individual Linux system. The best defenses come in concert with other Linux developers, when you keep up to speed with the latest security updates. Beyond the firewall and SELinux come security options associated with individual services. Isolation options such as chroot jails are generally configured as part of a service. A number of these options are based on recommendations from the NSA.

While the sections on bastion systems are intended to be a lead-in to the security measures used for RHCE-level services, they also incorporate those security options often associated with the RHCSA exam, which are described in earlier chapters.

Bastion Systems

Properly configured, a bastion system minimizes the risk of a security breach. It's based on a minimal installation, with less software than was installed on the systems configured in Chapters 1 and 2. A bastion system is configured with two services. One service defines the functionality of the system. It could be a web server, a file server, an authentication server, or something similar. The other service supports remote access, such as SSH, or perhaps VNC over SSH.

Before virtualization, the use of bastion system was frequently limited. Only the wealthiest enterprises could afford to dedicate different physical systems to each service. If redundancy was required, the costs only increased further.

With virtualization, bastion systems are within reach of even smaller businesses. All that's needed is a standard minimal installation. With a few Kickstart files, you as an administrator of such a network could easily create a whole group of bastion systems. Each system could then be customized with and dedicated to a single server.

Well-constructed bastion systems follow two principles:

- If you don't need the software, uninstall it.
- If you need the software but aren't using it, make sure it's not active.

In general, crackers can't take advantage of a security flaw if the associated service isn't installed. If you do have to install the service for test purposes, keep that service inactive. That can help keep risks to a minimum. Of course, firewalls configured for each bastion system should allow traffic through only for the dedicated service and the remote access method.

Best Defenses with Security Updates

The best defenses come from security updates. You can review available updates with the Software Update tool. You can start that tool in a GUI with the **gpk-update-viewer** command. As discussed in Chapter 7, you can set up automatic security updates with the Software Updates Preferences tool that you can start in a GUI with the **gpk-prefs** command.

In practice, security is often a race. When a vulnerability is discovered, responsible developers in the open-source community post a public notice of the problem. They get to work on updates. Until that update is available and installed, any affected services might be vulnerable.

As a Linux professional, it's your job to know these vulnerabilities. If you maintain servers like Apache, vsFTP, and Samba, monitor the information feeds from these developers. Security news may come in various forms, from message board updates to RSS feeds. Normally, Red Hat also keeps up to speed on such issues. However, if you've subscribed to the forums maintained by the developers of a service, it's best to hear about problems and planned solutions directly from the source. To some extent, that is a province of service-specific security.

Service-Specific Security

Most major services have some level of security that can be configured within. In many cases, you can configure a service to limit access by host, by network, by user, and by group. As listed in the RHCE objectives, you need to know how to configure host- and user-based security for each listed service, as listed by protocol. SELinux options are also available that can help secure each of these services. While details are discussed in appropriate upcoming chapters, the following is a brief overview of service-specific security options.

HTTP/HTTPS Service-Specific Security

While there are alternatives, the primary service for the HTTP and HTTPS protocol on Linux is the Apache web server. In fact, Apache is the dominant web server on the Internet. No question, Apache configuration files are complex. But they need to be, as the security challenges on the Internet are substantial. Some options for responding to these challenges are covered in Chapter 14.

Apache includes a good number of optional software components. Don't install more than is absolutely necessary. If there's a security breach in a Common Gateway Interface (CGI) script and you haven't installed Apache support for CGI scripts, that security issue doesn't affect you. But as the RHCE specifies an objective to deploy a "basic CGI application," you don't have that luxury.

Fortunately, with Apache, you can limit access in a number of ways. Limits can be created on the server, or on individual virtual hosts. Different limits can be created on regular and secure web sites. In addition, Apache supports the use of secure certificates.

DNS Service-Specific Security

Domain Name Service (DNS) servers are a big target for crackers. With that in mind, RHEL 6 includes the bind-chroot package, which configures the necessary files, devices, and libraries in an isolated subdirectory. That subdirectory provides a limit for any user who breaks through DNS security known as a chroot jail. It's designed to limit the directories where a cracker can navigate if he does break into the service. In other words, crackers who do break into a RHEL 6 DNS server should not be able to "escape" the subdirectory configured as a chroot jail.

Since RHCE exam candidates are not expected to create a master or a slave DNS server, the challenges and risks are somewhat limited. Nevertheless, in Chapter 17, you'll see how to limit access to the configured DNS server by host.

FTP Service-Specific Security

While there are alternatives, the primary FTP server for RHEL 6 is vsFTP. While you created a basic FTP server in Chapter 1 with that service, no additional configuration was required. In Chapter 16, you'll configure the main vsFTP configuration files to limit access by user and by chroot jail. While you can use options like TCP Wrappers and **iptables** to limit access by IP address, such limits are not directly available in the vsFTP configuration files.

NFS Service-Specific Security

With the move to the Network File System, version 4, it is now possible to set up Kerberos authentication to support user-based security. However, the configuration of servers for Kerberos and LDAP is beyond the scope of the RHCE objectives. Thus, the discussion in Chapter 16 is focused on host-based security options.

SMB Service-Specific Security

The SMB listed in the RHCE objectives stands for the Server Message Block protocol. It's the networking protocol originally developed by IBM, later modified by Microsoft as the network protocol for its operating systems. While Microsoft now refers to it as the Common Internet File System (CIFS), the Linux implementation of this networking protocol is still known as Samba.

As implemented for RHEL 6, you can take advantage of Microsoft authentication databases. Samba supports the mapping of such users and groups into a Linux authentication database. Samba also supports both user- and host-based security on a global and a shared directory level, as discussed in Chapter 15.

The standard Samba for RHEL 6 is version 3.5.4. While there's a Samba version 4 package available from the RHEL 6 repositories, it is not yet ready for production use. When it is ready, you'll be able to set up RHEL 6 as an Active Directory Domain Controller.

SMTP Service-Specific Security

RHEL supports two different services for e-mail communication through the Simple Mail Transport Protocol (SMTP): Postfix and sendmail. Both are released under open-source licenses. The sendmail service is listed in lowercase, to distinguish it from the commercial release of Sendmail.

The default SMTP e-mail service for RHEL 6 is Postfix, which is a change—the default for RHEL 5 is sendmail. You can configure either service to meet the

associated RHCE objective. In either case, the service normally only listens on the localhost address, which is one level of security. Other levels of security are possible based on hosts, usernames, and more. For more information, see Chapter 13.

SSH Service-Specific Security

The SSH service is installed by default even in the minimal installation of RHEL 6. That encourages its use as a remote administration tool. However, there are risks associated with the SSH server that can be minimized. For example, it's not necessary to send even encrypted passwords over a network. Remote logins to the root account do not have to be allowed. Security can be further regulated by user.

Host-Based Security

Host-based security refers to access limits, not only by the system hostnames, but also by their fully qualified domain names and IP addresses. The syntax associated with host-based security can vary. For example, while every system recognizes a specific IP address such as 192.168.122.50, the use of wildcards or Classless Inter-Domain Routing (CIDR) notation for a range of IP addresses varies by service. Depending on the service, you may use one or more of the following options for the noted range of network addresses:

```
192.168.122.0/255.255.255.0
192.168.122.0/24
192.168.122.*
192.168.122.
192.168.122
```

Just be careful, some of these options may lead to syntax errors on some but not all network services. In a similar fashion, any of the following options may or may not work to represent all of the systems on an example.com network:

```
*.example.com
.example.com
example.com
```

User-Based Security

User-based security includes users and groups. Generally, users and groups who are allowed or denied access to a service are collected in a list. That list could include a

user on each line, as in files like /etc/cron.allow, or it could be in a list that follows a directive, such as

```
valid users = michael donna @book
```

Sometimes the syntax of a user list is unforgiving; in some cases, an extra space after a comma or at the end of a line may result in an authentication failure.

Groups are frequently included in a list of users, with a special symbol in front, such as an **@** or an **+**.

Sometimes, users who are allowed access to a system are configured in a separate authentication database, such as that associated with the Samba server, configured with the **smbpasswd** command.

Console Security

As discussed in Chapter 8, console security is regulated in the /etc/securetty and /etc/security/access.conf files. It can help you regulate local console access to root and regular users.

But console access is not just local. For a full view of console security, you need to be able to configure limits on remote console access. Two primary options are SSH, as discussed earlier, and Telnet. While the **telnet** command has its uses, as described in Chapter 2, communications to Telnet servers are inherently insecure. Usernames, passwords, and other communication to and from a Telnet server are transmitted in clear text. That means a network protocol analyzer such as Ethereal could be used to read those usernames, passwords, and any other critical information.

Even though Kerberos-based options are available for Telnet servers, most security professionals avoid Telnet for remote consoles at almost all costs. And that's consistent with the recommendations from the NSA.

Recommendations from the U.S. National Security Agency

The NSA has taken a special interest in Linux, and specifically Red Hat Enterprise Linux. Not only has the NSA taken the time to develop SELinux, but also it has created guides to help administrators like yourself create a more secure RHEL configuration. (Yes, the "super-secret" NSA has released SELinux code under open-source licenses for all to see.) They recognize the importance of Linux in the infrastructure of computer networks. Observers of RHEL may notice how changes between RHEL 5 and RHEL 6 follow NSA recommendations.

The NSA includes five general principles for securing operating systems in general and RHEL in particular.

- **Encrypt Transmitted Data Whenever Possible** NSA recommendations for encryption include communications over what should be private and secure networks. SSH, with the security options described in Chapter 11, are an excellent step in this process.
- **Minimize Software to Minimize Vulnerability** As suggested by the NSA, "The simplest way to avoid vulnerabilities in software is to avoid installing that software." The NSA pays special attention to any software that can communicate over a network, including the Linux GUI. The minimal installation of RHEL 6 includes far fewer packages than the comparable installation of RHEL 5.
- **Run Different Network Services on Separate Systems** This is consistent with the concept of bastion servers described earlier in this chapter. Implementation is made easier by the flexibility afforded by virtual machine technologies such as KVM.
- **Configure Security Tools to Improve System Robustness** The RHCSA and RHCE objectives have this well covered, with the use of **iptables**-based firewalls, SELinux, and appropriate log collection services.
- **Least Privilege** In principle, you should give users the minimum privileges required to accomplish their tasks. Not only does that mean minimize access to the root administrative account, but also careful use of the **sudo** command privileges. SELinux options such as the user_u role for confinement described in Chapter 4 may also be helpful to that end.

The PolicyKit

The PolicyKit is one more security mechanism designed to help protect different administrative tools. When starting an administrative tool in the GUI from a regular account, most tools prompt for the root administrative password with a window similar to Figure 10-1.

Alternatively, you might see a slightly different window similar to that shown in Figure 10-2. Functionally, the effect is the same. As described in the window, authentication by the superuser is required. In this case, you'd still have to enter the root administrative password. However, there's a difference, as shown under details. The action specifies the policy required by the "vendor," in this case, the system-config-firewall command.

FIGURE 10-1

Access to Administrative Tools in the GUI requires the root password.

The action noted is "org.fedoraproject.config.firewall.auth," which is associated with a policy file in the /usr/share/polkit-1/actions directory. Policy configuration files are stored in this directory; the corresponding file for the **system-config-firewall** tool is org.fedoraproject.config.firewall.policy.

These policy files are configured in XML format and may be modified further to support fine-grained control by individual users. However, since the PolicyKit only works within the GUI, its fine-grained controls do not affect the use of administrative tools from the console. It certainly does not prevent administrators from configuring or otherwise controlling important services. If fine-grained control is required, the better tool is the /etc/sudoers file described in Chapter 8.

FIGURE 10-2

Access to Administrative Tools may be limited by the PolicyKit.

CERTIFICATION OBJECTIVE 10.02

Firewalls and Network Address Translation

Typically, firewalls reside between internal LANs and outside insecure networks such as the Internet. A firewall can be configured to examine every network packet that passes into or out of your LAN. When configured with appropriate rules, it can filter out those packets that may pose a security risk to the systems on the LAN.

However, to follow the spirit of the recommendations from the NSA, you'll configure a firewall on every system. While Network Address Translation uses the same **iptables** command, its use is generally still most appropriate for those systems on the gateway or router between a LAN and an outside network.

Definitions

Firewalls based on the **iptables** command work by reading the headers of each packet of network data. Based on the information contained in the headers, **iptables**-based rules can be used to filter each packet. To understand how *packet filtering* works, you have to understand a little bit about how information is sent across networks.

Before a message is sent over a network, that message is broken down into smaller units called *packets*. Administrative information, including the type of data, the source address, and the destination address, is added to the header of each packet. The packets are reassembled when they reach the destination computer. A firewall examines the fields in each header. Based on existing rules, the firewall may then take one of four actions with that packet:

- Allow the packet into the system.
- Forward the packet to other systems if the current system is a gateway or router between networks.
- Reject the packet with a message sent to the originating IP address.
- Drop the packet without sending any sort of message.

Whatever the result, the decision can be logged. If there are a substantial number of packets that are rejected or dropped, a log file may be useful.

RHEL 6 comes with everything you need to configure a system to be a firewall, including the **iptables** and **ip6tables** commands for IPv4 and IPv6 networks.

