Chapter

2

Introduction to Services

Previous versions of the CompTIA Linux+ certification exam included an objective on Linux server services, such as web servers, file servers, and database servers. The XK0-005 exam has dropped that as a formal objective, but there are other objectives that assume you know what Linux server services are and how they accept connections from clients. While Linux has had a rough time breaking into the desktop market, it has thrived in the server market, so it is important to know how Linux servers work and what server software packages are popular these days.

The popularity of Linux servers has much to do with their versatility, performance, and cost. This chapter helps get you up to speed in how Linux servers operate and covers the most common server software packages you can install and run in Linux to provide services to your network clients.

What Is a Linux Server?

Before we dive into Linux server details, we will first explain what we mean by a Linux server and show how a Linux server differs from a Linux desktop.

Both Linux desktops and Linux servers use the same Linux kernel, run the same shells, and even have the ability to run the same programs. The difference comes in which programs they primarily run and how those programs run on the system.

Linux desktops primarily focus on personal programs that you run from a graphical desktop interface, such as when you browse the Internet or edit a document. The graphical desktop provides an easy interface for users to interact with the operating system and all files and programs. You start programs by selecting them from a menu system or clicking a desktop icon. In the desktop world, everything is interactive.

Linux servers primarily operate without any human interaction. There's no one sitting at a desktop launching applications (and in fact, many servers don't even have a dedicated monitor and keyboard).

The server runs programs that provide shared resources (called *services*) to multiple users (clients), normally in a network environment. Many services run all the time, even when no clients are actively using them.

Server programs seldom rely on a graphical interface. Instead, they almost always utilize the Linux shell's command-line interface (CLI) to interact with a server administrator, and often, the administrator connects to the server from a remote client to perform any interactive work with the services.

Since there's little interaction with a human operator, servers must know how to launch the programs that provide the services to clients on their own. How the server runs those services can differ from server to server and service to service. The following sections describe how Linux servers start services and how they provide access to those services to clients.

Launching Services

There are two primary ways Linux servers run service programs:

- As a background process, running at all times and listening for requests
- As a process spawned by a parent program that listens for the requests

When a Linux service program runs continually as a background process, it's called a *daemon*. Linux servers often utilize scripts to launch service daemons as soon as the server boots up (see Chapter 6, "Maintaining System Startup and Services").

Linux daemon programs often end with the letter *d* to indicate they're daemon processes. Listing 2.1 shows an example of the MySQL database server daemon running in the background on a server.

Listing 2.1: Listing the MySQL database server daemon process

```
$ ps ax | grep mysql
 5793 ?        Sl     0:00 /usr/sbin/mysqld --daemonize --pid
file=/run/mysqld/mysqld.pid
 5900 pts/0    S+     0:00 grep --color=auto mysql
$
```

The mysqld daemon program listens for network connections from clients. When the daemon receives a request from a client, it processes the request and returns data to the client via the same network channel.

Note that the name for a background program running in Linux is "daemon" and not "demon," as it is often confused with. Daemons are from Greek mythology and were supernatural beings that provided help to humans when needed.

The more services a Linux server supports, the more daemons it must have running in the background, waiting for client requests. Each daemon requires memory resources on the server, even when it's just listening for clients. While today's servers have lots of memory at their disposal, that wasn't always the case in the old days of Linux. Thus came the necessity of *super-servers*.

Super-servers are programs that listen for network connections for several different applications. When the super-server receives a request for a service from a client, it spawns the appropriate service program.

The original super-server program created for Linux was the *internet daemon (inetd)* application. The inetd program ran as a daemon, listening for specific requests from clients

and launching the appropriate service program when needed. The inetd program uses the `/etc/inetd.conf` configuration file to allow you to define the services for which it handles requests.

The *extended internet daemon (xinetd)* application is an advanced version of inetd. It too launches service programs as requested by clients, but it contains additional features, such as access control lists (ACLs), more advanced logging features, and the ability to set schedules to turn services on and off at different times of the day or week.

