# Chapter 16

## NFS Secured with Kerberos

**CERTIFICATION OBJECTIVES**

| | | | |
|---|---|---|---|
| 16.01 | The Network File System (NFS) Server | ✓ | Two-Minute Drill |
| 16.02 | Test an NFS Client | Q&A | Self Test |
| 16.03 | NFS with Kerberos | | |

Linux is designed for networking. It allows you to share files in two major ways: Samba, covered in Chapter 15, and the Network File System (NFS). RHEL 7 does not include GUI tools for NFS, but that is not a problem because NFS configuration files are relatively simple.

This chapter starts with a description of NFS, a powerful and versatile way of sharing data between servers and workstations. A default installation of RHEL 7 includes an NFS client, which supports connections to NFS servers.

An NFS server can limit access to clients based on their hostnames or IP addresses. In addition, NFS trusts the UIDs sent by clients to verify file permissions. This provides only a basic level of security, which may not be adequate for some organizations. But if used in conjunction with Kerberos, NFS can authenticate access to network shares and provide data encryption. This chapter will explain how to set up such configuration.

Take the time you need to understand the configuration files associated with the NFS service and Kerberos, and practice making them work on a Linux computer. In some cases, two or three computers (such as the KVM virtual machines discussed in Chapters 1 and 2) running Linux can help you practice the lessons of this chapter.

## INSIDE THE EXAM

### Inside the Exam

As shown here, the RHCE objectives for NFS are essentially the same as for Samba. Of course, what you do to configure NFS is different.

- Provide network shares to specific clients
- Provide network shares suitable for group collaboration

The process for limiting NFS access to specific clients is straightforward. In addition, the way to set up group collaboration for an NFS network share is based on techniques that we have already discussed in Chapter 8.

The integration between NFS and Kerberos is a new requirement for the RHCE exam on RHEL 7. The objective is to

- Use Kerberos to control access to NFS network shares

In addition, you will configure firewalld and SELinux to work with NFS.

**CERTIFICATION OBJECTIVE 16.01**

## The Network File System (NFS) Server

NFS is the standard for sharing files with Linux and Unix computers. It was originally developed by Sun Microsystems in the mid-1980s. Linux has supported NFS (both as a client and a server) for years, and NFS continues to be popular in organizations with Unix- or Linux-based networks.

You can create NFS shares by editing the /etc/exports configuration file, or by creating a new file in the /etc/exports.d directory. In that way, you can set up NFS for basic operation. To set up more advanced configurations, it can be helpful to understand the way NFS works and how it communicates over a network.

You can enhance NFS security in a number of ways, including the following:

- A properly configured firewall
- TCP Wrappers
- SELinux
- Kerberos authentication and encryption

## NFS Options for RHEL 7

While NFS version 4 (NFSv4) is the default, RHEL 7 also supports NFS 3 (NFSv3). The differences between NFSv3 and NFSv4 include the way clients and servers communicate, the maximum file sizes, and support for Windows-style access control lists (ACLs).

If you use NFSv4, you do not need to set up Remote Procedure Call (RPC) communication with the **rpcbind** package. However, RPC is required for NFSv3.

NFSv3 introduced support for 64-bit file sizes to handle files larger than 2GB. NFSv4 extends NFSv3 and provides several performance improvements. It also supports better security, through integration with Kerberos. Whereas NFSv3 relies on a separate protocol for file locking known as "NLM" (the Network Lock Manager), NFSv4 includes file locking natively.

on the job

**NFS version 4.1 supports clustered deployments through the pNFS (parallel NFS) extension. pNFS allows NFS to scale by distributing data across multiple servers and by retrieving that data in parallel from clients. For more information, see the websites http://www.pnfs.org and https://github.com/nfs-ganesha/nfs-ganesha.**

## Basic NFS Installation

The primary group associated with NFS software is the "File and Storage Server" group. In other words, if you run the following command, **yum** installs the mandatory packages from that group:

```
# yum group install "File and Storage Server"
```

However, this group also includes packages for Samba, CIFS, and iSCSI target support. The only package required to set up an NFS server or client is **nfs-utils**:

```
# yum -y install nfs-utils
```

You may want to install additional packages, including the following:

- **nfs4-acl-tools** Provides command-line utilities to retrieve and edit access lists on NFS shares.
- **portreserve** Supports the **portreserve** service, the successor to **portmap** for NFS communication. Prevents NFS from taking ports needed by other services.
- **quota** Provides quota support for shared NFS directories.
- **rpcbind** Includes RPC communication support for different NFS channels.

## Basic NFS Server Configuration

NFS servers are relatively easy to configure. All you need to do is export a filesystem and then mount that filesystem from a remote client.

Of course, that assumes you have opened up the right ports in the firewall and modified appropriate SELinux options. NFS is controlled by a series of systemd service units. It also comes with a broad array of control commands.

### NFS Services

Once the appropriate packages are installed, they are controlled by several different systemd service units:

- **nfs-server.service** Service unit for the NFS server; refers to /etc/sysconfig/nfs for basic configuration.
- **nfs-secure-server.service** Starts the **rpc.svcgssd** daemon, which provides Kerberos authentication and encryption support for the NFS server.
- **nfs-secure.service** Starts the **rpc.gssd** daemon, which negotiates Kerberos authentication and encryption between an NFS client and server.
- **nfs-idmap.service** Runs the **rpc.idmapd** daemon, which translates user and group IDs into names. Automatically started by the **nfs-server** systemd unit.
- **nfs-lock.service** Required by NFSv3. Starts the **rpc.statd** daemon, which provides locks and the status for files currently in use.
- **nfs-mountd.service** Runs the **rpc.mountd** NFS mount daemon. Required by NFSv3.
- **nfs-rquotad.service** Starts the **rpc.rquotad** daemon, which provides filesystem quota services to NFS shares. Automatically started by the **nfs-server** systemd unit.
- **rpcbind.service** Executes the **rpcbind** daemon, which converts RPC program numbers into addresses. Used by NFSv3. Automatically started by the **nfs-server** systemd unit.

To bring up an NFS server, you don't have to memorize all the service units just listed. Given the default dependencies between service units, all you need to do is run the following commands on the NFS server machine:

```
# systemctl start nfs-server
# systemctl enable nfs-server
```

To enable Kerberos support for NFS, you also need to activate the **nfs-secure-server** and **nfs-secure** services on the server and client machines, respectively. This will be covered in more detail in the next sections.

## NFS Control Commands and Files

NFS includes a wide variety of commands to set up exports, to show what's available, to see what's mounted, to review statistics, and more. Except for specialized **mount** commands, these commands can be found in the /usr/sbin directory.

The NFS **mount** commands are **mount.nfs** and **umount.nfs**. There are also two symbolic links, **mount.nfs4** and **umount.nfs4**. Functionally, they work like regular **mount** and **umount** commands. As suggested by the extensions, they apply to filesystems shared via NFSv4 and other NFS versions. Like other **mount.*** commands, they have functional equivalents. For example, the **mount.nfs4** command is functionally equivalent to the **mount -t nfs4** command.

If you're mounting a share via the **mount.nfs** and **mount -t nfs** commands, NFS tries to mount the share using NFSv4 and fails back to NFSv3 if version 4 is not supported by the server.

The packages associated with NFS include a substantial number of commands in the /usr/sbin directory. The list of commands shown here are just the ones most commonly used to configure and test NFS:

- **exportfs** The **exportfs** command can be used to manage directories shared through and configured in the /etc/exports file.
- **nfsiostat** A statistics command for input/output rates based on an existing mount point. Uses information from the /proc/self/mountstats file.
- **nfsstat** A statistics command for client/server activity based on an existing mount point. Uses information from the /proc/self/mountstats file.
- **showmount** The command most closely associated with a display of shared NFS directories, locally and remotely.