In contrast, NAT hides the IP address of the computers of a LAN that connect to outside networks. NAT replaces the internal source address with the IP address of the gateway or router system with the firewall. That internal source address is cached on the gateway, so it knows which computer made the request.

When the firewall receives data such as a web page, the process is reversed. As the packets pass through the firewall, the originating computer is identified in the cache. The header of each packet is modified accordingly before the packets are sent on their way.

This approach is useful for several reasons. Hiding that internal IP addresses makes it harder for a cracker to know what IP address to use to break into an internal network. NAT supports connections between systems with private IP addresses and external networks such as the Internet. It's the reason why IPv4 addressing has survived for so long. In the Linux world, this process is known as *IP masquerading*.

The Structure of the iptables Command

The way **iptables** commands are assembled into a firewall is based on "chains." A chain of firewall rules may be applied to each network packet, in order. Each rule in a chain does two things: it specifies the conditions a packet must meet to match the rule, and it specifies the action if the packet matches.

The **iptables** command uses the following basic format:

```
iptables -t tabletype <action direction> <packet pattern> -j
<what to do>
```

Now analyze this command, item by item. The first item is the ***-t tabletype*** switch. There are three basic *tabletype* options of interest for **iptables**:

- **filter** Sets a rule for filtering packets.
- **nat** Configures Network Address Translation, also known as masquerading, discussed later in this chapter.
- **mangle** Changes packet headers.

The default is **filter**; if you don't specify a ***-t tabletype,*** the **iptables** command assumes that you're trying to create a filtering rule.

The next item is the <***action direction***>. There are four basic actions associated with **iptables** rules:

- **-A (--append)** Appends a rule to the end of a chain.

- **-D (--delete)** Deletes a rule from a chain. Specify the rule by the number or the packet pattern.
- **-L (--list)** Lists the currently configured rules in the chain.
- **-F (--flush)** Flushes all of the rules in the current **iptables** chain.

If you're appending to (**-A**) or deleting from (**-D**) a chain, you'll want to apply it to network data traveling in one of three directions. In most cases for a regular firewall that protects a system from external data, the appropriate direction to apply is **INPUT**:

- **INPUT** All incoming packets are checked against the rules in this chain.
- **OUTPUT** All outgoing packets are checked against the rules in this chain.
- **FORWARD** All packets being sent to another computer are checked against the rules in this chain.

Next, you need to configure a <***packet pattern***>. All **iptables** firewalls check every packet against this pattern. The simplest pattern is by IP address:

- **-s *ip_address*** All packets are checked for a specific source IP address.
- **-d *ip_address*** All packets are checked for a specific destination IP address.

Packet patterns can be more complex. In TCP/IP, most packets are transported using the Transport Control Protocol (TCP), the User Datagram Protocol (UDP), or the Internet Control Message Protocol (ICMP) protocols. You can specify the protocol with the **-p** switch, followed by the destination port (**--dport**). For example, the **-p tcp --dport 80** extension affects users outside your network who are trying to use an HTTP connection.

Once the **iptables** command finds a packet pattern match, it needs to know what to do with that packet, which leads to the last part of the command, **-j <*what to do*>**. There are three basic options:

- **DROP** The packet is dropped. No message is sent to the requesting computer.
- **REJECT** The packet is dropped. An error message is sent to the requesting computer.
- **ACCEPT** The packet is allowed to proceed as specified with the **-A** action: **INPUT**, **OUTPUT**, or **FORWARD**.

Take a look at some examples of how you can use **iptables** commands to configure a firewall. A good first step is to see what is currently configured, with the following command:

```
# iptables -L
```

If **iptables** is properly configured, it should return chain rules in three different categories: **INPUT**, **FORWARD**, and **OUTPUT**.

The following rule rejects all traffic from the 192.168.75.0 subnet, and it sends a "destination unreachable" error message back to any client from that network address who tried to connect:

```
# iptables -A INPUT -s 192.168.75.0/24 -j REJECT
```

This rule stops users from the computer with an IP address of 192.168.25.200 from "pinging" your system, as the **ping** command uses the ICMP protocol:

```
# iptables -A INPUT -s 192.168.25.200 -p icmp -j DROP
```

The following command guards against TCP SYN attacks from outside the local system, associated with packet floods and denial of service attacks. Assume that the IP address of the LAN to be protected is 192.168.1.0. The exclamation point (**!**) inverts the meaning; in this case, the command applies to all IP addresses except those with a 192.168.1.0 network address (and a 255.255.255.0 subnet mask).

SYN is not an acronym, but a type of packet that is sent from a client using TCP. The response is an SYN-ACK packet, which is an acknowledgment. The client then sends an ACK message to the server. The associated* iptables *rule stops potential attacks at the SYN level.

```
# iptables -A INPUT -s !192.168.1.0/24 -p tcp -j DROP
```

Then, if you want to delete the rule related to the **ping** command in this list, use the following command:

```
# iptables -D INPUT -s 192.168.25.200 -p icmp -j DROP
```

The default rule for **INPUT**, **OUTPUT**, and **FORWARD** is to **ACCEPT** all packets. One way to stop packet forwarding is to add the following rule:

```
# iptables -A FORWARD -j DROP
```

The Default Firewall

Now that you've seen the effect of various firewall rules, it's an appropriate time to look at the default firewall for RHEL 6, based on the VM systems installed in Chapters 1 and 2. While you can use the **iptables -L** command for that purpose, the firewall rules implemented during the boot process are stored in the /etc/sysconfig/iptables file. (For IPv6 networks, the corresponding firewall configuration file is /etc/sysconfig/ip6tables.) The meaning of each of these lines is described in Chapter 4.

Recommendations from the NSA

Simple firewalls are frequently the most secure. On an exam, it's best to keep everything, firewalls included, as simple as possible. But the NSA would go further. It has recommendations with respect to the default rules, limitations on the **ping** command, and blocks from suspicious groups of IP addresses. To those recommendations, I add a couple to reduce risks to a SSH system that may be applicable to other services. While these recommendations go beyond what's suggested by the RHCE objectives, read this section. If you're less than comfortable with the **iptables** command, this section can help.

To implement these changes on multiple systems, it may be most efficient to edit the /etc/sysconfig/iptables file directly. While you could implement these changes with the Custom Rules option in the Firewall Configuration tool, that is less efficient than a script that directly edits the noted file. Unfortunately, the Firewall Configuration tool overwrites any such changes. So if you administer the firewall on multiple systems with scripts, make sure to back up that /etc/sysconfig/iptables file. Nevertheless, the Custom Rules option in the Firewall Configuration tool is an excellent choice if you're just changing the firewall for one or two systems.

exam Watch

The suggested changes to iptables-based firewalls are just recommendations. However, since the requirement to "implement packet filtering" is generic, it's useful to examine a variety of examples.

Then you can test any of these suggestions on a system like the server1.example.com VM created in Chapter 2. To do so, edit the /etc/sysconfig/iptables file. And then to put such changes into effect, run the following command:

```
# /etc/init.d/iptables restart
```

Recommended Changes to Default Rules

It recommends changes to default rules that are appropriate on bastion servers such as the VMs configured early in this book. The default is

```
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
```

The NSA recommendation would change this to

```
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
```

While the rules that allow network traffic into the system still apply, these changes provide another limit on traffic into a system, and traffic that may be forwarded to another system.

Just be careful. When I tried these changes on a gateway system, specifically the physical host system for my VMs, it stopped communication by dropping packets between VMs and external networks. I should have limited such changes to the VMs. So test any changes before implementing them on production systems.

Regulate the ping Command

One earlier attack on various Internet systems involved the **ping** command. From Linux, it's possible to flood another system with the **-f** switch. It may transmit thousands of packets per second. It's important to be able to defend a system from such attacks, as they can prevent others from accessing your web sites and more.

on the job

The -f switch to the* ping *command was described solely to point out one of the major risks on a network. In many cases, it is illegal to run such a command on or against someone else's system. For example, one Wikipedia article suggests that such an attack could be a violation of the Police and Justice Act in the United Kingdom with a penalty of up to ten years in prison. Similar laws exist in other countries.

One potentially troublesome rule in the default firewall is

```
-A INPUT -p icmp -j ACCEPT
```

However, ICMP messages go both ways. If you run the **ping** command on a remote system, the remote system responds with an ICMP packet. So if you want to limit ICMP messages, the following rules allow "acceptable" responses to a **ping**:

```
-A INPUT -p icmp --icmp-type echo-reply -j ACCEPT
-A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT
-A INPUT -p icmp --icmp-type time-exceeded -j ACCEPT
```

The following rule limits the rate at which a **ping** command can be applied to a system:

```
-A INPUT -p icmp --icmp-type echo-request -m limit --limit 1/s -j ACCEPT
```

While the Firewall Configuration tool discussed later in this chapter can limit the effect of ICMP messages in most of the same ways, it does not have the ability to regulate the rate of **ping** network packets accepted onto a system.

Block Suspicious IP Addresses

Crackers who want to break into a system may hide their source IP address. As nobody is supposed to use a private, multicast, or experimental IPv4 address on the public Internet, such addresses are one way to hide. The following additions to the /etc/sysconfig/iptables file would drop packets sourced from the specified IPv4 network address blocks:

```
-A INPUT -i eth0 -s 10.0.0.0/8 -j DROP
-A INPUT -i eth0 -s 172.16.0.0/12 -j DROP
-A INPUT -i eth0 -s 192.168.0.0/16 -j DROP
-A INPUT -i eth0 -s 224.0.0.0/4 -j DROP
-A INPUT -i eth0 -s 240.0.0.0/5 -j DROP
```

Regulate Access to SSH

Since SSH is such an important means for the administration of remote systems, additional measures to protect such services are important. It's certainly possible to set up a nonstandard port for SSH communication. Such a measure can be a part of a layered security strategy. However, tools like **nmap** can detect the use of SSH on such nonstandard ports. So it's generally better to set up the configuration of the SSH server as discussed in Chapter 11 along with firewall rules such as the following. The first rule shown here creates a new chain, SSH_CHAIN:

```
-N SSH_CHAIN
```

The rule that follows looks at all TCP traffic to port 22:

```
-A INPUT -i eth0 -p tcp -m tcp --dport 22 -m state --state NEW -j SSH_CHAIN
```

This line marks the packet with the source address:

```
-A SSH_CHAIN -m recent --set --name SSH
```

The final rule starts the SSH_CHAIN, by limiting access requests to three per minute.

```
-A SSH_CHAIN -i eth0 -p tcp -m tcp --dport 22 -m state --state NEW -m recent
--update --seconds 60 --hitcount 3 --rttl --name SSH -j DROP
```

Make Sure the Firewall Is Running

Once desired changes are saved to the /etc/sysconfig/iptables configuration file, it's important to make sure the Firewall is in operation, with the new rules. Since the reload option is not available in the iptables service script, you'll need to do so with the following command:

```
# service iptables restart
```

As discussed in Chapter 11, this is functionally equivalent to the **/etc/init.d/iptables restart** command.

exam Watch

It's critical to understand how to secure a Red Hat Enterprise Linux system against unauthorized access.

IP Masquerading

Red Hat Enterprise Linux supports a variation of NAT called *IP masquerading*. IP masquerading supports Internet access from multiple computers with a single public IP address. IP masquerading maps multiple internal IP addresses to that single valid external IP address. That helps as all public IPv4 address blocks have now been allocated. IPv4 addresses are often still available from these third parties, at a price. That cost is one more reason for IP Masquerading. On the other hand, systems on IPv6 networks may not need masquerading, as it's relatively easy for many requesting users to get their own subnet of public IPv6 addresses. Nevertheless, even on IPv6 networks, masquerading can help keep that system secure.

exam Watch

The RHCE objectives specify the use of* iptables *to configure network address translation.

IP Masquerading is a fairly straightforward process. It's implemented on a gateway or router, where the system has two or more network cards. One network card is connected to an outside network such as the Internet, and the second (and additional) network card is connected to a LAN. The card connected to the outside network may connect through an external device such as a cable "modem" or Digital Subscriber Line (DSL) adapter. The following assumptions are made for the configuration:

- The public IP address is assigned to the network card that is directly connected to the outside network.

- Network cards on the LAN get IP addresses associated with a single private network.
- One network card on the gateway or router system gets an IP address on that same private network.
- The same **iptables** command and configuration files used to set up a firewall are also used to set up IP masquerading.
- IP forwarding is enabled on the router or gateway system, as discussed later in this chapter.
- Each system on the LAN is configured with the private IP address of the router or gateway system as the default gateway address.

When a computer on a LAN wants a web page on the Internet, it sends packets to the firewall. The firewall replaces the source IP address on each packet with the firewall's public IP address. It then assigns a new port number to the packet. The firewall caches the original source IP address and port number.

When a packet comes in from the Internet to the firewall, it should include a port number. If the firewall can match an associated rule with the port number assigned to a specific outgoing packet, the process is reversed. The firewall replaces the destination IP address and port number with the internal computer's private IP address and then forwards the packet back to the original client on the LAN.