Linux systems that utilize the Systemd startup method (see Chapter 6) can utilize `systemd` unit files to replace the functionality provided by inetd or xinetd.

Listening for Clients

A standard Linux server supports lots of services. Usually, a single Linux server will support multiple services at the same time. This means multiple clients will be making requests to the server for multiple services. The trick is in getting requests from clients to the correct server service.

Each service, whether it's running as a daemon or running from a super-server, uses a separate network protocol to communicate with its clients. Common service protocols are standardized by the Internet Engineering Task Force (IETF) and published as Request for Comments (RFC) documents. Each server software program communicates with its clients using the protocol specified for its service, such as a web server using HTTP or an email server using SMTP.

The network protocol for a service defines exactly how network clients communicate with the service, using preassigned network *ports*. Ports are defined within the TCP and UDP standards to help separate network traffic going to the same IP address. The IETF assigns different services to different ports for communication. This works similarly to telephone extensions used in a large business. You enter a single phone number to reach the business and then select a separate extension to get to a specific individual within the office. With services, clients use a common IP address to reach a server and then different ports to reach individual services.

The IETF has defined a standard set of ports to common services used on the Internet. These are called *well-known ports*. Table 2.1 shows just a few of the more common well-known ports assigned.

A host of Linux services are available for serving applications to clients on the network. The `/etc/services` file contains all of the ports defined on a Linux server.

The following sections explore the different types of services you will find on Linux servers as well as common Linux applications that provide those services.

TABLE 2.1 Common Internet well-known port numbers

Port number	Protocol	Description
20 and 21	FTP	File Transfer Protocol (FTP) is used for sending files to and from a server.
22	SSH	The Secure Shell (SSH) protocol is used for sending encrypted data to a server.
23	Telnet	Telnet is an unsecure protocol for providing an interactive interface to the server shell.
25	SMTP	The Simple Mail Transport Protocol (SMTP) is used for sending email between servers.
53	DNS	The Domain Name System (DNS) provides a name service to match IP addresses to computer names on a network.
67	DHCP	The Dynamic Host Configuration Protocol (DHCP) enables client computers to obtain a valid IP address on a network automatically.
80	HTTP	The Hypertext Transfer Protocol (HTTP) allows clients to request web pages from servers.
109 and 110	POP	The Post Office Protocol (POP) allows clients to communicate with a mail server to read messages in their mailbox.
137–139	SMB	Microsoft servers use the Server Message Block (SMB) protocol for file and print sharing with clients.
143, 220	IMAP	The Internet Message Access Protocol (IMAP) provides advanced mailbox services for clients.
389	LDAP	The Lightweight Directory Access Protocol (LDAP) provides access to directory services for authenticating users, workstations, and other network devices.
443	HTTPS	The secure version of HTTP provides encrypted communication with web servers.
2049	NFS	The Network File System (NFS) provides file sharing between Unix and Linux systems.

Serving the Basics

There are some basic Internet services that Linux servers are known to do well and that have become standards across the Internet. The three Internet services Linux servers provide are as follows:

- Web services
- Database services
- Email services

The following sections discuss each of these types of Linux services and show you the open source software packages commonly used to support them.

Web Servers

By far the most popular use of Linux servers on the Internet is as a web server. Linux-based web servers host the majority of websites, including many of the most popular websites.

As is true for many Linux applications, there are multiple programs that you can use to build a Linux web server. These are the most popular ones you'll run into and should know about.

The Apache Server

Over the years, the *Apache* web server has become one of the most popular web servers on the Internet. It was developed from the first web server software package created by the National Center for Supercomputing Applications (NCSA) at the University of Illinois.

The Apache web server has become popular due to its modularity. Each advanced feature of the Apache server is built as a plug-in module. When features are incorporated as modules, the server administrator can pick and choose just which modules a particular server needs for a particular application. This helps reduce the amount of memory required to run the Apache server daemons on the system.