You can use ACL-related commands from the nfs4-acl-tools RPM. You can run these commands against filesystems mounted locally with the **acl** option, as discussed in Chapter 6. The commands themselves are straightforward, as they set (**nfs4_setfacl**), edit (**nfs4_editfacl**), and list (**nfs4_getfacl**) the current ACLs of specified files. While these commands go beyond the basic operation of NFS, they are briefly discussed here and in Chapter 4.

Assume you have mounted a /home directory with the **acl** option. You've shared that directory via NFS. When you apply the **nfs4_getfacl** command on a file on that shared directory, you may see the following output:

```
A::OWNER@:rwatTcCy
A::GROUP@:tcy
A::EVERYONE@:tcy
```

The ACLs are set to either Allow (A) or Deny (D) access to the file owner (OWNER, GROUP, or EVERYONE). The permissions that follow are finer-grained than regular rwx permissions. For example, to represent Linux write permissions, ACLs enable both write (w) and append (a) permissions.

Perhaps the simplest way to modify these ACLs is with the **nfs4_setfacl -e *filename*** command, which allows you to edit current permissions in a text editor. As an example, to edit a file ACL on a share mounted via NFSv4 from a remote system, run the following command:

```
$ nfs4_setfacl -e /tmp/michael/filename.txt
```

This command opens the given NFSv4 ACLs in the default text editor for the user (normally **vi**). When we deleted the append permissions for the owner of the file and then saved the changes, this action actually removed both append and write permissions for the file. To review the result, run the **nfs4_getfacl** command again:

```
D::OWNER@:wa
A::OWNER@:rtTcCy
A::GROUP@:rwatcy
A::EVERYONE@:rtcy
```

If you try the **ls -l** command on the same file, you will note that the file owner no longer has write permissions.

## Configure NFS for Basic Operation

The NFS share configuration file, /etc/exports, is fairly simple. Once it's configured, you can export the directories configured in that file with the **exportfs -a** command.

Each line in /etc/exports lists the directory to be exported, the hosts to which it will be exported, and the options that apply to this export. While you can set multiple conditions, you can export a particular directory only once. Take the following examples from an /etc/exports file:

```
/pub    tester1.example.com(rw,sync) *(ro,sync)
/home   *.example.com(rw,async) 172.16.10.0/24(ro)
/tftp   nodisk.example.net(rw,no_root_squash,sync)
```

In this example, the /pub directory is exported to the tester1.example.com client with read/write permissions. It is also exported to all other clients with read-only permissions. The /home directory is exported with read/write permissions to all clients on the example .com network, and read-only to clients on the 172.16.10.0/24 subnet. Finally, the /tftp directory is exported with full read/write permissions (even for root users) to the nodisk .example.net computer.

While these options are fairly straightforward, the /etc/exports file is somewhat picky. A space at the end of a line could lead to a syntax error. A space between a hostname and the conditions in parentheses would open access to all hosts.

All of these options include the **sync** flag. This requires write operations to be committed to disk before returning the status to the client. Before NFSv4, many such options included the **insecure** flag, which allows access on ports above 1024. More options will be discussed in the following sections.

You can also split the NFS configuration in multiple files with a .exports extension, within the /etc/exports.d directory. For instance, you could take the three configuration lines in the previous /etc/exports file and move them into separate files named pub.exports, home.exports, and tftp.exports within the /etc/exports.d directory.

exam watch

**Be careful with the /etc/exports file. For example, an extra space after either comma in (ro,no_root_squash,sync) means that the specified directory won't get exported.**

## Wildcards and Globbing

In Linux network configuration files, you can specify a group of computers with the right wildcard, which in Linux is also known as *globbing.* What can be used as a wildcard depends on the configuration file. The NFS /etc/exports file uses "conventional" wildcards: for example, *.example.net specifies all computers within the example.net domain. In contrast, in the /etc/hosts.deny file, .example.net, with the leading dot, specifies all computers in that same domain.

For IPv4 networks, wildcards often specify an implicit subnet mask. For example, 192.168.0.* is equivalent to 192.168.0.0/255.255.255.0, which specifies the 192.168.0.0 network of computers with IP addresses that range from 192.168.0.1 to 192.168.0.254. Some services, including NFS, support the use of CIDR (Classless Inter-Domain Routing) notation. In CIDR, since 255.255.255.0 masks 24 bits, CIDR represents this with the number *24.* When configuring a network in CIDR notation, you can represent this network as 192.168.0.0/24.

## More NFS Server Options

With /etc/exports, it's possible to use a number of different parameters. The parameters described in Tables 16-1 and 16-2 fall into two categories: general and security options.

**TABLE 16-1** NFS /etc/exports General Options

| Parameter | Description |
|---|---|
| async | Write operations are performed asynchronously. Provides better throughput, at the risk of losing data if the NFS server crashes. |
| hide | Hides filesystems; if you export a directory and subdirectory such as /mnt and /mnt/inst, shares to /mnt/inst must be explicitly mounted. |
| mp | Exports a directory only if it was successfully mounted; requires the export point to also be a mount point on the server. |
| ro | Exports a volume read-only. |
| rw | Exports a volume read-write. |
| sync | Commits write operations to disk before replying to the client. Active by default. |

Other parameters relate to security settings of NFS shared directories. As shown in Table 16-2, the options are associated with the root administrative user, anonymous-only users, and Kerberos authentication.

### Activate the List of Exports

After you configure the /etc/exports file, make those directories available to clients with the **exportfs -a** command. The next time RHEL 7 is booted, if the right services are activated, the nfs-server systemd unit runs the **exportfs -r** command, which re-exports directories configured in /etc/exports.

However, if you're modifying, moving, or deleting NFS shares, you should temporarily un-export all directories first with the **exportfs -ua** command. You can make desired

**TABLE 16-2** NFS /etc/exports Security Options

| Parameter | Description |
|---|---|
| all_squash | Maps all local and remote accounts to the anonymous user. |
| anongid=*groupid* | Specifies a group ID for the anonymous user account. |
| anonuid=*userid* | Specifies a user ID for the anonymous user account. |
| insecure | Supports communications above port 1024, primarily for NFS versions 2 and 3. |
| no_root_squash | Treats the remote root user as local root; if this parameter is not set, by default the root user will be mapped to the nfsnobody user. |
| sec=*value* | Specifies a list of colon-separated security options. The default value is **sys**, which instructs the NFS server to rely on UIDs/GIDs for file access. Kerberos-related values are **krb5**, **krb5i**, and **krb5p**. |

changes and then export the shares with the **exportfs -a** or **exportfs -r** command. The difference between **-a** and **-r** is subtle but important: whereas **-a** exports (or un-exports, in combination with **-u**) all directories, **-r** re-exports all directories by synchronizing the list of shares and removing those that have been deleted from the /etc/exports configuration file.

Once exports are active, you can review their status with the **showmount -e *servername*** command. For example, the **showmount -e server1.example.com** command looks for the list of exported NFS directories from the server1.example.com system. If this command is not successful, communication may be blocked by a firewall.