In practice, the following command uses **iptables** to enable masquerading. The noted command assumes that eth1 represents the network card that is directly connected to the Internet, with a private IP network of 192.168.0.0/24:

```
# iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o eth1 -j MASQUERADE
```

In most cases, the private IP network address is not required, as most LANs protected by a masquerade are configured on a single private IP network.

If you're using separate private networks, such as for the KVM-based virtual machines configured in Chapters 1 and 2, masquerading of those networks enables communication between those systems. In that configuration, masquerading rules would be applied to the firewall of the physical host system. On my system, I have the following masquerading rules:

```
# iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
# iptables -t nat -A POSTROUTING -o virbr0 -j MASQUERADE
# iptables -t nat -A POSTROUTING -o virbr1 -j MASQUERADE
```

These rules work hand in hand with the IP forwarding rules discussed next. Just be aware, masquerading substitutes the IP address of the router for that of the

originating system. So if you set up masquerading on the physical host system for two VMs, communication from one VM appears to be coming from the IP address of the physical host system. If that's not desirable, you may prefer to configure IP forwarding.

IP Forwarding

IP forwarding is more commonly referred to as *routing*. Routing is critical to the operation of the Internet or any IP network. Routers connect and facilitate communication between multiple networks. When you set up a computer to find a site on an outside network, it needs a gateway address. This corresponds to the IP address of a router on the LAN.

A router looks at the destination IP address of each packet. If the IP address is on one of its LANs, it routes the packet directly to the proper computer. Otherwise, it sends the packet to another gateway closer to its final destination. To use a Red Hat Enterprise Linux system as a router, you should enable IP forwarding in the /etc/sysctl.conf configuration file by changing

```
net.ipv4.ip_forward = 0
```

to

```
net.ipv4.ip_forward = 1
```

These settings take effect on the next reboot. Until then, IPv4 forwarding can be enabled with the following command:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

But that is not enough. You'll also need to enable routing through the firewall for communications to other networks. For example, with the KVM-based VM systems on two virtual networks, they include two virtual network devices in the output to the **ifconfig** command on the physical host: virbr0 and virbr1. On the system that I'm using to write this book, it also includes the local wireless network device, wlan0. For a physical host with a regular wired Ethernet card, you may need to substitute eth0 for wlan0.

```
# iptables -A FORWARD -o wlan0 -j ACCEPT
# iptables -A FORWARD -o virbr0 -j ACCEPT
# iptables -A FORWARD -o virbr1 -j ACCEPT
```

If you actually need to configure a connection to external networks such as the Internet, a change is also required to the /etc/resolv.conf file on the VMs. Normally,

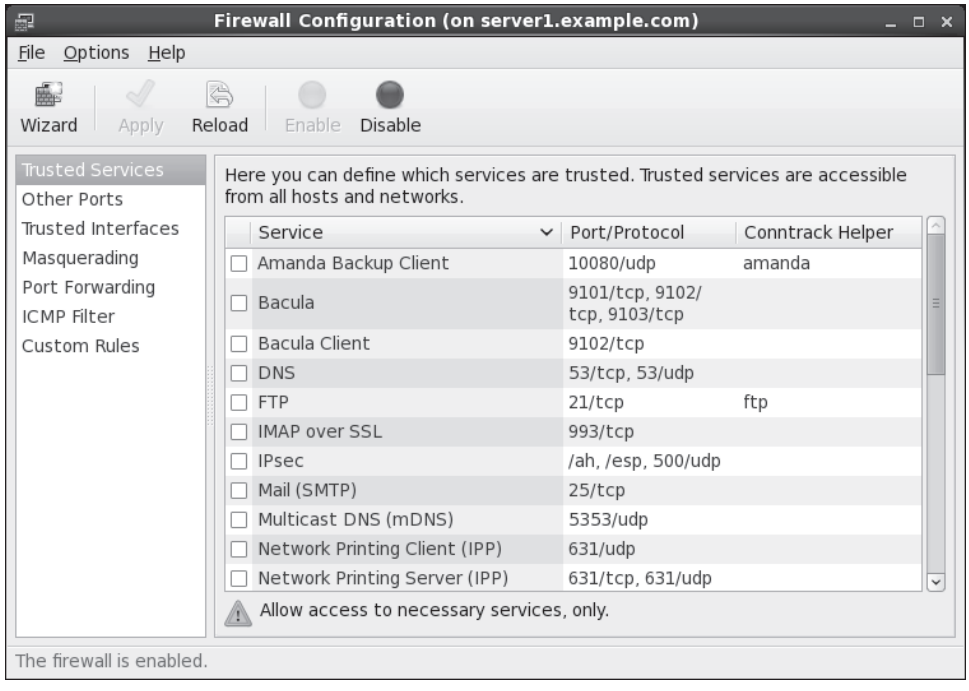
it sets the default DNS server to the IP address of the virbr0 and virbr1 networks on the physical host system. You may need to change that DNS server IP address to that used for other systems on the local network. However, that should not be necessary, as Internet access is not available during the Red Hat exams.

The Red Hat Firewall Configuration Tool

The basic functionality of the Red Hat Firewall Configuration tool was discussed in Chapter 4, with respect to RHCSA objectives. The Firewall Configuration tool can do more. Start it with the **system-config-firewall** command or by clicking System | Administration | Firewall. This is a tool with a number of capabilities, as shown in Figure 10-3. In general, you can immediately implement any changes made with the Apply button. But for that reason, you may want to back up the current version of the /etc/sysconfig/iptables file before doing anything with the Firewall Configuration tool.

FIGURE 10-3

The Firewall Configuration tool



Similar functionality is available from the console version of the tool. You can open it with the **system-config-firewall-tui** command. Select Customize for access to the same options shown in Figure 10-3. The following sections address those options not previously discussed in Chapter 4.

Trusted Interfaces

In the Firewall Configuration tool, click Trusted Interfaces to reveal the window shown in Figure 10-4. Routers and gateways have two or more network cards. Administrators who trust the systems on the internal network may choose to disable the firewall on that interface. However, it can be a risky practice. Threats can come from within as well as from outside a network.

The options shown in Figure 10-4 would apply to all interfaces of each type, as listed in Table 10-1. They all end with a +, which is effectively like a wildcard. For example, eth+ is associated with device eth0, eth1, and so on.

In most cases, the gateway or router system will have two Ethernet devices. Assume those devices are eth0 and eth1, where eth0 is connected to an external network. If the local network is to be trusted, then you might set up device eth1 as the trusted device.

Sometimes wireless devices appear as other names. In some configurations, wireless hardware appear as Ethernet devices such as eth0 or eth1. In other cases, I've seen wireless devices appear as wlan0 or even ath0. So it's important to know the device files associated with each network device on a system.

Click Add to open the Interface window shown in Figure 10-5. Then enter the device name of the interface, in this case, eth1.

TABLE 10-1 Network Interface Types

Device	Description
eth+	Ethernet devices
ippp+	Integrated Services Digital Network (ISDN) devices for Point to Point Protocol (PPP) communication
isdn+	Regular ISDN devices
ppp+	PPP devices, normally associated with telephone modems
tun+	Tunneling devices, often associated with virtual private networks
wlan+	Wireless LAN devices

FIGURE 10-4

Trusted Interfaces

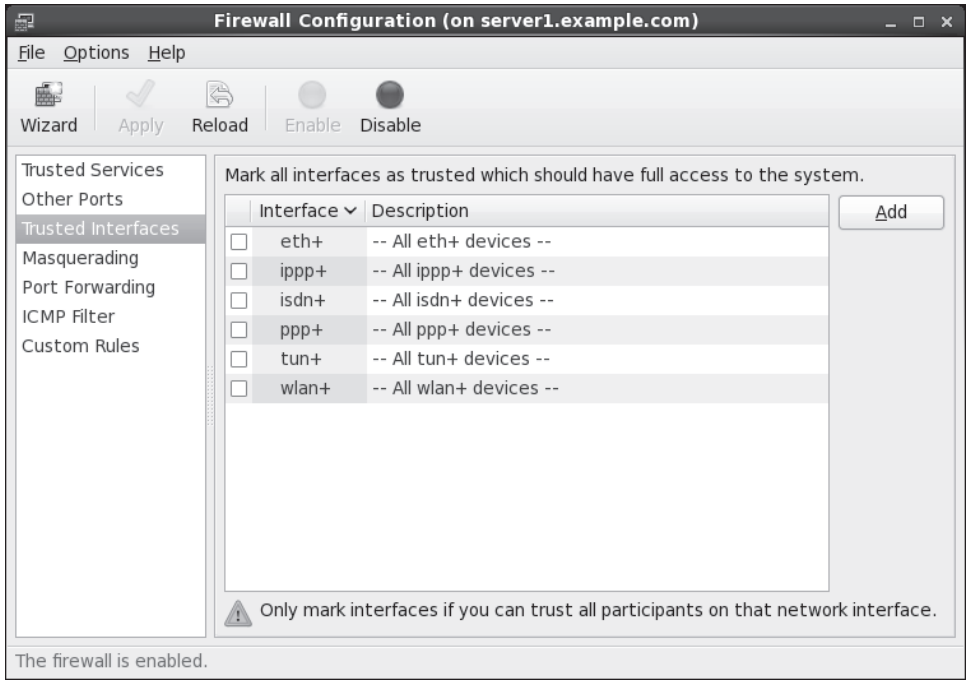
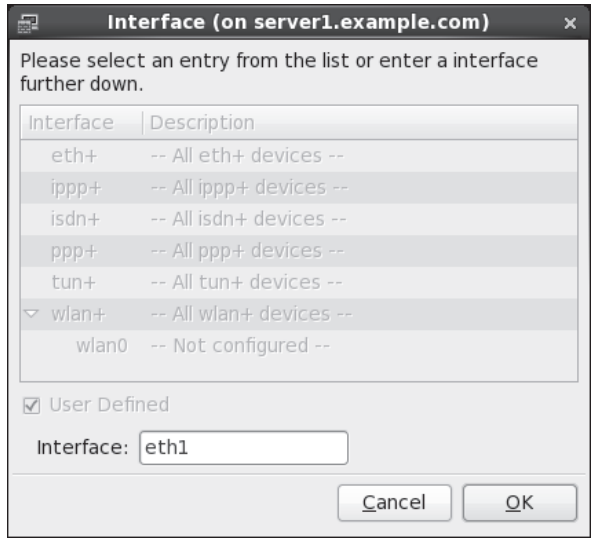


FIGURE 10-5

A user-defined trusted interface



Once applied, trusted interfaces add rules to the /etc/sysconfig/iptables file. For example, if you select the wlan+ devices, all devices with that name are trusted, as documented with the following directive in the noted file:

```
-A INPUT -i wlan+ -j ACCEPT
```

which accepts all network packets that come into all "wireless" devices. In contrast, the following directive is more specific, based on the user-defined trusted interface from Figure 10-5.

```
-A INPUT -i eth1 -j ACCEPT
```

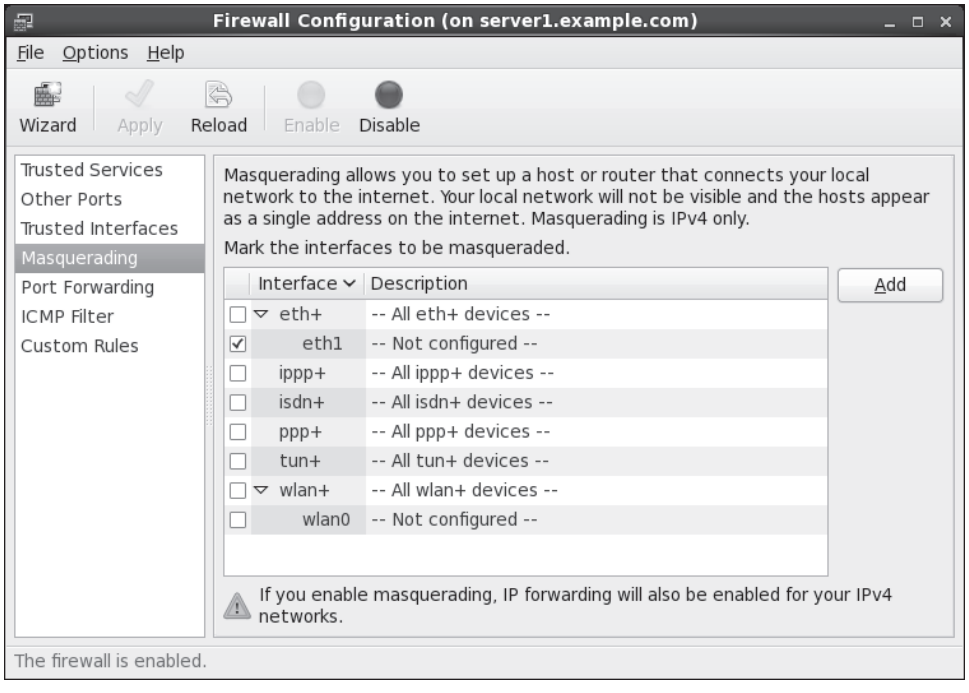
You should also note these rules appear just after a similar rule for the loopback device, lo. These rules appear before any other rule that allows packets in through certain ports, such as 22 for SSH communication.