The nginX Server

The *nginX* web server (pronounced like "engine-X") is the relatively new kid on the block. Released in 2004, nginX was designed as an advanced replacement for the Apache web server, improving on performance and providing some additional features, such as working as a web proxy, mail proxy, web page cache, and even load-balancing server. While the Apache web server can be configured to provide some of these features by using modules, such as load balancing, the nginX web server was designed to have these features built in to increase performance.

The core nginX program has a smaller memory footprint than the larger Apache program, making it ideal for high-volume environments. It's capable of handling over 10,000 simultaneous network client connections.

While still relatively new, the nginX web server is gaining in popularity, especially in high-traffic web environments. One configuration that's becoming popular is to use a combination of the nginX web server as a load-balancing front end to multiple Apache web servers on the backend. This takes advantage of the nginX server's capabilities of handling large traffic volumes and the Apache web server's versatility in handling dynamic web applications.

The lighthttpd Package

On the other end of the spectrum, there may be times you need a lightweight web server to process incoming client requests for a network application. The *lighthttpd* package provides such an environment.

The lighthttpd web server is known for low memory usage and low CPU usage, making it ideal for smaller server applications, such as in embedded systems. It also incorporates a built-in database, allowing you to combine basic web and database services in a single package.

Database Servers

Storing and retrieving data is an important feature for most applications. Although the use of standard text files is often enough for simple data storage applications, there are times when more advanced data storage techniques are required.

The advent of the relational database allowed applications to quickly store and retrieve data. Relational database servers allowed multiple clients to access the same data from a centralized location. The Structured Query Language (SQL) provides a common method for clients to send requests to the database server and retrieve the data.

Many popular commercial database servers are available for Unix and Windows (and even Linux); a few high-quality open source databases have risen to the top in the Linux world. These database server packages offer many (if not most) of the same features as the expensive commercial database packages and can sometimes even outperform the commercial packages.

The following sections discuss the three most popular open source database servers you'll encounter when working in the Linux environment.

The PostgreSQL Server

The *PostgreSQL* database server started out as a university project and became an open source package available to the public in 1996. The goal of the PostgreSQL developers was to implement a complete object-relational database management system to rival the popular commercial database servers of the day.

PostgreSQL is known for its advanced database features. It follows the standard atomicity, consistency, isolation, and durability (ACID) guidelines used by commercial databases and supports many of the fancy features you'd expect to find in a commercial relational database server, such as transactions, updatable views, triggers, foreign keys, functions, and stored procedures.

PostgreSQL is very versatile, but with versatility comes complexity. In the past the PostgreSQL database had a reputation for being somewhat slow, but it has made vast improvements in performance. Unfortunately, old reputations are hard to shake, and PostgreSQL still struggles to gain acceptance in the web world.

The MySQL Server

Unlike the PostgreSQL package, the *MySQL* database server didn't originally try to compete with commercial databases. Instead, it started out as a project to create a simple but fast database system. No attempt was made to implement fancy database features; it just offers basic features that performed quickly.

Because of its focus on speed, the MySQL database server became the de facto database server used in many high-profile Internet web applications. The combination of a Linux server running the Apache web server and the MySQL database server and utilizing the PHP programming language became known as the LAMP platform and can be found in Linux servers all over the world.

Since its inception, the MySQL database has added features that can rival those found in PostgreSQL and commercial databases. However, staying true to its roots, MySQL still maintains the option of utilizing the faster storage engine that it became famous for.

In 2008 the MySQL project was acquired by Sun Microsystems. In 2010, when Oracle purchased Sun Microsystems, by default it also took control over MySQL development. This concerned many in the open source community, and shortly after the purchase a group of MySQL developers left Oracle to start the MariaDB project. MariaDB is a replica of MySQL, using the same source code and having the same features (with some new features added). Many Linux distributions now use MariaDB by default instead of MySQL, so don't be alarmed if you see that.