## Fixed Ports in /etc/sysconfig/nfs

NFSv4 is easier to configure, especially with respect to firewalls. To enable communication with an NFSv4 server, the only port you need to open is TCP port 2049. This port is part of the nfs service in firewalld, so you should run the following commands on an NFS server to allow inbound connections:

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

While NFSv4 is the default, RHEL 7 also supports NFSv3. So given the publicly available information on the RHCE exam, you might also need to know how to handle this version of NFS. NFSv3 uses dynamic port numbers through the RPC service, which listens on UDP port 111, and is associated to the rpc-bind firewalld service. You also need to grant access to the mountd service, so in total you need to allow the following services to support NFSv3 through firewalld:

```
# firewall-cmd --permanent --add-service=nfs --add-service=rpc-bind \
> --add-service=mountd
# firewall-cmd --reload
```

Once the NFS service is started with the **systemctl start nfs-server** command, if successful you'll see the associated ports in the output to the **rpcinfo** command, which lists all communication channels associated with RPC. The following command is more precise because it isolates actual port numbers:

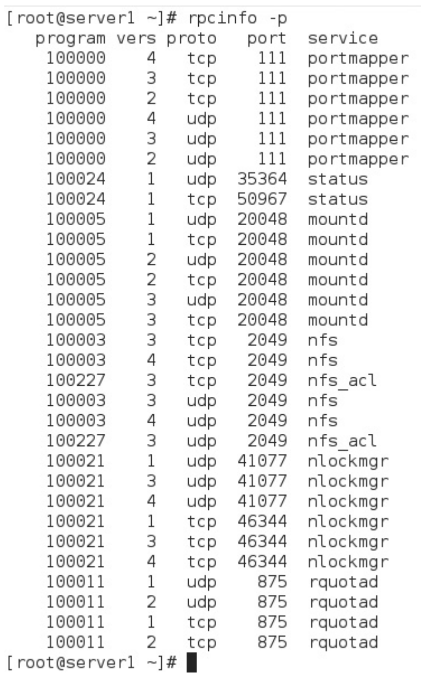
```
# rpcinfo -p
```

Sample output is shown in Figure 16-1. At first glance, the lines may appear repetitive; however, every line has a purpose. Unless another RPC-related service such as the Network Information Service (NIS) is running, all of the lines shown here are required for NFS communications. Examine the first line shown here:

```
program vers proto   port  service
 100000    4   tcp    111  portmapper
```

FIGURE 16-1

Sample rpcinfo -p output with NFS-related ports



```
[root@server1 ~]# rpcinfo -p
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  35364  status
    100024    1   tcp  50967  status
    100005    1   udp  20048  mountd
    100005    1   tcp  20048  mountd
    100005    2   udp  20048  mountd
    100005    2   tcp  20048  mountd
    100005    3   udp  20048  mountd
    100005    3   tcp  20048  mountd
    100003    3   tcp   2049  nfs
    100003    4   tcp   2049  nfs
    100227    3   tcp   2049  nfs_acl
    100003    3   udp   2049  nfs
    100003    4   udp   2049  nfs
    100227    3   udp   2049  nfs_acl
    100021    1   udp  41077  nlockmgr
    100021    3   udp  41077  nlockmgr
    100021    4   udp  41077  nlockmgr
    100021    1   tcp  46344  nlockmgr
    100021    3   tcp  46344  nlockmgr
    100021    4   tcp  46344  nlockmgr
    100011    1   udp    875  rquotad
    100011    2   udp    875  rquotad
    100011    1   tcp    875  rquotad
    100011    2   tcp    875  rquotad
[root@server1 ~]# █
```

The first line represents the arbitrary RPC program number, the NFS version, and the use of TCP as a communications protocol, over port 111, with the portmapper service. Note the availability of the portmapper service to NFS versions 2, 3, and 4, communicating over the TCP and UDP protocols.

Communication through selected ports should also be allowed through any configured firewall. For example, Figure 16-2 shows that the firewalld configuration supports remote access to a local NFS server through protocol versions 3 and 4.

You can set up these firewall rules with the graphical **firewall-config** tool discussed in Chapter 4.

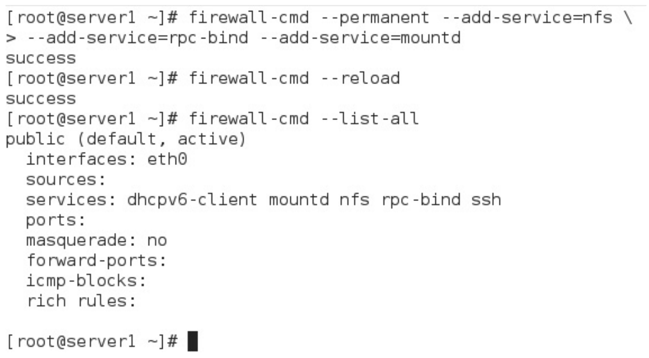
## Make NFS Work with SELinux

Of course, you need to configure more than a firewall. SELinux is an integral part of the security landscape, with respect to both boolean options and files. First, be aware of the following NFS SELinux file types:

- **nfs_t** Associated with NFS shares that are exported read-only or read-write.
- **public_content_ro_t** Associated with NFS shares that are exported read-only.

FIGURE 16-2

Firewall rules for NFS



```
[root@server1 ~]# firewall-cmd --permanent --add-service=nfs \
> --add-service=rpc-bind --add-service=mountd
success
[root@server1 ~]# firewall-cmd --reload
success
[root@server1 ~]# firewall-cmd --list-all
public (default, active)
  interfaces: eth0
  sources:
  services: dhcpv6-client mountd nfs rpc-bind ssh
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:

[root@server1 ~]# 
```

- **public_content_rw_t** Associated with NFS shares that are exported read-write. Requires the **nfsd_anon_write** boolean to be set.
- **var_lib_nfs_t** Associated with dynamic files in the /var/lib/nfs directory. Files in this directory are updated as shares are exported and mounted by clients.
- **nfsd_exec_t** Assigned to system executable files such as **rpc.mountd** and **rpc.nfsd** in the /usr/sbin directory. Closely related are the rpcd_exec_t and gssd_exec_t file types for services associated with RPCs and communications with Kerberos servers.

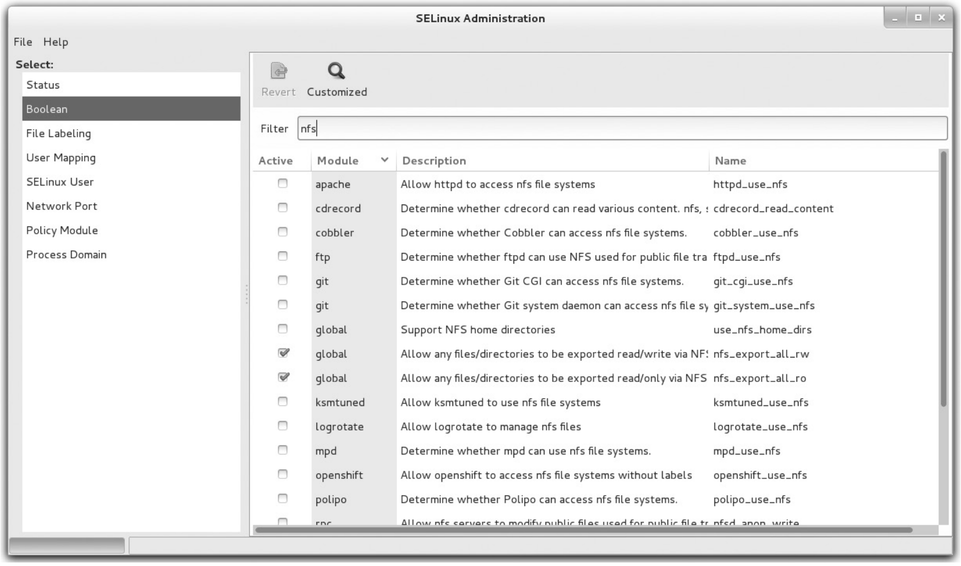
In general, you won't have to assign a new file type to a shared NFS directory. In fact, the SELinux file types that are related to file shares (nfs_t, public_content_ro_t, and public_content_rw_t) are effective only when the nfs_exports_all_ro and nfs_exports_all_rw booleans are disabled. So for most administrators, these file types are shown for reference.