Masquerading

In the Firewall Configuration tool, click Masquerading to reveal the window shown in Figure 10-6. Routers and gateways have two or more network devices. In most

FIGURE 10-6

Masquerading with the Firewall Configuration tool



cases, you should set up masquerading for the systems on an internal network. That has three advantages:

- It hides the IP address identity of the internal systems from external networks.
- It requires only one public IP address.
- It sets up forwarding across the configured network devices.

As with trusted interfaces, you should know the device filename associated with each network card. If you've selected a specific device as a trusted interface, such as eth1, that device should also appear in this section. Otherwise, it's selectable in a similar fashion.

Administrators can choose to set up masquerading on the network interface of their choice. The selected network interface should be the one connected to an external network such as the Internet. The action adds several commands to the /etc/sysconfig/iptables file. The following example is based on the eth1 device as the network interface connected to the external network.

The first line specifies the table type, associated with the **iptables -t** switch. The **nat** option stands for network address translation, the functionality associated with masquerading.

```
nat
```

The four lines that follow accept data for forwarding, before routing (PREROUTING), as output to another network (OUTPUT), and after routing has been determined (POSTROUTING) back out through the eth1 interface with outside networks. The COMMIT directive actually commits the commands to the firewall.

```
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -o eth1 -j MASQUERADE
COMMIT
```

Several directives are also added toward the end of the firewall, applied to packets that are to be forwarded. The first FORWARD directive shown here continues communication that is already in process:

```
-A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

The next directive accepts **ping** and other ICMP packets, which you may want to change.

```
-A FORWARD -p icmp -j ACCEPT
```

The next two options accept packets forwarded through the loopback adapter (lo), and back out (-o) through the interface to the external network (eth1).

```
-A FORWARD -i lo -j ACCEPT
-A FORWARD -o eth1 -j ACCEPT
```

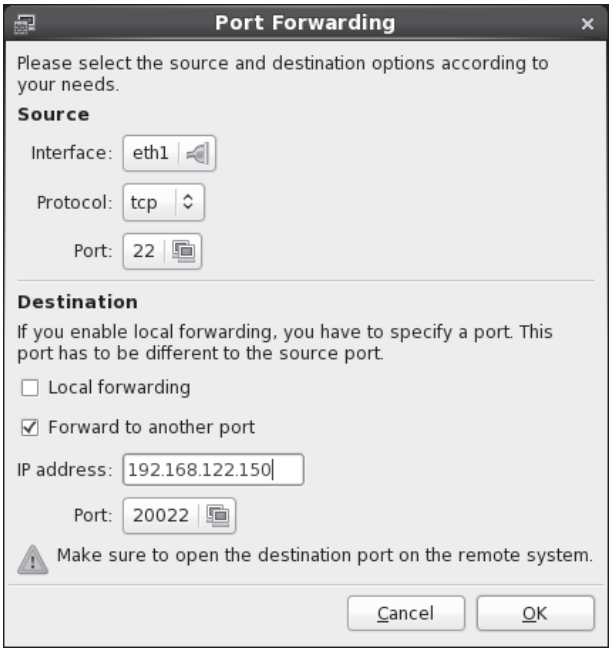
Port Forwarding

In the Firewall Configuration tool, select Port Forwarding. As suggested in the description, forwarding in this fashion works only in combination with masquerading. With such rules, port forwarding can be used to set up communication to one port on a specific network interface to a port on a remote system, as defined by its IP address. One example is shown in Figure 10-7.

The options shown in the figure would include two additional rules, which redirect traffic destined for port 22 on the eth1 network device to a remote destination, with

FIGURE 10-7

Port forwarding with the Firewall Configuration tool



an IP address of 192.168.122.150. The port on that remote system is 20022. The second rule makes sure that information forwarded to that port on the noted IP address is accepted and forwarded through the firewall.

```
-A PREROUTING -i eth1 -p tcp --dport 22 -j DNAT --to-destination
192.168.122.150:20022
-A FORWARD -i eth1 -m state --state NEW -m tcp -p tcp -d 192.168.122.150 --dport
20022 -j ACCEPT
```

ICMP Filter

In the Firewall Configuration tool, click ICMP filter to open the screen shown in Figure 10-8. As suggested in the description, the options listed related to different messages associated with the ICMP protocol. This is not limited to the **ping** command and responses. The options shown are further described in Table 10-2. If a filter in that table is activated, the packets of the noted filter are blocked.

As shown in Figure 10-8, if you hover the cursor over an option, the Firewall Configuration tool provides more information.

FIGURE 10-8

ICMP filters with the Firewall Configuration tool

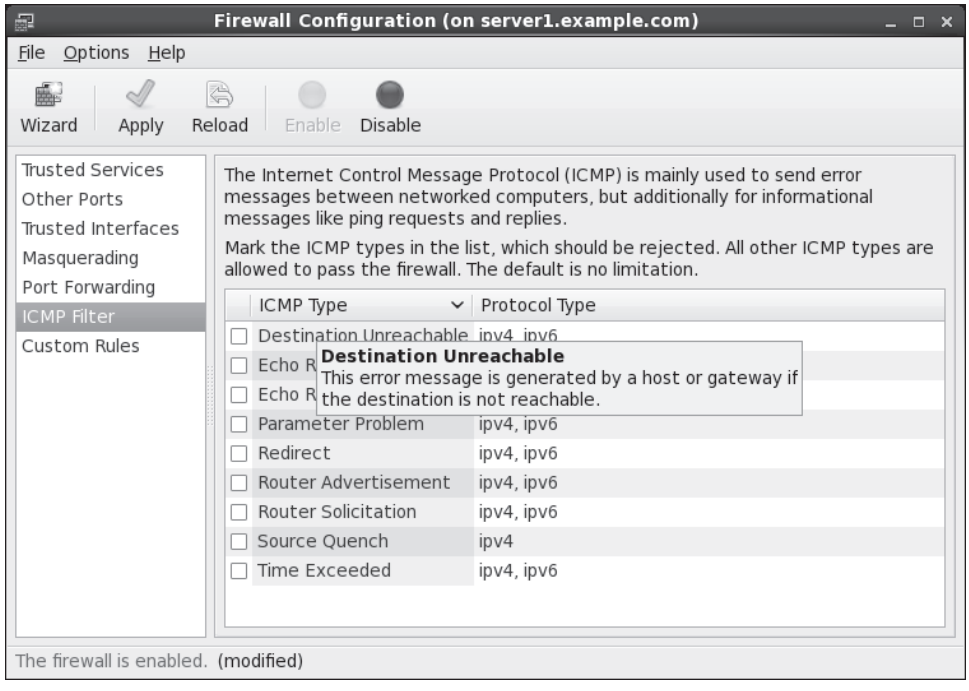


TABLE 10-3 ICMP Filter Options

Filter	Description
Destination Unreachable	A host not found message in response to a **ping** command
Echo Reply	Regular response messages to the **ping** command
Echo Request	A packet associated with the actual **ping** command
Parameter Problem	Error message not otherwise defined
Redirect	For a routing message
Router Advertisement	Periodic message to other routers of a multicast IP address
Router Solicitation	A request for a router advertisement
Source Quench	Response to a host to slow down packet transfers
Time Exceeded	Error message if a "Time To Live" message in a packet is exceeded

Custom Rules

The developers behind Red Hat and Fedora have done excellent work to improve the flexibility of the Firewall Configuration tool. A Linux guru who wants to create a custom firewall might uninstall both firewall tools by uninstalling the system-config-firewall and system-config-firewall-tui packages. As suggested earlier, that would help ensure that custom rules created by scripts aren't overwritten by the Firewall Configuration tool. However, someone who is administering just a few systems can use the Red Hat Firewall Configuration tool to set up a file with custom rules. For the purpose of this section, I created a file, /root/iptables-custom, with the following lines:

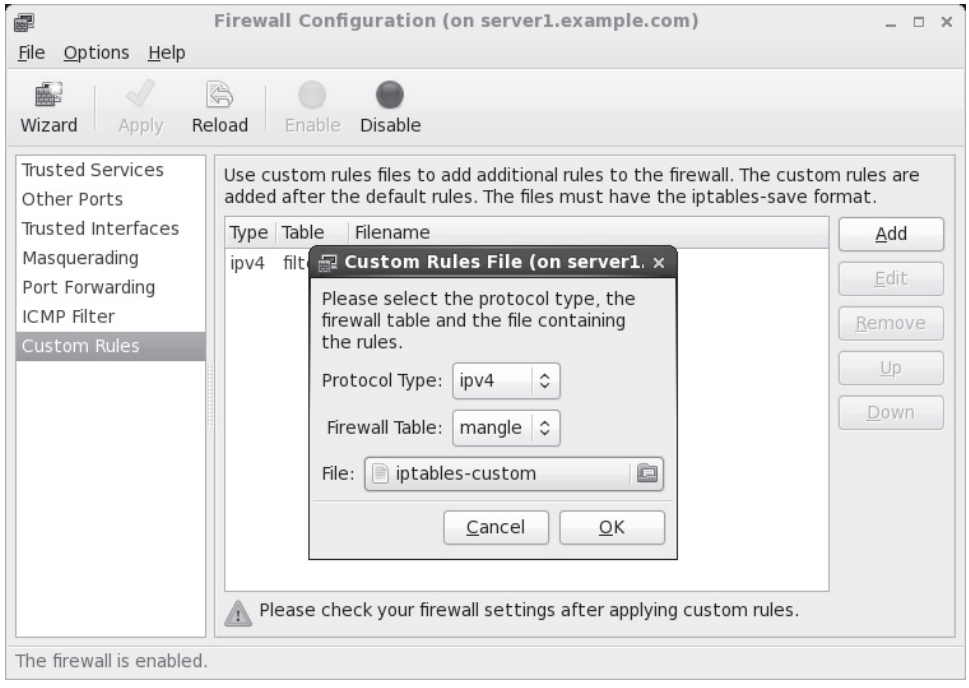
```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 2222 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 8080 -j ACCEPT
```

In the Firewall Configuration tool, I then selected Custom Rules and clicked Add to open the Custom Rules File window shown in Figure 10-9.

If you understand the **iptables** command, it's easier to set up a custom file. Such files are easier to transfer form system to system. Figure 10-9, in fact, shows the result after I clicked the File button and selected the /root/iptables-custom file that I created for this purpose. The name of the custom file that you may choose to create does not matter. When the change is applied, the rules are added to the /etc/sysconfig/iptables file in an appropriate location.

FIGURE 10-9

Take advantage of custom rules.



CERTIFICATION OBJECTIVE 10.03

The Extended Internet Super-Server

Linux typically supports network communication between clients and servers. For example, even though it's insecure, it is still possible to use Telnet on Linux to connect to a remote system. The Telnet client on a local computer makes a connection with a Telnet server daemon on the remote system. This section assumes that you've installed the default RHEL xinetd and telnet-server packages. The use of Telnet in this section is for illustration purposes only. This book does not endorse the use of Telnet or any clear-text protocol for private data.

While the focus in this section is on Telnet, other xinetd packages of note include rsync, which is popular for backups and cvs, which is popular for software development version control. As no xinetd service is explicitly cited in the RHCE objectives, I keep

the coverage of xinetd services to a minimum. Nevertheless, xinetd services are "Network Services," an important subset of the exam objectives.

The **xinetd** (also known as the Extended Internet Services Daemon) service can start a number of server daemons simultaneously. The **xinetd** service listens for connection requests for all *active* servers with scripts in the /etc/xinetd.d directory. There's a generic configuration file for xinetd services, /etc/xinetd.conf. The scripts in the /etc/xinetd.d directory function as service-specific configuration files.

Generic xinetd Configuration

The generic configuration for xinetd services is stored in the /etc/xinetd.conf file. As RHCE candidates need to configure services only for "basic operation," this chapter analyzes only the active directives in this file. First, a number of default settings are enabled with the following command:

```
defaults
```

This allows services such as rsync to retain their default TCP/IP ports (873) within the xinetd service.

This is followed by

```
log_type = SYSLOG daemon info
```

which specifies logging through the rsyslog daemon as described in Chapters 9 and 17, as configured in /etc/rsyslog.conf.

This is followed by

```
log_on_failure = HOST
```

which specifies the logging information when a login through an xinetd-controlled service fails. Naturally, this specifies the hostname (or IP address) of the client host. It might help to add USERID to the list, which lists the UID number associated with the failed login. This can help you identify compromised accounts.

This line,

```
log_on_success = PID HOST DURATION EXIT
```

specifies the logging information associated with a successful connection. For example, once I logged off a Telnet connection from a remote system, and this led to the following entries in /var/log/messages:

```
Jan 31 08:46:55 server1 xinetd[16543]: START: telnet pid=16582
from=::ffff:tester1.example.com
```

```
Jan 31 08:47:01 server1 xinetd[16543]: EXIT: telnet status=0 pid=16582
duration=6(sec)
```

The effect from /etc/xinetd.conf is straightforward. The next active line is

```
cps = 50 10
```

The **cps** command prevents attempts to "flood" any xinetd service; this line limits connections to 50 per second. If this limit is exceeded, xinetd waits 10 seconds before allowing a remote user to try again.

The next line,

```
instances = 50
```

limits the number of active services for a particular service; in this case, no more than 50 users can be logged into the local Telnet server simultaneously. That number goes down if other xinetd services are running.

This is followed by a related directive:

```
per_source = 10
```

which limits the number of connections from each IP address.

The next active directive is almost self-explanatory:

```
v6only = no
```

If this were set to **yes**, access would be limited to systems with IPv6 addresses.

A couple of environment directives follow, which allow execution with the xinetd group, or one defined with the **group** directive (singular), since there is no default xinetd group.