The MongoDB Server

With the rising popularity of object-oriented programming and application design, the use of object-oriented databases has also risen. Currently one of the most popular object-oriented methods of storing data is called *NoSQL*.

As its name suggests, a NoSQL database system stores data differently than the traditional relational database systems using SQL. A NoSQL database doesn't create tables but instead stores data as individual documents. Unlike relational tables, each NoSQL document can contain different data elements, with each data element being independent from the other data elements in the database.

One NoSQL database package that is gaining in popularity is the *MongoDB* package. MongoDB was released in 2009 as a full NoSQL-compliant database management system. It stores data records as individual *JavaScript Object Notation (JSON)* elements, making each data document independent of the others.

The MongoDB database server supports many relational database features, such as indexes, queries, replication, and even load balancing. It allows you to incorporate JavaScript in queries, making it a very versatile tool for querying data.

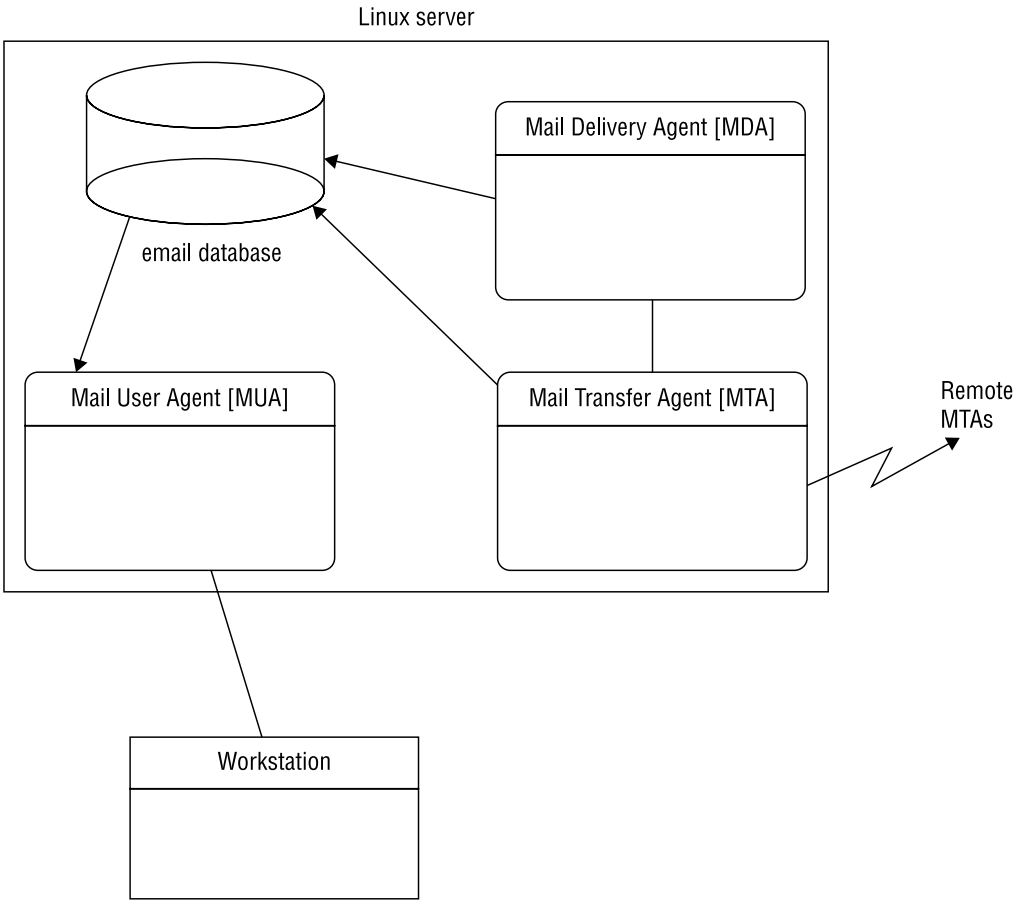
The MongoDB server installs with a default of no security—anyone can connect to the server to add and retrieve data records. This "gotcha" has been a problem for even some high-profile websites where data has been breached. Please be careful when using a MongoDB database for your web applications.