For SELinux, the boolean directives are most important. The options are shown in the Booleans section of the SELinux Administration tool, with the nfs filter, as shown in Figure 16-3. The figure reflects the default configuration; as you can see, two of the options in the global module are enabled by default.

The following directives are associated with making NFS work with SELinux in targeted mode. While most of these options were already listed in Chapter 10, they're worth repeating, if only to help those who fear SELinux. The options are described in the order shown in the figure.

- **httpd_use_nfs** Supports access by the Apache web server to shared NFS shares.
- **cdrecord_read_content** Enables access to mounted NFS shares by the **cdrecord** command.
- **cobbler_use_nfs** Allows Cobbler to access NFS filesystems.
- **ftpd_use_nfs** Allows the use of shared NFS directories by FTP servers.

FIGURE 16-3 NFS-related SELinux boolean options



Filter nfs

| Active | Module | Description | Name |
|---|---|---|---|
| ☐ | apache | Allow httpd to access nfs file systems | httpd_use_nfs |
| ☐ | cdrecord | Determine whether cdrecord can read various content. nfs, s | cdrecord_read_content |
| ☐ | cobbler | Determine whether Cobbler can access nfs file systems. | cobbler_use_nfs |
| ☐ | ftp | Determine whether ftpd can use NFS used for public file tra | ftpd_use_nfs |
| ☐ | git | Determine whether Git CGI can access nfs file systems. | git_cgi_use_nfs |
| ☐ | git | Determine whether Git system daemon can access nfs file sy | git_system_use_nfs |
| ☐ | global | Support NFS home directories | use_nfs_home_dirs |
| ✓ | global | Allow any files/directories to be exported read/write via NFS | nfs_export_all_rw |
| ✓ | global | Allow any files/directories to be exported read/only via NFS | nfs_export_all_ro |
| ☐ | ksmtuned | Allow ksmtuned to use nfs file systems | ksmtuned_use_nfs |
| ☐ | logrotate | Allow logrotate to manage nfs files | logrotate_use_nfs |
| ☐ | mpd | Determine whether mpd can use nfs file systems. | mpd_use_nfs |
| ☐ | openshift | Allow openshift to access nfs file systems without labels | openshift_use_nfs |
| ☐ | polipo | Determine whether Polipo can access nfs file systems. | polipo_use_nfs |
| ☐ | rpc | Allow nfs servers to modify public files used for public file tr | nfsd_anon_write |

- **git_cgi_use_nfs** Supports access to NFS shares by the git revision control system service in CGI scripts.
- **git_system_use_nfs** Supports access to NFS shares by the git revision control system service.
- **use_nfs_home_dirs** Enables the mounting of /home from a remote NFS server.
- **nfs_export_all_rw** Supports read-write access to shared NFS directories.
- **nfs_export_all_ro** Supports read-only access to shared NFS directories.
- **ksmtuned_use_nfs** Allows ksmtuned to access NFS shares.
- **logrotate_use_nfs** Allows logrotate to access NFS files.
- **mpd_use_nfs** Allows the Music Player Daemon to access content from NFS shares.
- **openshift_use_nfs** Allows OpenShift to access NFS filesystems.
- **polipo_use_nfs** Allows access by the Polipo web proxy to NFS-mounted filesystems.
- **nfsd_anon_write** Allows NFS servers to modify public files. Files must labeled with the public_content_rw_t type.
- **samba_share_nfs** Allows Samba to export NFS-mounted filesystems.
- **sanlock_use_nfs** Enables the SANlock lock manager daemon to access NFS files.

- **sge_use_nfs** Allows the Sun Grid Engine to access NFS files.
- **virt_use_nfs** Enables access by VMs to NFS-mounted filesystems.
- **virt_sandbox_use_nfs** Allows sandbox containers to access NFS filesystems.
- **xen_use_nfs** Allows access by the Xen hypervisor to NFS-mounted filesystems.

To set these directives, use the **setsebool** command. For example, to activate access to NFS filesystems by an FTP server, in a way that survives a reboot, run the following command:

```
# setsebool -P ftpd_use_nfs 1
```

## Quirks and Limitations of NFS

NFS does have its limitations. Any administrator who controls shared NFS directories would be wise to take note of these limitations.

### Statelessness

NFSv3 is a "stateless" protocol. In other words, you don't need to log in separately to access a shared NFS directory. Instead, the NFS client normally contacts **rpc.mountd** on the server. The **rpc.mountd** daemon handles mount requests. It checks the request against currently exported filesystems. If the request is valid, **rpc.mountd** provides an NFS *file handle* (a "magic cookie"), which is then used for further client/server communication for this share.

The stateless protocol allows the NFS client to wait if the NFS server ever has to be rebooted. The software waits, and waits, and waits. This can cause the NFS client to hang. The client may even have to reboot or even power-cycle the system.

This can also lead to problems with insecure single-user clients. When a file is opened through a share, it may be "locked out" from other users. When an NFS server is rebooted, handling the locked file can be difficult.

The changes that led to the development of NFSv4 introduced a stateful protocol to make the locking mechanism more robust, and should help address this problem.

### Root Squash

By default, NFS is set up to **root_squash**, which prevents root users on an NFS client from gaining root access to a share on an NFS server. Specifically, the root user on a client (with a user ID of 0) is mapped to the *nfsnobody* unprivileged account (if in doubt, check the local /etc/passwd file).

This behavior can be disabled via the **no_root_squash** server export option in /etc/exports. For exported directories with the **no_root_squash** option, remote root users can use their root privileges on the shared NFS directory. While it can be useful, it is also a security risk, especially from "black hat" hackers who use their own Linux systems to take advantage of those root privileges.

## NFS Hangs

Because NFSv3 is stateless, NFS clients may wait up to several minutes for a server. In some cases, an NFS client may wait indefinitely if a server goes down. During the wait, any process that looks for a file on the mounted NFS share will hang. Once this happens, it is generally difficult to unmount the offending filesystems, unless you pass the "lazy" option to the **umount** command (**umount -l**). This may still leave some processes in an uninterruptible sleep state, waiting for I/O. You can do several things to reduce the impact of this problem:

- Take great care to ensure the reliability of NFS servers and the network.
- Mount infrequently used NFS exports only when needed. NFS clients should unmount these shares after use.
- Don't use **async**, and set up NFS shares with the **sync** option (the default), which should at least reduce the incidence of lost data.
- Keep NFS-mounted directories out of the search path for users, especially that of root.
- Keep NFS-mounted directories out of the root (/) directory; instead, segregate them to a less frequently used filesystem, if possible, on a separate partition.

## Inverse DNS Pointers

An NFS server daemon checks mount requests. First, it looks at the current list of exports, based on /etc/exports. Then it looks up the client's IP address to find its hostname. This requires a reverse DNS lookup.

This hostname is then finally checked against the list of exports. If NFS can't find a hostname, **rpc.mountd** will deny access to that client. For security reasons, it also adds a "request from unknown host" entry in /var/log/messages.