```
groups = yes
umask = 002
```

Finally, the last active line supports the use of the other configuration files specified in the /etc/xinetd.d directory:

```
includedir /etc/xinetd.d
```

Service-Specific xinetd Configuration

Each file in the /etc/xinetd.d directory specifies a particular service for xinetd to manage. By default, scripts in this directory are disabled. The following code shows a sample of the /etc/xinetd.d/telnet configuration file, with this service disabled.

```
# default: on
# description: The telnet server serves telnet sessions; it uses
#           unencrypted username/password pairs for authentication.
service telnet
{
     flags           = REUSE
     socket_type     = stream
     wait            = no
     user            = root
     server          = /usr/sbin/in.telnetd
     log_on_failure  += USERID
     disable         = yes
}
```

This is a typical /etc/xinetd.d configuration file. The variables (and a few additional variables that you can use) are described in Table 10-3. This is a versatile configuration file; other fields are described in the man pages for xinetd.conf. Do read the xinetd.conf man page; the **only_from** and **no_access** directives may be of particular interest, as they can help you "configure host-based and user-based security for the service."

You can enable any xinetd service by changing **disable = yes** to **disable = no**.

TABLE 10-3 Standard Parameters for xinetd Configuration Files

Field	Description of Field Entry
flags	Supports different parameters for the service; **REUSE** is a default that supports continuous use of the service. Options include IPv6 to set this as a service for those types of networks.
socket_type	Specifies the communication stream.
wait	Set to **yes** for single-threaded applications or **no** for multithreaded applications.
user	Account under which the server should run.
group	Group under which the server should run.
server	The server program.
only_from	Hostname or IP address allowed to use the server. CIDR notation (such as 192.168.0.0/24) is okay.
no_access	Hostname or IP address not allowed to use the server. CIDR notation is okay.
log_on_failure	If there's a failed login attempt, this specifies the information sent to a log file.
disable	By default, set to **yes**, which disables the service.

exam Watch

Always remember to make sure that a service will be active after a reboot. The* chkconfig *servicename* on *command is one way to do this for xinetd services. Otherwise, anything you configure may not work after a system is rebooted.

There are two ways to activate an xinetd service. You can edit the configuration file directly by changing the **disable** field from **no** to **yes**. Then make the xinetd daemon reread the configuration files with the **service xinetd reload** command.

Alternatively, you can use the **chkconfig** *servicename* **on** command, which automatically makes this change and makes xinetd reread the configuration file.

EXERCISE 10-1

Configure xinetd

In this exercise, you will enable the Telnet service using xinetd. This exercise assumes that the telnet-server package is already installed. Before starting this exercise, try to establish a Telnet session using the command **telnet localhost**. If you're successful, Telnet was previously enabled; in that case, disable it first with the **chkconfig telnet off** command.

1. Edit /etc/xinetd.d/telnet and change the value of **disable** from **yes** to **no**.
2. Tell xinetd to reread its configuration file using this command:

   ```
   # service xinetd reload
   ```

3. Try the **telnet localhost** command again. It should work.
4. Try to log in with an incorrect username or password.
5. Log in to another terminal. What do you see when you run **utmpdump /var/log/wtmp**?
6. Now log in with a correct username and password.
7. What do you see in /var/log/messages?
8. Log out of the Telnet session. What do you see now in /var/log/messages?
9. Use the **chkconfig** command to disable Telnet. (Remember that the name of the service is telnet.) Try connecting to the Telnet server again. Do you have to restart or reload xinetd?
10. What happens when you use **chkconfig** to disable Telnet? Does it change the /etc/xinted.d/telnet configuration file?

11. For the security of this system, uninstall the associated package with the **rpm -e telnet-server** command. As suggested by the NSA, crackers can't exploit weaknesses in uninstalled software.

CERTIFICATION OBJECTIVE 10.04

TCP Wrappers

As suggested by its name, TCP Wrappers protects those services that communicate using the TCP protocol. It was originally designed to help protect services configured through the Extended Internet Super-Server just described. But TCP wrappers protection is no longer limited to such services; the protection can apply to all services statically and dynamically linked to the associated library wrapper file, libwrap.so.0.

The way TCP wrappers protects a service is defined in the /etc/hosts.allow and /etc/hosts.deny configuration files.

Is a Service Protected by TCP Wrappers?

The **strings** command can be used to identify those daemons protected by TCP Wrappers. It does so by listing the strings associated with various components of binary files. The string associated with TCP Wrappers is hosts_access. Daemons can be found in the /sbin and /usr/sbin directories. Thus, the quickest way to scan the daemons in these directories for the host_access string is with the following commands:

```
# strings /sbin/* | grep hosts_access
# strings /usr/sbin/* | grep hosts_access
```

The output depends on installed packages. One example is the SSH daemon, /usr/sbin/sshd. You can then use the full path to that daemon to confirm a link to the TCP Wrappers library, libwrap.0.so.

The library dependencies command, **ldd**, can list the libraries used by the **sshd** daemon. To identify those dependencies, run the following command:

```
# ldd /usr/sbin/sshd
```

But that's not convenient, as it returns the files for more than a couple of dozen library files. As an expert at the Linux command line, you should know how to pipe

that output to the **grep** command to see if it's associated with the TCP Wrappers library file: libwrap.so.0:

```
# ldd /usr/sbin/sshd | grep libwrap.0.so
```

And from the output, it's confirmed:

```
libwrap.so.0 => /lib64/libwrap.so.0 (0x00007f231674e000)
```

Now it's confirmed. You can use the TCP Wrappers configuration files to help protect the SSH service. That protection comes over and above any settings included in standard **iptables** command firewalls, the SSH server configuration file, SELinux, and so on. But such redundant protection is important in a layered security strategy.

TCP Wrappers Configuration Files

When a system receives a network request for a service linked to the libwrap.so.0 library, it passes the request on to TCP Wrappers. This system logs the request and then checks its access rules. If there are no limits on the particular host or IP address, TCP Wrappers passes control back to the service.

The key files are hosts.allow and hosts.deny. The philosophy is fairly straightforward: users and clients listed in hosts.allow are allowed access; users and clients listed in hosts.deny are denied access. As users and/or clients may be listed in both files, the TCP Wrappers system takes the following steps:

1. It searches /etc/hosts.allow. If TCP Wrappers finds a match, it grants access. No additional searches are required.
2. It searches /etc/hosts.deny. If TCP Wrappers finds a match, it denies access.
3. If the host isn't found in either file, access is automatically granted to the client.

You use the same access control language in both /etc/hosts.allow and /etc/hosts.deny files. The basic format for commands in each file is as follows:

```
daemon_list : client_list
```

The simplest version of this format is

```
ALL : ALL
```

This specifies all services and makes the rule applicable to all hosts on all IP addresses. If you set this line in /etc/hosts.deny, access is prohibited to all services. Of course, since that is read after /etc/hosts.allow, services in that file are allowed.

Of course, you can create finer-grained filters than just prohibiting access to ALL daemons from ALL systems. For example, the following line in /etc/hosts.allow allows the client with an IP address of 192.168.122.50 to connect to the local system through the Secure Shell:

```
sshd : 192.168.122.50
```

The same line in /etc/hosts.deny would prevent the computer with that IP address from using SSH to connect. If the same line exists in both files, /etc/hosts.allow takes precedence, and users from the noted IP address will be able to connect through SSH, assuming other security settings such as **iptables**-based firewalls allow it. You can specify clients a number of different ways, as shown in Table 10-4.

As you can see in Table 10-4, there are two different types of wildcards. **ALL** can be used to represent any client or service, and the dot (**.**) specifies all hosts with the specified domain name or IP network address.

You can set up multiple services and addresses with commas. Exceptions are easy to make with the **EXCEPT** operator. Review the following example excerpt from a /etc/hosts.allow file:

```
#hosts.allow
ALL : .example.com
sshd : 192.168.122.0/255.255.255.0 EXCEPT 192.168.122.150
rpc.mountd, in.tftpd : 192.168.100.100
```

TABLE 10-4 Sample Commands in /etc/hosts.allow and /etc/hosts.deny

Client	Description
.example.com	Domain name. Since this domain name begins with a dot, it specifies all clients on the example.com domain.
172.16.	IP address. Since this address ends with a dot, it specifies all clients with an IP address of 172.16.*x.y*.
172.16.72.0/255.255.254.0	IP network address with subnet mask. CIDR notation not recognized.
ALL	Any client, any daemon.
user@linux1.example.com	Applies to the specific user on the given computer.

The first line in this file is simply a comment. The next line opens **ALL** services to all computers in the example.com domain. The following line opens the SSH service to any computer on the 192.168.122.0 network, except the one with an IP address of 192.168.122.150. Then the mount and TFTP services are opened to the computer with an IP address of 192.168.100.100. You may want to add the localhost IP address network to the noted daemons in the /etc/hosts.allow file, as follows:

```
sshd : 127. 192.168.122.0/255.255.255.0 EXCEPT 192.168.122.150
rpc.mountd, in.tftpd : 127. 192.168.100.100
```

Otherwise, attempts to connect from the local system may be denied based on directives in the /etc/hosts.deny file that follows.

The code that follows contains a hosts.deny file to see how lists can be built to control access:

```
#hosts.deny
ALL EXCEPT in.tftpd : .example.org
sshd : ALL EXCEPT 192.168.122.150
ALL:ALL
```

The first line in the hosts.deny file is a comment. The second line denies all services except TFTP to computers in the example.org domain. The third line states that the only computer allowed to access the local SSH server has an IP address of 192.168.122.100. Finally, the last line is a blanket denial; all other computers are denied access to all services controlled by TCP Wrappers.

EXERCISE 10-2

Configure TCP Wrappers

In this exercise, you will use TCP Wrappers to control access to network resources. Since such controls are enabled by default, you shouldn't have to make any modifications to installed services.

1. Try to connect to the local telnet server using the address localhost. You may need to do several things first:
   A. Install the Telnet server service, from the telnet-server RPM.
   B. Activate the service with the **chkconfig telnet on** command.
   C. Allow Telnet through any active firewall, on the default port of 23.

D. Add the following line to /etc/hosts (substitute your computer's hostname for server1 and server1.example.com).

```
127.0.0.1   server1 server1.example.com   localhost.localdomain  localhost
```

E. Recognize that the Telnet service included with RHEL 6 does not normally allow root logins.

2. Edit /etc/hosts.deny and add the following line (don't forget to write the file):

```
ALL : ALL
```

3. What happens when you try to telnet to the address localhost?
4. Edit /etc/hosts.allow and add the following line:

```
in.telnetd : 127.0.0.1
```

5. Now what happens when you try to telnet to the address localhost? Do you need to add anything else to the /etc/hosts.allow file? Try to add the hostname localhost.
6. If other network services associated with TCP Wrappers are available on the local system, try restricting access to those daemons in the /etc/hosts.allow and /etc/hosts.deny files.
7. Undo any changes made when finished. If you agree that a clear-text communications protocol such as Telnet is inherently insecure, that should include the removal of the telnet-server package.

CERTIFICATION OBJECTIVE 10.05

Pluggable Authentication Modules

RHEL uses the Pluggable Authentication Modules (PAM) system as another layer of security primarily for administrative tools and related commands. PAM includes a group of dynamically loadable library modules that govern how individual applications verify their users. You can modify PAM configuration files to customize security requirement for different administrative utilities. Most PAM configuration files are stored in the /etc/pam.d directory.

PAM modules also standardize the user authentication process. For example, the login program uses PAM to require usernames and passwords at login. Open the /etc/pam.d/login file. Take a look at the first line:

```
auth [user_unknown=ignore success=ok ignore=ignore default=bad] \
 pam_securetty.so
```

To interpret, this line means that root users can log in only from secure terminals as defined in the /etc/securetty file, and unknown users are ignored.

on the job

A backslash in a command line "escapes" the meaning of the next character; in the preceding command, pam_securetty.so is added to the end of the command line. Due to limits in the format of this series, I've had to change the spacing of some lines and add backslashes to others.

The configuration files shown in the /etc/pam.d directory often have the same name as the command that starts the administrative utility. These utilities are "PAM aware." In other words, you can change the way users are verified for applications such as the console login program. Just modify the appropriate configuration file in the /etc/pam.d directory.

Configuration Files

Take a look at the configuration files in a typical /etc/pam.d directory, as shown in Figure 10-10. Depending on what's installed, you may see a somewhat different list of files.

As suggested earlier, most of the filenames in the /etc/pam.d directory are descriptive. Take a look at some of these files. In most cases, they refer to PAM modules. These modules can be found in the /lib64/security directory (on 32-bit systems, in the /lib/security directory). Excellent descriptions of each module can be found in the /usr/share/doc/pam-*versionnumber* directory, in the txt/ and html/ subdirectories. For example, the functionality of the pam_securetty.so module is described in the README.pam_securetty file.

In fact, there's an HTML version of the Linux-PAM System Administrators' Guide available in the /usr/share/doc/pam-*versionnumber*/html directory, starting with the Linux-PAM_SAG.html file.

FIGURE 10-10

PAM configuration files in the /etc/pam.d directory