## File Locking

Multiple NFS clients can be set up to mount the same exported directory from the same server. It's quite possible that people on different computers end up trying to use the same shared file. This is addressed by the file-locking daemon service.

While mandatory locks are supported by NFSv4, NFS has historically had serious problems with file locks. If you have an application that depends on file locking over NFS, test it thoroughly before putting it into production.

In addition, you should never share the same directory with NFS and Samba simultaneously because the different locking mechanisms used by these services can cause data corruption.

## Performance Tips

You take several steps to keep NFS running in a stable and reliable manner. As you gain experience with NFS, you might monitor or even experiment with the following factors:

- Eight NFS processes, which is the default, is generally sufficient for good performance, even under fairly heavy loads. To increase the capacity of the service, you can add more NFS processes through the **RPCNFSDCOUNT** directive in the /etc/sysconfig/nfs configuration file. Just keep in mind that the extra processes consume additional system resources.
- NFS write performance can be slow. In applications where data loss is not a big concern, you may try the **async** option. This makes NFS faster because the server immediately returns the state of a write operation to the client, without waiting for the data to be written to disk. However, a loss of power or network connectivity can result in a loss of data.
- Hostname lookups are performed frequently by the NFS server; you can start the Name Switch Cache Daemon (**nscd**) to speed lookup performance.

## NFS Security Directives

NFS includes a number of potential security problems and should never be used in hostile environments (such as on a server directly exposed to the Internet), at least not without strong precautions.

### Shortcomings and Risks

NFS is an easy-to-use yet powerful file-sharing system. However, it is not without its limitations. The following are a few security issues to keep in mind:

- **Authentication** NFS relies on the host to report user and group IDs. However, this can be a security risk if root users on other computers access your NFS shares. In other words, data that is accessible via NFS to *any user* can potentially be accessed by *any other* user. This risk is addressed by NFSv4 if Kerberos is used for authentication.
- **Privacy** Before NFSv4, NFS did not support encryption. NFSv4 with the support of Kerberos can provide encrypted communications.
- **rpcbind infrastructure** Both the NFSv3 client and server depend on the RPC portmap daemon. The earlier versions of the daemon had historically a number of serious security holes. For this reason, RHEL 7 has replaced it with the rpcbind service.

### Security Tips

If NFS *must* be used in or near a hostile environment, you can reduce the security risks:

- Educate yourself in detail about NFS security. If possible, set up encrypted NFSv4 communications with the help of Kerberos. Otherwise, restrict NFS to friendly, internal networks protected with a good firewall.
- Export as little data as possible, and export filesystems as read-only if possible.
- Unless absolutely necessary, don't supersede the **root_squash** option. Otherwise, "black hat" hackers on allowed clients may grant root-level access to exported filesystems.
- Use appropriate firewall settings to deny access to the portmapper and nfsd ports, except from explicitly trusted hosts or networks. If you're using NFSv4, it's good enough to open only the following port via the nfs firewalld service:

```
2049    TCP          nfsd           (server)
```

## Options for Host-Based Security

To review, host-based security on NFS systems is based primarily on the systems allowed to access a share in the /etc/exports file. Of course, host-based security can also include limits based on firewall rules.

**exam watch**

**As long as there's a common user database, such as LDAP, the permissions associated with a common group directory carry over to a mount shared via NFS.**

## Options for User-Based Security

As NFS mounts should reflect the security associated with a common user database, the standard user-based security options should apply. That includes the configuration of a common group, as discussed in Chapter 8.

**EXERCISE 16-1**

### NFS

This exercise requires two systems: one set up as an NFS server, the other as an NFS client. Then, on the NFS server, take the following steps:

1. Set up a group named IT for the Information Technology group in /etc/group.
2. Create the /MIS directory. Assign ownership to the MIS group with the **chgrp** command.

3. Set the SGID bit on this directory to enforce group ownership.
4. Ensure that the **nfs-utils** RPM package is installed.
5. On the server, start and enable the NFS service to run at boot:

```
# systemctl start nfs-server
# systemctl enable nfs-server
```

6. Update the /etc/exports file to allow read and write permissions to the share for the local network. Run the following command to apply the change:

```
# exportfs -a
```

7. Make sure the SELinux booleans are set appropriately; specifically, make sure the nfs_export_all_ro and nfs_export_all_rw booleans are both enabled. This is the default setting. You can do so either with the **getsebool** command or the SELinux Management tool.
8. Open the required ports on the firewall. For NFSv4, the following commands are required:

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

   Then, on an NFS client, take the following steps:

9. Ensure that the **nfs-utils** RPM package is installed.
10. Create a directory for the server share called /mnt/MIS.
11. Mount the shared NFS directory on /mnt/MIS.
12. List all exported shares from the server and save this output in the shares.list file in the /mnt/MIS directory.
13. Make this a permanent mount in the /etc/fstab file. Assume that the connection might be troublesome, and add the appropriate options, such as **soft** mounting.
14. Run the **mount -a** command to reread /etc/fstab. Check to see if the share is properly remounted.
15. Test the NFS connection. Stop the NFS service on the server and then try copying a file to the /mnt/MIS directory. While the attempt to copy will fail, it should not hang the client.
16. Restart the NFS service on the server.
17. Edit /etc/fstab again. This time, assume that NFS is reliable and remove the special options added in Step 13.
18. Now shut down the server and test what happens. The mounted NFS directory on the client should hang when you try to access the service.
19. The client computer may lock. If so, you can boot into the rescue target, as described in Chapter 5, to avoid the pain of a reboot. Restore the original configuration.

**CERTIFICATION OBJECTIVE 16.02**

## Test an NFS Client

Now you can mount a shared NFS directory from a client computer. The commands and configuration files are similar to those used for any local filesystem. In the preceding section, you configured an NFS server. For now, stay on the NFS server system, as the first client test can be run directly from this machine.

### NFS Mount Options

Before doing anything elaborate, you should check for the list of shared NFS directories. Then you can mount a shared NFS volume from a second Linux system, presumably a RHEL 7 system (or equivalent). To that end, the **showmount** command displays the available shared volumes.

Run the **showmount** command with the **-e** option; when coupled with the hostname or IP address of the NFS server, the command displays the export list, possibly including the host limits of the share. For example, given a simple share of the /mnt and /home directories on a given NFS server, the **showmount -e server1.example.com** command provides the following result:

```
Export list for server1.example.com:
/mnt  192.168.100.0/24
/home 192.168.122.0/24
```

If you don't see a list of shared directories, log in to the NFS server system. Repeat the **showmount** command, substituting localhost or 127.0.0.1 for the hostname or IP address. If there's still no output, review the steps described earlier in this chapter. Make sure the /etc/exports file is configured properly. Remember to export the shared directories. Use the command

```
# systemctl status nfs-server
```

to confirm that the NFS services are running.

Now to mount this directory locally, you'll need an empty local directory. Create a directory such as /remotemnt. You can then mount the shared directory from a system such as 192.168.122.50 with the following command:

```
# mount.nfs 192.168.122.50:/share /remotemnt
```

This command mounts the NFS /share directory from the computer on the noted IP address. If desired, you could substitute the **mount -t nfs** command for **mount.nfs**. When it works, you'll be able to access files from the remote /share directory as if it were a local directory. If the local mount works but the remote mount does not, check the firewall settings and ensure that the service is running.

## Configure NFS in /etc/fstab

You can also configure an NFS client to mount a remote NFS directory during the boot process, as defined in /etc/fstab. For example, the following entry in a client /etc/fstab mounts the /homenfs share from the computer named nfsserv on the local /nfs/home directory, using the default version 4 of the protocol:

```
nfsserv:/homenfs   /nfs/home  nfs  soft,timeo=100  0  0
```

The **soft** and **timeo** options are two specialized NFS mount options. Such options, as shown here, can also be used to customize how mounts are done during the boot process in the /etc/fstab file.

Consider using the **soft** option when mounting NFS filesystems. When an NFS server fails, a soft-mounted NFS filesystem will fail rather than hang. However, this can cause data corruption in case of a temporary network outage. Use this option only when the responsiveness of a client is more important than data integrity. In addition, you can use the **timeo** option to set a timeout interval, in tenths of a second.

**exam watch** **NFS-specific options to the mount command that can also be used in /etc/fstab can be found in the nfs man page.**

For more information on these and related options, see the nfs man page, available with the **man nfs** command.

Alternatively, an automounter can be used to mount NFS filesystems dynamically as required by the client computer. The automounter can also unmount these remote filesystems after a period of inactivity. For more information on the governing autofs service, see Chapter 6.

## Diskless Clients

NFS supports diskless clients, which are computers that do not store the operating system locally. A diskless client may use a flash memory chip to get started. Then embedded commands can mount the appropriate root (/) directory, set up swap space, set the /usr directory as read-only, and configure other shared directories such as /home in read/write mode. If your computer is set up as a diskless client, you'll also need access to DHCP and TFTP servers to boot the system from a network boot server.

Red Hat Enterprise Linux includes features that support diskless clients. While they are not listed as part of the current Red Hat exam requirements or related course outlines, we would not be surprised to see such requirements in the future.

## Current NFS Status

The current status of NFS services is documented in two directories: /var/lib/nfs and /proc/fs/nfsd. If there's a problem with NFS, look at some of the files in these directories. Take these directories one at a time. First, there are two key files in the /var/lib/nfs directory:

- **etab** Includes a full description of exported directories, including default options
- **rmtab** Specifies the state of shared directories currently mounted

Take a look at the contents of the /proc/fs/nfsd directory. As this is a virtual directory, files in the /proc directory tree have a size of zero. However, as dynamic files, they can contain valuable information. Perhaps the key option for basic operation is the file /proc/fs/nfsd/versions. The content of that file specifies the currently recognized versions of NFS.

The normal content of this file is just a little cryptic, which suggests that the current NFS server can communicate using NFSv3, NFSv4, and NFSv4.1, but not with NFSv4.2 and NFSv2:

```
-2 +3 +4 +4.1 -4.2
```

If you set the **RPCNFSDARGS="-V 4.2"** option in the /etc/sysconfig/nfs file and restart the NFS service, the contents of the versions file will change to

```
-2 +3 +4 +4.1 +4.2
```

The difference is subtle but important. In fact, NFSv4.2 provides an experimental feature that allows you to keep the original SELinux context of each file in the shared directory. You may want to switch to NFSv4.2 if you need this feature.

## CERTIFICATION OBJECTIVE 16.03

## NFS with Kerberos

For several years, NFS was considered an insecure protocol. One reason is that NFS, by default, trusts the UID and GID sent by a client. A "black hat" hacker that has access to an NFS share can easily impersonate the identity of another user and pass her UID/GID credentials, because NFS is based on trust.

NFSv4 security issues have been addressed with Kerberos, which can provide strong authentication, integrity, and encryption services. If you need security with NFS, protect NFS exports with Kerberos.

This section is focused on the configuration of NFS with a Kerberos server. It assumes that you have set up a Kerberos KDC and that clients have joined the Kerberos realm, as described in Chapter 12.

## Kerberos-Enabled NFS Services

To set up a simple NFS service as you did in Exercise 16-1, you need to activate the nfs-server systemd unit on the NFS server host. If you want to integrate NFS with Kerberos, you need to enable two additional services, nfs-secure-server and nfs-secure, as illustrated in Tables 16-3 and 16-4.

Therefore, the simplest way to set up all the required services on an NFS server is with the following commands:

```
# systemctl start nfs-server
# systemctl start nfs-secure-server
# systemctl enable nfs-server
# systemctl enable nfs-secure-server
```

It's also important to enable the following service unit on all NFS clients:

```
# systemctl start nfs-secure
# systemctl enable nfs-secure
```

As noted in Chapter 11, these commands start the noted service units and make sure the services start the next time the system is rebooted.

**TABLE 16-3** The systemd Service Units on a Kerberos-Enabled NFS Server

| systemd Service Unit | Description |
|---|---|
| nfs-server | The main service unit for the NFS server. It activates other service units, such as nfs-idmap, nfs-rquotad, and rpcbind.service. It uses /etc/sysconfig/nfs for basic configuration. |
| nfs-secure-server | Provides Kerberos-based authentication and encryption for an NFS server through the **rpc.svcgssd** daemon. |

**TABLE 16-4** The systemd Service Units on a Kerberos-Enabled NFS Client

| systemd Service Unit | Description |
|---|---|
| nfs-secure | Provides Kerberos authentication and encryption services to an NFS client via the **rpc.gssd** daemon |

## Configure NFS Exports with Kerberos

The configuration of a Kerberos-enabled NFS export is straightforward and is based on the **sec** security option of /etc/exports, which we have already encountered in Table 16-2.

The **sec** option is followed by a colon-separated list of security flavors that an NFS server provides to its client. As an example, examine the following line from an /etc/exports file:

```
/nfs-share *.example.com(rw,sec=sys:krb5:krb5p)
```

This configuration exports the directory /nfs-share via NFS to clients in the example.com domain, with read-write access. Clients can mount the NFS share using one of the following security options: sys, krb5, or krb5p.

These options are illustrated in Table 16-5. From the information in the table, the most secure export method is **krb5p** because it provides Kerberos authentication, data integrity, and encryption. However, this comes at a cost, as data encryption requires CPU resources and may significantly affect performance.

The **krb5** and **krb5i** security options provide authentication and integrity services, and are a good compromise between security and throughput. Finally, the **sys** security method corresponds to the UID/GID trust model of NFS, which is always assumed as the default security method when the **sec** security option is not specified in /etc/exports.

If you want to force NFS clients to mount an NFS share using a specific security option, include that option as part of the **sec** parameter. For example, the following line in /etc/exports ensures that clients in the example.com domain mount the nfs-share directory with Kerberos authentication, integrity, and encryption:

```
/nfs-share *.example.com(rw,sec=krb5p)
```

Do remember to run **exportfs -r** on the NFS server to apply the change and refresh the list of exported directories.

**TABLE 16-5** NFS Security Options

| Security Option | Description |
|---|---|
| sys | Trusts the UID/GID provided by clients to determine file access permission. Enabled by default when no **sec**= option is specified. |
| krb5 | Verifies the UID/GID provided by clients using Kerberos authentication. |
| krb5i | Has the same effect as the **krb5** option, but in addition provides strong communication integrity. |
| krb5p | Has the same effect as the **krb5i** option, but in addition provides encryption services. |

## Configure NFS Clients with Kerberos

NFS clients can easily mount an NFS share with Kerberos authentication, integrity, and encryption services using the **sec** option with the values listed in Table 16-5. To do so, include the **sec** option either with the **mount** command or in the /etc/fstab file.