```
[root@server1 ~]# ls /etc/pam.d/
atd                   halt              smartcard-auth-ac
authconfig            ksu               smtp
authconfig-gtk        login             smtp.postfix
authconfig-tui        newrole           sshd
chfn                  other             su
chsh                  passwd            sudo
config-util           password-auth     sudo-i
crond                 password-auth-ac  su-l
cups                  polkit-1          system-auth
cvs                   poweroff          system-auth-ac
eject                 reboot            system-config-authentication
fingerprint-auth      remote            system-config-date
fingerprint-auth-ac   rhn_register      system-config-kdump
gdm                   run_init          system-config-keyboard
gdm-autologin         runuser           system-config-network
gdm-fingerprint       runuser-l         system-config-network-cmd
gdm-password          setup             system-config-users
gdm-smartcard         smartcard-auth    xserver
[root@server1 ~]#
```

Control Flags

The PAM system divides the process of verifying users into four separate tasks. These are the four different types of PAM flags:

- **Authentication management (auth)** Establishes the identity of a user. For example, a PAM **auth** command decides whether to prompt for a username and/or a password. Related options may even grant group membership.
- **Account management (account)** Allows or denies access according to the account policies. For example, a PAM **account** command may deny access according to time, password expiration, or a specific list of restricted users.
- **Password management (password)** Manages other password policies. For example, a PAM **password** command may limit the number of times a user can try to log in before a console is reset.
- **Session management (session)** Applies settings for an application. For example, the PAM **session** command may set default settings for a login console.

The code shown in Figure 10-11 is from an example PAM configuration file, /etc/pam.d/login. Every line in all PAM configuration files is written in the following format:

```
module_type  control_flag  module_path  [arguments]
```

FIGURE 10-11

The PAM /etc/pam.d/login configuration file

```
#%PAM-1.0
auth [user_unknown=ignore success=ok ignore=ignore default=bad] pam_securetty.so
auth       include      system-auth
account    required     pam_nologin.so
account    include      system-auth
password   include      system-auth
# pam_selinux.so close should be the first session rule
session    required     pam_selinux.so close
session    required     pam_loginuid.so
session    optional     pam_console.so
# pam_selinux.so open should only be followed by sessions to be executed in the
user context
session    required     pam_selinux.so open
session    required     pam_namespace.so
session    optional     pam_keyinit.so force revoke
session    include      system-auth
-session   optional     pam_ck_connector.so
~
~
~
~
"/etc/pam.d/login" 16L, 728C
```

The **module_type**, as described previously, can be **auth**, **account**, **password**, or **session**. The **control_flag** determines what PAM does if the module succeeds or fails. The **module_path** specifies the location of the actual PAM module file. Finally, as with regular shell commands, you can specify arguments for each module.

The **control_flag** field requires additional explanation. It determines how the configuration file reacts when a module flags success or failure. The five different control flags are described in Table 10-5.

TABLE 10-5

PAM Control Flags

control_flag	Description
required	If the module works, the command proceeds. If it fails, PAM proceeds to the next command in the configuration file—but the command controlled by PAM will still fail.
requisite	Stops the process if the module fails.
sufficient	If the module works, the login or other authentication proceeds. No other commands need be processed.
optional	PAM ignores success or failure of this module, unless no other modules are used.
include	Includes all **module_type** directives from the noted configuration file; for example, if the directive is **password include system-auth**, this includes all password directives from the PAM system-auth file.

To see how control flags work, take a look at the commands from the /etc/pam.d/reboot configuration file:

```
auth    sufficient    pam_rootok.so
```

The first **auth** command checks the pam_rootok.so module. In other words, if the root user runs the **reboot** command, the **control_flag** is **sufficient**, and the other **auth** commands in this file are ignored. Linux runs the **reboot** command. This is explained in the README.pam_rootok file in the /usr/share/doc/pam-*versionnumber*/txts directory.

```
auth    required    pam_console.so
```

Given the purpose of the first line, this second **auth** command is run only for nonroot users, to govern permissions within the console. In this case, it just confirms console ownership to any user who is logged in to that console.

```
#auth    include    system-auth
```

The third line is commented out by default. If you make this line active, it includes the commands from the system-auth configuration file, which requires root user privileges. Remote users who connect with root privileges are still allowed to reboot the system.

```
account    required    pam_permit.so
```

The module associated with the **account** command (pam_permit.so) accepts all users, even those who've logged in remotely. In other words, this configuration file would allow any root user, local or remote, to reboot the Linux system, unless rejected by a previous directive.

The Format of a PAM File

This section is a little complex. It starts with the /etc/pam.d/login configuration file shown in Figure 10-12. In addition, as the file includes references to the /etc/pam.d/system-auth configuration file, you'll need to go back and forth between files to follow along with this section.

When a user opens a text console and logs in, Linux goes through this configuration file line by line. As previously noted, the first line in /etc/pam.d/login limits root user access to secure terminals as defined in the /etc/securetty file:

```
auth [user_unknown=ignore success=ok ignore=ignore default=bad] \
 pam_securetty.so
```

FIGURE 10-12 The PAM /etc/pam.d/system-auth configuration file

```
#%PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth        required      pam_env.so
auth        sufficient    pam_fprintd.so
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 500 quiet
auth        required      pam_deny.so

account     required      pam_unix.so
account     sufficient    pam_localuser.so
account     sufficient    pam_succeed_if.so uid < 500 quiet
account     required      pam_permit.so

password    requisite     pam_cracklib.so try_first_pass retry=3 type=
password    sufficient    pam_unix.so sha512 shadow nullok try_first_pass use_au
thtok
password    required      pam_deny.so

session     optional      pam_keyinit.so revoke
session     required      pam_limits.so
session     [success=1 default=ignore] pam_succeed_if.so service in crond quiet
use_uid
session     required      pam_unix.so
~
"/etc/pam.d/system-auth" 22L, 937C
```

The next line includes the **auth** commands from the system-auth PAM configuration file:

```
auth  include     system-auth
```

The system-auth configuration file shown in Figure 10-12 includes five **auth** directives:

```
auth        required      pam_env.so
auth        sufficient    pam_fprintd.so
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 500 quiet
auth        required      pam_deny.so
```

In order, they set up environment variables, check authentication via a fingerprint reader (pam_fprintd.so) if available, and check passwords (pam_unix.so). The **sufficient** flag associated with those modules means that authentication works if a valid fingerprint or password has been entered. The User ID of the account must be

exam Watch

If the /etc/nologin file exists, regular users are not allowed to log in to the local console. Any regular user that tries to log in gets to read the contents of /etc/nologin as a message.

500 and above. If these conditions are not met, the user is locked out (pam_deny.so).

Now return to the /etc/pam.d/login file. The next line, which looks for an **account** module type, checks for accounts not allowed to log in as listed in the /etc/nologin file:

```
account   required    pam_nologin.so
```

The following account module includes the **account** modules from the /etc/pam.d/system-auth configuration file:

```
account    include     system-auth
```

These are the **account** module type lines from /etc/pam.d/system-auth:

```
account   required    pam_unix.so
account   sufficient  pam_localuser.so
account   sufficient  pam_succeed_if.so uid < 500 quiet
account   required    pam_permit.so
```

The first line refers to the pam_unix.so module in the /lib/security directory, which brings up the normal username and password prompts. Based on the pam_localuser.so module, users found in /etc/passwd are automatically accepted. Based on the pam_succeed_if.so module, service users (with user IDs less than 500) are automatically logged in, without messages (quiet). The pam_permit.so module always returns success.

Now return to the /etc/pam.d/login file. The next line is a **password** module, which includes other password module type lines from the /etc/pam.d/system-auth file:

```
password  include     system-auth
```

These are the **password** module type lines from /etc/pam.d/system-auth:

```
password requisite   pam_cracklib.so try_first_pass retry=3
password sufficient  pam_unix.so sha512 shadow nullok try_first_pass \
use_authok
password required    pam_deny.so
```

The first command from this list allows the use of a previously successful password (**try_first_pass**) and then sets a maximum of three retries. The next command encrypts passwords using the SHA512 hash, supports the Shadow Password Suite

described in Chapter 8, allows the use of null (zero-length) passwords, allows the use of a previously successful password (**try_first_pass**), and prompts the user for a password (**use_authok**). The **password required pam_deny.so** directive is trivial; as noted in README.pam_deny in the /usr/share/doc/pam-*versionlevel*/txt directory, that module always fails, so PAM moves on to the next directive.

Finally, there are eight **session** commands in the /etc/pam.d/login file. Take them four at a time:

```
session    required     pam_selinux.so close
session    required     pam_loginuid.so
session    optional     pam_console.so
session    required     pam_selinux.so open
```

The first and fourth lines deactivate (**pam_selinux.so close**) and reactivate SELinux (**pam_selinux.so open**), just for this part of the login process. The second line (**pam_loginuid.so**) records the user ID for audits. The third line gives ownership of the process to the user logged in to the console (**pam_console.so**).

```
session    required     pam_namespace.so
session    optional     pam_keyinit.so force revoke
session    include      system-auth
session    optional     pam_ck_connector.so
```

The fifth **session** command makes sure the user's home directory has standard /etc/skel files. The following command, even though it's listed as optional, revokes any existing unique session keyring. Jumping ahead, the last of this group of commands allows logins to work with the Console Kit described in Chapter 9. The command that precedes it includes the following **session** module–type commands from the system-auth file:

```
session     optional      pam_keyinit.so revoke
session     required      pam_limits.so
session     [success=1 default=ignore] pam_succeed_if.so service in crond quiet
use_uid
session     required      pam_unix.so
```

The first of these commands repeats the revoke of the keyring already accomplished in the main /etc/pam.d/login file. The next command sets limits (pam_limits.so) on individual users through /etc/security/limits.conf. The following command relates to cron jobs. The final command logs the result when the user logs out.

EXERCISE 10-3

Configure PAM

In this exercise, you can experiment with some of the PAM security features of Red Hat Enterprise Linux 6.

1. Make a backup copy of /etc/securetty with the following command:

   ```
   # cp /etc/securetty /etc/securetty.sav
   ```

2. Edit /etc/securetty and remove the lines for tty3 through tty11. Save the changes and exit.
3. Use ALT-F3 (CTRL-ALT-F3 if you're running X Window) to switch to virtual console number 3. Try to log in as root. What happens?
4. Repeat Step 3 as a regular user. What happens? Do you know why?
5. Use ALT-F2 to switch to virtual console number 2 and try to log in as root.
6. Review the messages in /var/log/secure. Do you see where you tried to log in as root in virtual console number 3?
7. Restore the original /etc/securetty file with the following command:

   ```
   # mv /etc/securetty.sav /etc/securetty
   ```

One thing to remember is that the /etc/securetty file governs the consoles from which you can log in to Linux as the root user. Therefore, the changes that were made do not affect regular (nonroot) users.

PAM and User-Based Security

In this section, you'll learn how to configure PAM to limit access to specific users. The key to this security feature is the pam_listfile.so module. The location varies with architecture; it may be in the /lib64/security or /lib/security directories. If you've installed the vsFTP server, the /etc/pam.d/vsftpd file includes an example of this module.

As described earlier, four settings are available for each PAM configuration command. First, the following line clears out any existing keyrings, making sure the authentication options that follow are actually used.