For example, the following command mounts the nfs-share directory from the host 192.168.122.50 using Kerberos authentication:

```
mount -t nfs -o sec=krb5 192.168.122.50:/nfs-share /mnt
```

Similarly, the following line in /etc/fstab instructs the system to mount the nfs-share directory during the boot process using Kerberos authentication, encryption, and strong integrity:

```
192.168.122.50:/nfs-share   /mnt  nfs  soft,sec=krb5p 0  0
```

### EXERCISE 16-2

### Prepare a System for NFS Secured with Kerberos

To prepare a system to export shared directories via NFS secured with Kerberos, you need to complete a few configuration steps. We assume that you have installed a Kerberos KDC and configured server1.example.com for Kerberos authentication, as illustrated in Exercise 12-5.

Then on the KDC, take the following steps:

1. Create host principals for the NFS server (server1.example.com) and all clients (such as tester1.example.com):

```
# kadmin.local
Authenticating as principal root/admin@WAMPLE.COM with password
kadmin.local:  addprinc -randkey host/server1.example.com
WARNING: no policy specified for host/server1.example.com@EXAMPLE.COM;
defaulting to no policy
Principal "host/server1.example.com@EXAMPLE.COM" created.
kadmin.local: addprinc -randkey host/tester1.example.com
WARNING: no policy specified for host/tester1.example.com@EXAMPLE.COM;
defaulting to no policy
Principal "host/tester1.example.com@EXAMPLE.COM" created.
kadmin.local:
```

2. Add NFS service principals for the server and client machines:

```
kadmin.local:  addprinc -randkey nfs/server1.example.com
WARNING: no policy specified for nfs/server1.example.com@EXAMPLE.COM;
defaulting to no policy
Principal "nfs/server1.example.com@EXAMPLE.COM" created.
kadmin.local: addprinc -randkey nfs/tester1.example.com
WARNING: no policy specified for nfs/tester1.example.com@EXAMPLE.COM;
defaulting to no policy
Principal "nfs/tester1.example.com@EXAMPLE.COM" created.
kadmin.local:
```

3. Generate the keytab files for the NFS server and client machines:

```
# kadmin.local: ktadd -k /tmp/server1.keytab nfs/server1.example.com
[output truncated]
# kadmin.local: ktadd -k /tmp/tester1.keytab nfs/server1.example.com
[output truncated]
```

4. Copy the keytab files to the /etc/krb5.keytab file on the remote systems:

```
# scp /tmp/server1.keytab server1.example.com:/etc/krb5.keytab
# scp /tmp/tester1.keytab tester1.example.com:/etc/krb5.keytab
```

5. Copy the /etc/krb5.conf file from the KDC to all NFS servers and clients:

```
# scp /etc/krb5.conf server1.example.com:/etc/krb5.conf
# scp /etc/krb5.conf tester1.example.com:/etc/krb5.conf
```

## EXERCISE 16-3

### Configure a Kerberos-Enabled NFS Share

In this exercise, you'll install an NFS server on a RHEL system and export a share with Kerberos authentication and encryption. This exercise assumes that you've set up a Kerberos Key Distribution Center and configured your server1.example.com and tester1.example.com virtual machines as described in Exercises 12-5 and 16-2.

1. Make sure the NFS server is installed on server1.example.com. The easiest way is with the following command:

```
# rpm -q nfs-utils
```

2. If it isn't already installed, use the techniques discussed earlier to install the nfs-utils RPM package.

3. Start the NFS service and its secure component to provide Kerberos authentication and encryption services:

```
# systemctl start nfs-server nfs-secure-server
```

4. Make sure the services are automatically activated the next time the system boots with the following command:

```
# systemctl enable nfs-server nfs-secure-server
```

5. Create a directory named nfs-secure:

```
# mkdir /nfs-secure
```

6. Configure the share in the /etc/exports file to allow read and write permissions to all clients with Kerberos authentication and encryption:

```
# echo "/nfs-secure *(rw,sec=krb5p)" >> /etc/exports
```

7. Apply the change:

```
# exportfs -r
```

8. Ensure that the nfs service is enabled on the firewall default zone:

```
# firewall-cmd --list-all
```

9. If it isn't enabled, add the service to the default zone:

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

10. On the tester1.example.com client, ensure that the **nfs-utils** RPM package is installed.
11. Start the nfs-secure service and activate the service at boot:

```
# systemctl start nfs-secure
# systemctl enable nfs-secure
```

12. Create a directory for the server share called /mnt/nfs:

```
# mkdir /mnt/nfs
```

13. Add the following line to /etc/fstab:

```
192.168.122.50:/nfs-secure   /mnt/nfs  nfs  sec=krb5p 0  0
```

14. Run the **mount -a** command to mount the share.

## SCENARIO & SOLUTION

| | |
|---|---|
| You're having trouble configuring a firewall for NFS. | Enable the nfs service by running **firewall-cmd --add-service=nfs**. |
| You want to prohibit read/write access to shared NFS directories. | Make sure shares are configured with the **ro** parameter in /etc/exports. |
| You need to set up automatic mounts of a shared NFS directory. | Configure the shared directory in /etc/fstab. |
| You want to export an NFS share with Kerberos authentication and encryption. | Export and mount the share with the **sec=krb5p** option. Ensure that your systems are set up for Kerberos authentication, as described in Appendix A. |
| You need to start NFS services to export an NFS share with Kerberos authentication. | Enable the services **nfs-server** and **nfs-secure-server** on the NFS server, and **nfs-secure** on the clients. |

## CERTIFICATION SUMMARY

NFS allows you to share filesystems between Linux and Unix computers. It is an efficient way to share files between such systems, and it can be secured with Kerberos authentication and encryption.

While RHEL 7 supports NFSv4, it also supports access by NFSv3 clients. It's controlled by a group of systemd units. The service unit **nfs-server** is required to start the NFS daemon. Kerberos-based authentication and encryption are controlled by the **rpcsvcgssd** and **rpcgssd** daemons, which depend, respectively, on the **nfs-secure-server** service unit (on the server) and **nfs-secure** (on the client). The global options for the NFS service are set up primarily in the /etc/sysconfig/nfs file. Related commands include **exportfs** and **showmount**.

In most cases, you can set up a basic configuration of NFS via a straightforward one-line directive in the /etc/exports file. Once the NFS service is running, such exports are activated through the **exportfs** command. Firewalls should be configured by enabling the nfs service through the appropriate zone. Active ports and services can be confirmed with the **rpcinfo -p** command.

Generally, the default configuration of SELinux supports basic NFS operation. You can configure security for mounted NFS directories as if the mounted filesystems were local. You can also automate NFS mounts in /etc/fstab or through the automounter. The current status of NFS is documented in various files in the /var/lib/nfs and /proc/fs/nfsd directories.

## ✓ TWO-MINUTE DRILL

Here are some of the key points from the certification objectives in Chapter 16.

### The Network File System (NFS) Server

- ❑ NFS is the standard for sharing files between Linux and Unix computers. RHEL 7 supports NFS versions 3 and 4; NFSv4 is the default.
- ❑ Key NFS daemons are **rpc.mountd** for mount requests, **rpc.rquotad** for quota requests, and the **nfsd** daemon.
- ❑ You can find configuration options for these processes in the /etc/sysconfig/nfs file.
- ❑ NFS shares are configured in /etc/exports and activated with the **exportfs -r** command.
- ❑ Firewalls can be set by enabling the nfs service through the appropriate zone in firewalld.
- ❑ In most cases, required booleans for SELinux are already active.
- ❑ To disallow read/write access in SELinux, disable the nfs_export_all_rw boolean.
- ❑ When NFS directories are mounted, they should appear seamless. User permissions work in the same way as with a local directory.

### Test an NFS Client

- ❑ Clients can mount permanent NFS shares through /etc/fstab.
- ❑ You can review shared directories on a client with the **showmount** command.
- ❑ The **mount** command is designed to mount directories shared via NFSv4 and NFSv3.
- ❑ If an NFS server fails, it can "hang" an NFS client. The **soft** and **timeo** options to the **mount** command can help prevent such hangs. However, using them would risk compromising the integrity of the data if a system crashes.

### NFS with Kerberos

- ❑ By default, NFS is insecure because it trusts the UID/GID sent by clients.
- ❑ When integrated with Kerberos, NFS can provide strong authentication (**sec=krb5**), communication integrity (**sec=krb5i**), and encryption (**sec=krb5p**).
- ❑ To configure Kerberos-based NFS shares, specify the appropriate security parameter via the **sec=** option on the NFS clients and server.

- ❑ The **nfs-secure-server** service must be running on the NFS server to provide Kerberos services.
- ❑ The **nfs-secure** service must be running on the NFS clients to support Kerberos-authenticated mounts.
- ❑ NFS with Kerberos requires you to set up a KDC, as explained in Chapter 12.

## SELF TEST

The following questions will help you 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 Network File System (NFS) Server

1. In the /etc/exports file, you want to export the /data directory as read-only to all hosts and grant read and write permission to the hostname superv in the example.com domain. What directive would you enter in that file?

   ______________________________

2. Once you've configured /etc/exports, what command exports these shares?

   ______________________________

3. What port number is associated with the portmapper?

   ______________________________

4. What port number is associated with NFSv4?

   ______________________________

5. What is the NFS configuration option that supports access by the root administrative user?

   ______________________________

### Test an NFS Client

6. You're experiencing problems with NFS clients for various reasons, including frequent downtime on the NFS server and network disconnections between NFS clients and servers. What type of mounting can prevent NFS clients from hanging and retrying NFS requests indefinitely?

   ______________________________

7. What is the command that can display NFS shared directories from the outsider1.example.org system?

   ______________________________

### NFS with Kerberos

8. Which service should you start on an NFS client to support Kerberos-based authentication via the **rpcgssd** daemon?

   ______________________________

9. What directive should you include to mount an NFS share with Kerberos authentication and encryption?

   ______________________________

10. What directive should you add to /etc/exports to export an NFS share with standard file access permissions and optionally with Kerberos authentication?

    ______________________________

## LAB QUESTIONS

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

Red Hat presents its exams electronically. For that reason, the labs in this chapter are available in the Chapter 16/ subdirectory from the media that accompanies the book. In case you haven't yet set up RHEL 7 on a system, refer to Chapters 1 and 2 for installation instructions.

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

## SELF TEST ANSWERS

### The Network File System (NFS) Server

1. The following entry in /etc/exports would export the /data directory as read-only to all hosts and grant read and write permission to the host superv in the example.com domain:

   ```
   /data superv.example.com(rw,sync) (ro,sync)
   ```

2. Once you've revised /etc/exports, the **exportfs -a** command exports all filesystems. Yes, you can also re-export filesystems with the **exportfs -r** command.
3. The port number associated with the portmapper is UDP port 111.
4. The port number associated with NFSv4 is TCP port 2049.
5. The NFS configuration option that supports access by the root administrative user is **no_root_squash**.

### Test an NFS Client

6. Soft mounting and timeouts associated with the **soft** and **timeo** options can prevent clients from hanging and retrying NFS requests indefinitely.
7. The command that can display NFS shared directories from the named remote system is **showmount -e outsider1.example.org**.

### NFS with Kerberos

8. You should start the **nfs-secure** server to provide support for Kerberos-based authentication on a client via the **rpcgssd** daemon.
9. The directive that you should include to mount an NFS share with Kerberos authentication and encryption is **sec=krb5p**.
10. You can export the share with the **sec=sys:krb5p** security option.

## LAB ANSWERS

### Lab 1

When this lab is complete, you'll see the following features on the system with the NFS server:

- The nfs-utils RPM in the list of installed packages.
- An active NFS service, which can be confirmed in the output to the **systemctl status nfs-server** command.
- A zone-based firewall that supports access to the nfs service. It should also be limited by IP address network.

In addition, you'll be able to perform the following tasks from the NFS client:

- You can run the **showmount -e server1.example.com** command, where server1.example.com is the name of the NFS server system (substitute if and as needed).
- You can mount the shared directory as the root user with the **mount -t nfs server1.example .com:/shared /testing** command.
- The first time the share is mounted, you should be able to copy local files as the root user to the /testing directory.
- The second time the share is mounted, with the **no_root_squash** directive in effect, such copying should not work, at least from the client root user account.

### Lab 2

This lab is the first step toward creating a single /home directory for your network. Once you get it working on a single client/server combination, you can set it up on all clients and servers. You can then use an LDAP server to set up a single Linux/Unix database of usernames and passwords for the network. Alternatively, matching usernames (with matching UID and GID numbers) on different local systems should also work. On the NFS server, take the following steps:

1. Set up a couple of users and identifying files such as user1 and user1.txt on the system being used as the NFS server.
2. Share the /home directory in /etc/exports on the server1.example.com client. You can do this in this file with the following command:

   ```
   /home *.example.com(rw,sync)
   ```

3. Export this directory with the following command:

   ```
   # exportfs -a
   ```

4. Make sure that the exported /home directory shows in the export list. On the local server, you can do this with the following command:

```
# showmount -e server1.example.com
```

5. If problems appear during this process, check the /etc/exports file carefully. Make sure there aren't extra spaces in /etc/exports, even at the end of a code line. Make sure the NFS service is actually running with the **systemctl status nfs-server** command.
6. You may also want to check your firewall and make sure the appropriate services described in this chapter are running with the **rpcinfo -p** command.
7. Remember to make sure that the NFS server starts automatically the next time the system is booted. One way to do so is with the following command:

```
# systemctl enable nfs-server
```

Now on the NFS client, take the following steps to connect to the shared /home directory:

1. Make sure you can see the shared /home directory. You can substitute the IP address of the server1.example.com system:

```
# showmount -e server1.example.com
```

2. Now mount the share that is offered on the local /remote directory:

```
# mount -t nfs server1.example.com:/home /remote
```

3. Run the **mount** command. If you see the NFS mount, all is well.
4. Examine the mounted /home directory. Look for the *.txt files created earlier in this lab. If you find those files, you've successfully created and connected to the /home directory share.
5. To make the mount permanent, add it to the /etc/fstab file on the client. Once you've added a line such as the following to that file, the Linux client automatically mounts the shared /home directory from the NFS server the next time the client is booted, with the soft option and a timeout of 100 seconds, which can help prevent a "hang":

```
server1.example.com:/home    /remote nfs soft,timeout=100  0  0
```

## Lab 3

The reference to SELinux is deliberate and should provide an important hint. You may not have enough time to modify every directory shared and configured in the /etc/exports file on each NFS server. One simple way to prevent writes to shared NFS directories is to deactivate the associated SELinux boolean, with the following command:

```
# setsebool -P nfs_export_all_rw off
```

You should then be able to test the result with the next mounting of a shared NFS directory.

## Lab 4

This lab is an extension of Exercise 16-2 and tries to familiarize you with some of the common problems when configuring NFS shares with Kerberos.

Export the share with the **sec=sys:krb5:krb5i:krb5p** security option to provide optional Kerberos authentication, communication integrity, and encryption. See if the tester1.example.com client can mount the NFS share using any of the available security methods. Reproduce the troubleshooting scenarios described in the lab and take note of the error messages you encounter.