```
session optional pam_keyinit.so  force revoke
```

To make sure that the command respects what's done with this module, this directive should come next:

```
auth required pam_listfile.so
```

The way PAM limits user access is in the last part of the command—in the details. For example, if you added the following line to a PAM configuration file, access to the associated tool would be limited to any users listed in /etc/special:

```
auth required pam_listfile.so item=user sense=deny \
file=/etc/vsftpd/ftpusers onerr=succeed
```

To understand how this works, break this command into its component parts. You already know the first three parts of the command from the previous section. The switches that are shown are associated with the pam_listfile.so module, as described in Table 10-6.

TABLE 10-6

Options for the pam_listfile.so Module

pam_listfile Switch	Description
item	This switch can be used to limit access to a terminal (**tty**), users in a specific file (**user**), groups (**group**), or more.
sense	If the item is found in the specified **file**, take the noted action. For example, if the user is in /etc/special, and **sense=allow**, then this command allows use of the specified tool.
file	Configures a file with a list, such as **file = /etc/special**.
onerr	If there is a problem, tell the module what to do. The options are **onerr=succeed** or **onerr=fail**.

Thus, for the specified command (**onerr=succeed**), an error, strangely enough, returns success (**item=user**), based on a specific list of users. If the user is in the specified list (**file=/etc/special**), allow that user (**sense=allow**) to access the specified tool. To see how this works, run through the steps in Exercise 10-4.

exam Watch

Make sure you understand how Red Hat Enterprise Linux handles user authorization through the /etc/pam.d configuration files. When you test these files, make sure you create a backup of everything in PAM before making any changes, because any errors that you make to a PAM configuration file can disable your system completely (PAM is that secure).

EXERCISE 10-4

Use PAM to Limit User Access

You can also use the PAM system to limit access to regular users. In this exercise, you'll limit access by adding one or more users to the /etc/nologin file. It should work hand-in-hand with the default /etc/pam.d/login security configuration file, specifically the following line:

```
account  required  pam_nologin.so
```

1. Look for an /etc/nologin file. If it doesn't already exist, create one with a message such as:

   ```
   I'm sorry, access is limited to the root user
   ```

2. Access another terminal with a command such as CTRL-ALT-F2. Try logging in as a regular user. What do you see?
3. If the message flashes by too quickly for you, log in as the root user. You'll see the same message; but as the root user, you're allowed access.
4. Inspect the /var/log/secure file. Did your system reject the attempted login from the regular user? What were the associated messages for the root user?

SCENARIO & SOLUTION	
You have only one official IP address, but you need to provide Internet access to all of the systems on your LAN. Each computer on the LAN has its own private IP address.	Use **iptables** to implement IP masquerading. Make sure IP forwarding is active.
You have installed an SSH server on a corporate network and want to restrict access to certain departments. Each department has its own subnet.	Use the /etc/hosts.deny file in the tcp_wrappers package to block SSH access, using the sshd daemon, to the unwanted subnets. A better alternative would be to use /etc/hosts.allow to support access to desired departments, and then use /etc/hosts.deny to deny access to everyone else. Similar options are possible based on **iptables** firewall rules.
You want to modify the commands associated with halting and rebooting your computer so that they're accessible only to the root user.	Set up the appropriate Pluggable Authentication Module configuration files in /etc/pam.d to use the **system-auth** module.
You want to modify the local firewall to defend against ICMP attacks such as **ping** command floods.	Modify the firewall to reject or deny certain types of ICMP packets. Many such settings are available with the Firewall Configuration tool.

CERTIFICATION OBJECTIVE 10.06

Secure Files and More with GPG2

With the importance of security on current networks, you should know how to encrypt files for secure transmission. The computer standard for file encryption is known as Pretty Good Privacy (PGP). The open source implementation of PGP is known as the GNU Privacy Guard (GPG). The version released for RHEL 6 is more advanced and capable, documented as GPG version 2 (GPG2) You've likely already used GPG2 to verify the authenticity of RPM packages as discussed in Chapter 7. This section takes such checks one step further; you'll generate private and public keys, and use those keys to encrypt and decrypt selected files.

While GPG is not listed in the RHCE objectives, it's a security topic consistent with other security objectives discussed in this book. It is also part of Red Hat's prep

course for the RHCE, RH254. I believe it's an excellent topic that might be included in future versions of the RHCE exam.

GPG2 Commands

There's a new version of the GPG package, implemented for RHEL 6. It includes a more modular approach to encryption and authentication. There's even a related package used for smart card authentication. But that's not the point of the new **gpg2** commands. Available GPG commands are briefly described in Table 10-7.

Table 10-7 is just intended to describe the range of capabilities associated with the RHEL 6 GPG2 packages. The focus of this section is on the encryption and decryption of files.

Current GPG2 Configuration

While the man page for the **gpgconf** command suggests that it's just used to modify the directory with associated configuration files, that command does more. By itself, it defaults to the **--list-components** switch, which specifies the full path to related executable files. With the **--check-programs** switch, it makes sure all related programs can be executed. It can also be used to check the syntax of a GPG2

TABLE 10-7 GPG2 Commands

Command	Description
gpg	Executable file soft-linked to the **gpg2** command
gpg2	The GPG2 encryption and signing tool
gpg-agent	GPG2 key management command
gpgconf	Status command for GPG2 components
gpg-connect-agent	Communicate with an active GPG2 agent
gpg-error	Command to interpret a GPG2 error number
gpg-error-config	Command to build applications based on a GPG2 error
gpgkey2ssh	Conversion command for GPG2 keys for SSH
gpgparsemail	Under development
gpgsplit	Command to split a GPG2 message into packets
gpgv	Executable file soft-linked to the **gpg2v** command
gpgv2	Command to verify GPG signatures; requires a signature file
gpg-zip	Command to encrypt and sign files into an archive

configuration file. One typical option is in the current user's home directory, in the .gnupg/ subdirectory. Another typical option is in the /etc/gnupg directory.

GPG2 Encryption Options

The development of a GPG2 key includes a choice of three different cryptographic algorithms. Each of these algorithms include a public and a private key. The public key can be distributed to others, for use in encrypting files and messages. The private key is used by the owner, and is the only way to decrypt the file or message.

- **RSA** Named for its developers, Rivest, Shamir, and Adelman. While typical RSA keys are 1024 or 2048 bits in length, they can be up to 4096 bits. Shorter keys of 512 bits have been cracked. It is in the public domain.
- **DSA** The Digital Signature Algorithm. Owned by the U.S. National Institute of Science and Technology, it has been made available for worldwide use, royalty-free. This is a U.S. Government standard that uses Secure Hash Algorithm (SHA) versions SHA-1 and SHA-2 as message digest hash functions. SHA-1 is being phased out; SHA-2 includes four hash functions with message digests of up to 512 bits, also known as SHA-512, the same hash as is now used for the RHEL 6 shadow password suite.
- **ElGamel** Developed by Taher Elgamel, this probabilistic encryption scheme predates the others. The choices for the number of bits for encryption keys range from 1024 to 3072 bits. While it was never patented, it may be covered under the patent for the Diffie-Hellman key exchange protocol.

Generate a GPG2 Key

The **gpg2 --gen-key** command can be used to set up key pairs with one of four types of encryption schemes. Before running the command, be prepared with answers to the following questions:

- The number of bits for the encryption keys. Normally, the maximum number of bits is 4096, but an encryption key that complex may take a number of minutes to develop.
- The desired lifetime of the keys. Especially if you set up keys with a smaller number of bits, you should assume that a determined cracker would be able to decrypt the key within some number of months or weeks.

- A name, an e-mail address, and a comment. While the name and e-mail address do not have to be real, they will be seen by others as part of the public key.
- A passphrase. Good passphrases should include spaces, lower- and uppercase letters, numbers, and punctuation.

As given, the **gpg2 --gen-key** command prompts for one of four different encryption schemes. As suggested by the (sign only) label associated with choices 3 and 4, those options work just as digital signatures, not for encryption.

```
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection?
```

All four options follow a similar sequence of steps. For example, if you select option 2, the following output appears:

```
DSA keys may be between 1024 and 3072 bits long.
What keysize do you want? (2048)
```

The default is 2048 bits, which is selected if you just press ENTER. The command then prompts for a key lifetime:

```
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      < >  = key expires in n days
      < >w = key expires in n weeks
      < >m = key expires in n months
      < >y = key expires in n years
Key is valid for? (0) 2m
```

In this case, I've selected two months. The command responds with a date and time two months into the future and prompts for confirmation.

```
Key expires at Sun 03 Apr 2011 11:14:17 AM PDT
Is this correct? (y/N) y
```

At this point, the **gpg2** command prompts for identifying information for the key. The "User ID" requested here is not related to the UID in the standard Linux authentication database. In this example, I've responded to the prompts in bold:

```
Real name: Michael Jang
Email address: michael@example.com
Comment: DSA and Elgamel key
You selected this USER-ID:
    "Michael Jang (DSA and Elgamel key) <michael@example.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
```

The system should now prompt for a passphrase. If you're working in the GNOME Desktop Environment, the window shown in Figure 10-13 would appear.

As you type in a passphrase, the system evaluates its quality, based on the length, use of upper- and lowercase characters, numbers, and punctuation. Of course, a similar window appears a second time to allow you to type in the passphrase a second time.

At this point, the **gpg2** command goes to work. Especially with larger keys, it may seem to pause for a few minutes with a message about creating random bytes. You might need to run some other programs to stimulate the process. When complete, it displays a message similar to the following:

```
gpg: key D385AFDD marked as ultimately trusted
public and secret key created and signed.
```

To make sure the public and private keys were actually written, run the following command:

```
$ gpg2 --list-key
```

The output should include the latest key, along with any others created from the user's home directory, in the .gnupg/ subdirectory. For the given options, if this is the only key pair on the local account, you'll see something similar to the following output:

```
/home/michael/.gnupg/pubring.gpg
--------------------------------
pub   2048D/1665FE84 2011-02-02 [expires: 2011-04-03]
uid   Michael Jang (DSA and ElGamel) <michael@example.com>
sub   2048g/EEC05210 2011-02-02 [expires: 2011-04-03]
```

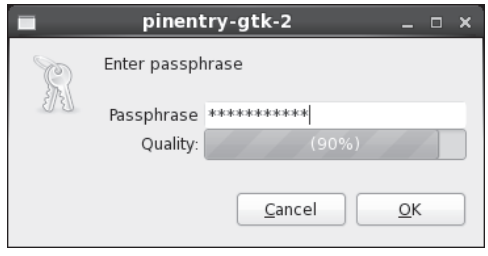


FIGURE 10-13

Prompting for a passphrase

Use a GPG2 Key to Secure a File

Now you can send the public key to a remote system. To start the process, you'll need to export the public key. For the key pair just created, you could do so with the following command. Substitute your name for "Michael Jang."

```
$ gpg2 --export Michael Jang > gpg.pub
```

Now copy that key to a remote system. The delivery vehicle, such as e-mail, a USB stick, or the **scp** command shown here, is not important. This particular command from user michael's account would copy the gpg.pub key to user michael's home directory on the tester1.example.com system. If you prefer, substitute the IP address.

```
$ scp gpg.pub tester1.example.com:
```

Now go to the remote system, in this case, tester1.example.com. Log in to user michael's account (or the account home directory to which you copied the gpg.pub key). If desired, you can even log in to that system remotely with a command like **ssh**. Once connected to that system, first check for existing GPG keys with the following command:

```
$ gpg2 --list-key
```

If this is a system without previous GPG keys, the list should be empty, and nothing will appear in the output to this command. Now import the gpg.pub file into the list of local GPG keys with the following command:

```
$ gpg2 --import gpg.pub
```

Confirm the import by running the **gpg2 --list-key** command again.

Now on the remote system, you can encrypt a file with the **gpg2** command. The following example encrypts the local keepthis.secret file:

```
$ gpg2 --out underthe.radar --recipient 'Michael Jang' --encrypt
keepthis.secret
```

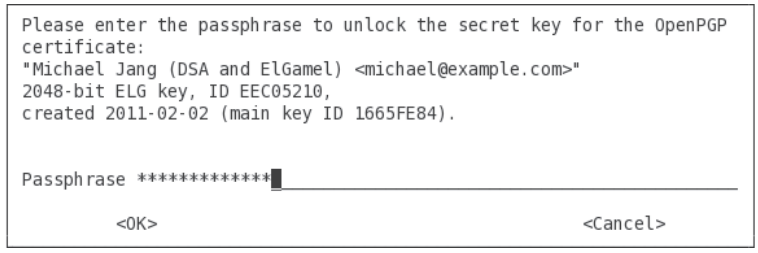
The username in this case is 'Michael Jang'. If you've just imported a private key, the username as shown in the output to the **gpg2 --list-key** command may be different. Substitute as appropriate.

Now when the underthe.radar file is copied to the original system, server1.example.com, you can start the decryption process with the private key with the following command:

```
$ gpg --out keepthis.secret --decrypt underthe.radar
```

FIGURE 10-14

Prompting for a passphrase for decryption



In a console, you'd be prompted for the passphrase created earlier, with a screen similar to that shown in Figure 10-14.

CERTIFICATION SUMMARY

To help defend the data, the services, and the systems on a network, Linux provides layers of security. If a service is not installed, a cracker can't use it to break into a system. Those systems that are installed should be kept up to date. Such services can be protected by firewalls, along with host- and user-based security options. Many services include their own layers of security. RHEL 6 incorporates several recommendations from the NSA, including SELinux.

Firewalls based on the **iptables** command can regulate and protect gateways as well as individual systems. That same command can be used to set up packet forwarding, as well as masquerading of private networks. Such options can be configured directly in the /etc/sysconfig/iptables file, or set up with the help of the Firewall Configuration tool.

Some services are still configured through the Extended Internet Super-Server, xinetd. Those systems can be regulated and protected through the /etc/xinetd.conf file and service-specific configuration files in the /etc/xinetd.d directory.

Those systems connected to the TCP Wrappers library can be protected by appropriate settings in the /etc/hosts.allow and /etc/hosts.deny files. If there is a conflict, /etc/hosts.allow is read first. Regulation through TCP Wrappers is possible by user or host.

PAM supports user-based security for a number of administrative tools. They're configured individually through files in the /etc/pam.d directory. These files refer to modules in the /lib64/security directory.

Linux supports encryption with the help of GPG. RHEL 6 includes GPG2 for this purpose. It includes commands like **gpg2** to set up private/public key pairs using the RSA, DSA, or ElGamel schemes.

TWO-MINUTE DRILL

The following are some of the key points from the certification objectives in Chapter 10.

The Layers of Linux Security

- Bastion systems are more secure, as they're configured with a single service. With virtualization, bastion systems are now a practical option even for smaller organizations.
- You may choose to automate at least security updates with the Software Updates Preference tool.
- Many services include their own security options in their configuration files.
- Host-based security can be configured by domain name or IP address.
- User-based security includes specified users and groups.
- The PolicyKit can regulate security of administrative tools run from the GNOME desktop environment.

Firewalls and Network Address Translation

- The main firewall configuration command is **iptables**, configured in the /etc/sysconfig/iptables file.
- With **iptables**, you can regulate packet traffic into, out of, and forwarded through a system.
- With **iptables**, you can also masquerade the IP addresses from one network on an outside network such as the Internet.
- Listed **iptables** options can also be configured with the help of the Firewall Configuration tool, which you can start with the **system-config-firewall** command.

The Extended Internet Super-Server

- Some services are still regulated by the xinetd service.
- Services configured can be activated and protected through individual configuration files in the /etc/xinetd.d directory.

TCP Wrappers

- The **strings /sbin/*** and **strings /usr/sbin/*** commands can identify daemons with the hosts_access setting, to identify services that can be regulated by TCP Wrappers.
- You can confirm regulation by TCP Wrappers with the **ldd** command, applied to the full path to the daemon. The TCP Wrappers library file for 64-bit systems is /lib64/libwrap.so.0.
- Clients and users listed in /etc/hosts.allow are allowed access; clients and users listed in /etc/hosts.deny are denied access.
- Services can also be configured in /etc/hosts.allow and /etc/hosts.deny. Remember to use the actual executable name of the daemon, normally in /usr/sbin, such as **in.tftpd**.

Pluggable Authentication Modules

- RHEL 6 uses the Pluggable Authentication Modules (PAM) system to check for authorized users.
- PAM modules are called by configuration files in the /etc/pam.d directory. These configuration files are usually named after the service or command that they control.
- There are four types of PAM modules: authentication, account, password, and session management.
- PAM configuration files include lines that list the module_type, the control_flag, and the path to the actual module, followed by arguments.
- PAM modules are well documented in the /usr/share/doc/pam-*versionnumber*/txts directory.

Secure Files and More with GPG2

- GPG is the Linux implementation of PGP.
- RHEL 6 includes more advanced encryption known as GPG2.
- The **gpgconf** command can reveal the current GPG2 configuration.
- GPG2 encryption can use the DSA, RSA, and ElGamel schemes.
- GPG2 keys can be created with the **gpg2 --gen-key** command, and listed with the **gpg2 --list-key** command.

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 Layers of Linux Security

1. What security option is best for a service that isn't currently required on a system?

   _____

Firewalls and Network Address Translation

2. Consider the following command:

   ```
   # iptables -A INPUT -s 192.168.77.77 -j REJECT
   ```

   Once saved to a firewall, what effect will this have when the client with an IP of 192.168.77.77 tries to connect to this system?

   _____

3. What file includes rules associated with the **iptables** command?

   _____

4. You are setting up a small office and would like to provide Internet access to a small number of users but don't have the money for dedicated IPv4 address for each system on the network. What can you do?

   _____

5. What **iptables** command switch sets up masquerading?

   _____

The Extended Internet Super-Server

6. In what directory are files associated with individual xinetd services located?

_____

7. What command rereads the configuration files associated with the xinetd service?

_____

TCP Wrappers

8. You are using the xinetd program to start services. With TCP Wrappers configuration files, how could you limit Telnet access to clients on the 192.168.170.0 network? Hint: The telnet daemon, when installed, is in /usr/sbin/telnetd.

_____

9. What happens to a service if you allow the service in /etc/hosts.allow and prohibit it in /etc/hosts.deny?

_____

Pluggable Authentication Modules

10. What are the four basic PAM module types?

_____

_____

_____

_____

11. You are editing the PAM configuration file by adding a module. Which control flag immediately terminates the authentication process if the module succeeds?

_____

Secure Files and More with GPG

12. What command lists the GPG public keys loaded on the current local account?

_____

LAB QUESTIONS

Several of these labs involve exercises that can seriously affect a system. You should do these exercises on test machines only. The second Lab of Chapter 1 sets up KVM for this purpose. However, some readers may not have hardware that supports KVM. Options to KVM include virtual machine solutions such as VMware, available from www.vmware.com, or Virtualbox, open-source edition, available from www.virtualbox.org.

Red Hat presents its exams electronically. For that reason, the labs for this chapter are available from the CD that accompanies the book, in the Chapter10/ subdirectory. It's available in .doc, .html, and .txt formats, with filenames starting with 56510-labs. In case you haven't yet set up RHEL 6 on a system, refer to the first lab of Chapter 2 for installation instructions. However, the answers for each lab follows the Self Test answers for the fill-in-the-blank questions.

SELF TEST ANSWERS

The Layers of Linux Security

1. The security option that is best for a service that isn't currently required on a system is to not install that service.

Firewalls and Network Address Translation

2. Based on the given command, any connection attempt (including packets sent with the **ping** command) from the 192.168.77.77 system is rejected.
3. Rules associated with **iptables**-based firewalls are stored in /etc/sysconfig/iptables.
4. To set up a small office while providing Internet access to a small number of users, all you need is one dedicated IP address. The other addresses can be on a private network. Masquerading makes this possible.
5. The **iptables** command switch that sets up masquerading is **-t nat**.

The Extended Internet Super-Server

6. The configuration files for individual xinetd services are stored in the /etc/xinetd.d directory.
7. Several acceptable options are available to reread the configuration files associated with the xinetd service. The standard option is **/etc/init.d/xinetd reload**; the **service xinetd reload** command is functionally equivalent. While you could substitute **restart** for **reload**, that's an inferior answer. A "restart" kicks off any connected users.

TCP Wrappers

8. You are using the xinetd program to start services. To limit Telnet access to clients on the 192.168.170.0 network, you'd allow access to the network in /etc/hosts.allow and deny it to all others in /etc/hosts.deny. As /usr/sbin is in the root user path, you can cite the **telnetd** daemon directly and add the following directive to /etc/hosts.allow:

```
telnetd : 192.168.0.170/255.255.255.0
```

Then add the following to /etc/hosts.deny:

```
telnetd : ALL
```

9. If you allow a service in /etc/hosts.allow and prohibit it in /etc/hosts.deny, the service is allowed.

Pluggable Authentication Modules

10. The four basic PAM types are: **auth**, **account**, **password**, and **session**. The **include** type refers to one or more of the other PAM types in a different file.
11. The **sufficient** control flag immediately terminates the authentication process if the module succeeds.

Secure Files and More with GPG

12. The command that lists currently loaded public keys is **gpg2 --list-key**. The **gpg --list-key** command is acceptable.

LAB ANSWERS

Lab 1

Verifying this lab should be straightforward. If it works, you should be able to confirm with the following command on the tester1.example.com system:

```
$ gpg2 --list-keys
```

It should include the GPG2 public key just imported to that system. Of course, if the encryption, file transfer, and decryption worked, you should also be able to read the decrypted 56510-labs.txt file in a local text editor.

Lab 2

This lab is somewhat self-explanatory, in that it can help you think about how to make a system more secure. As discussed in the chapter, it starts with a minimal installation. The minimal installation of RHEL 6 happens to include the SSH server, for remote administrative access.

While RHEL 6 has greatly reduced the number of standard services installed, most users will find some services that are not required. For example, how many administrators actually need Bluetooth services for a RHEL 6 system installed on a virtual machine?

Lab 3

If you want to set up an RHEL computer as a secure web server, it's a straightforward process described in Chapter 14. But firewall configuration is part of the process covered in this chapter. To that end, you'll want to set up a firewall to block all but the most essential ports. This should include TCP/IP ports 80 and 443, which allow outside computers to access local regular and secure web services. Open ports should also include port 22, for SSH communication.

The easiest way to set this up is with the Red Hat Firewall Configuration tool, which you can start with the **system-config-firewall** command. Once in the Firewall Configuration tool, take the following steps, which vary slightly between the GUI- and console-based versions of the tool.

1. Customize the firewall.
2. Select the Trusted Services window. (If you're in the text-based tool, click Customize to open the Firewall Configuration – Customize window.) Activate the WWW (HTTP) option. This allows access from outside the local computer to the local regular web site. Activate the Secure WWW (HTTPS) option as well. Make sure the SSH option remains active.
3. Make sure to apply any changes, and exit from the Firewall Configuration tool.
4. Enter the following command to check the resulting firewall:

```
# iptables -L
```

5. Once you've configured a web service as described in Chapter 14, users will be able to access both the regular and secure web servers from remote systems.

Lab 4

Several steps are required to set up any xinetd service such as Telnet. You'll need to modify the xinetd Telnet configuration file. The following steps demonstrate three different methods to limit access to the noted system on IP address 192.168.122.150. Any of the three methods would be acceptable. These methods secure Telnet in three ways: in the /etc/xinetd.d/telnet configuration file, through TCP Wrappers, or with the appropriate firewall commands. In a "real-world" scenario, you might use all three methods in a layered security strategy. These steps assume you're performing this lab on the server1.example.com system.

1. Make sure that the telnet-server RPM is installed. Back up the current version of the /etc/xinetd.d/telnet configuration file.
2. Activate Telnet. Use the **chkconfig telnet on** command to revise the /etc/xinetd.d/telnet configuration script.

3. Edit the /etc/xinetd.d/telnet configuration file. Add the **only_from = 192.168.122.150** line, which represents the tester1.example.com system.
4. Save the configuration file and reload the xinetd service script with the **service xinetd reload** command. Try accessing Telnet from the local computer. What happens?
5. Try accessing Telnet from the computer with the IP address of 192.168.122.150. What happens? Try again from a different system on the LAN.
6. Restore the previous /etc/xinetd.d/telnet configuration file. Don't forget to reload the xinetd service script with the **service xinetd reload** command.
7. Back up the current version of the /etc/hosts.deny file. Open that file in a text editor. Add the **telnetd : ALL EXCEPT 192.168.122.150** line.
8. Try accessing Telnet from the computer with the IP address of 192.168.122.50. What happens? Try again from a different computer on your LAN.
9. Restore the previous /etc/hosts.deny file.
10. Save any existing **iptables** chains. Back up the current /etc/sysconfig/iptables file.
11. Flush current firewall rules with the **iptables -F** command.
12. Block the Telnet port, 23, for all IP addresses except 192.168.122.150 with the **iptables -A INPUT -s ! 192.168.122.150 -p tcp --dport 23 -j DROP** command.
13. Activate the firewall with the **/etc/init.d/iptables restart** command.
14. Try accessing the Telnet server from the computer with the IP address of 192.168.122.150. What happens? Try again from a different computer on the LAN.
15. Restore any previous firewall rules; restore the original version of /etc/sysconfig/iptables from backup. Reload the firewall with the **/etc/init.d/iptables restart** command.
16. Bonus: Repeat these commands for the SSH service on port 22.

Lab 5

To confirm that TCP Wrappers can be used to help protect the SSH service, run the following command:

```
# ldd /usr/sbin/sshd | grep libwrap
```

Output, which includes a reference to the libwrap.so.0 library, confirms a library link to the TCP Wrappers library. In general, it's safest to deny access to all services by including the following entry in the /etc/hosts.deny file:

```
ALL : ALL
```

You can then set up access to the SSH service with a line like the following in the /etc/hosts.allow file:

```
sshd : 192.168.122.50
```

While in most cases, the use of the fully qualified domain name for the noted IP address (server1.example.com) should work too, the use of the IP address is often appropriate. Limits by IP address don't depend on connections to DNS servers.

Of course, this is not the only way to limit access to the SSH to one system. It's possible within the /etc/hosts.deny file with a directive such as the following:

```
sshd : ALL EXCEPT 192.168.122.50
```

It's possible to set this up with other security options such as **iptables** command–based firewalls.

Lab 6

Before this lab can work, however, you'll need to activate one SELinux boolean, ftp_home_dir. It's listed in the SELinux Management tool as "Allow ftp to read and write files in the user home directories." While this boolean setting is covered in Chapter 11, the basic management of SELinux, including the activation of boolean settings, is an RHCSA skill covered in Chapter 4. Therefore, with the key boolean identified, you should be able to set up vsFTP as described.

The description in this lab should point you to the /etc/pam.d/vsftpd configuration file. The model command line in this file is

```
auth required  pam_listfile.so item=user sense=deny file=/etc/vsftpd/ftpusers
onerr=succeed
```

which points to the /etc/vsftpd/ftpusers file, a list of users to "deny" access. As the conditions in the lab suggest that you need a list of (one) user to which access is to be allowed, a second line of a similar type in this file is appropriate. To verify this lab, I included the following line:

```
auth required  pam_listfile.so item=user sense=allow file=/etc/vsftpd/testusers
onerr=succeed
```

which allows all users listed in the /etc/vsftpd/testusers file. The **onerr=succeed** directive means that the vsFTP server still works if there's an error elsewhere in the line. For example, if there is no testusers file in the /etc/vsftpd directory, the directives in this line are forgiving, allowing the conditions for the **auth** module type to succeed.

As an experiment, try this lab with the boolean ftp_home_dir variable set and unset. That should demonstrate the power of SELinux and serve as an appropriate preview of Chapter 11.