Mail Servers

At one time, email was the backbone of the Internet. Just about everyone had an email address, and it was crucial to be plugged into an email server to communicate with the world. While these days newer technology is taking over (such as texting, tweeting, and messaging), email is still a vital operation for most Internet users. Just about every Linux server installation uses some type of email server package.

Instead of having one monolithic program that handles all of the pieces required for sending and receiving mail, Linux uses multiple small programs that work together in the processing of email messages. Figure 2.1 shows you how most open source email software modularizes email functions in a Linux environment.

FIGURE 2.1 The Linux modular email environment



As you can see in Figure 2.1, the Linux email server is normally divided into three separate functions:

- The mail transfer agent (MTA)
- The mail delivery agent (MDA)
- The mail user agent (MUA)

MUA is the program that interacts with end users, allowing them to view and manipulate email messages. Therefore, the MUA programs don't usually run on the server side but rather on the client side. Graphical applications such as Evolution and K-Mail are popular for reading email in Linux desktop environments. The MTA and MDA functions are found on the Linux server. The following sections show the more common MTA and MDA applications you'll see in Linux.

The Mail Transfer Agent

The *mail transfer agent (MTA)* is responsible for handling both incoming and outgoing email messages on the server. For each outgoing message, the MTA determines the destination host of the recipient address. If the destination host is a remote mail server, the MTA must establish a communication link with another MTA program on the remote host to transfer the message.

There are quite a few MTA software packages for the Linux environment, but the Linux+ exam focuses on three of them:

- *sendmail*: The sendmail MTA package gained popularity by being extremely versatile. Many of the features in sendmail have become synonymous with email systems—virtual domains, message forwarding, user aliases, mail lists, and host masquerading. Unfortunately, sendmail is very complex to configure correctly. Its large configuration file is sometimes overwhelming for novice mail administrators to handle.
- *Postfix*: The Postfix MTA was written as a modular application, using several different programs to implement the MTA functionality. One of Postfix's best features is its simplicity. Instead of one large complex configuration file, Postfix uses just two small configuration files with plaintext parameters and value names to define the functionality.
- *Exim*: The Exim MTA package sticks with the sendmail model of using one large program to handle all the email functions. It attempts to avoid queuing messages as much as possible, instead relying on immediate delivery in most environments.

The Mail Delivery Agent

Often, Linux implementations rely on separate stand-alone *mail delivery agent (MDA)* programs to deliver messages to local users. Because these MDA programs concentrate only on delivering messages to local users, they can add bells and whistles that aren't available in MTA programs that include MDA functionality.

The MDA program receives messages destined for local users from the MTA program and then determines how those messages are to be delivered. Messages can be delivered directly to the local user account or to an alternate location defined by the user, often by incorporating filters.

There are two common MDA programs used in Linux:

- *Binmail:* The binmail program is the most popular MDA program used in Linux. Its name comes from its normal location in the system, `/bin/mail`. It has become popular thanks to its simplicity. By default, it can read email messages stored in the standard `/var/spool/mail` directory, or you can point it to an alternative mailbox.
- *Procmail:* The procmail program was written by Stephen R. van den Berg and has become so popular that many Linux implementations install it by default. The popularity of procmail comes from its versatility in creating user-configured recipes that allow a user to direct how the server processes received mail. A user can create a personal `.procmailrc` file in their `$HOME` directory to direct messages based on regular expressions to separate mailbox files, forward messages to alternative email addresses, or even send messages directly to the `/dev/null` file to trash unwanted email automatically.

Serving Local Networks

Besides running large Internet web and database applications, Linux servers are also commonly used in local network environments to provide simple network services. Running a local network requires lots of behind-the-scenes work, and the Linux server is up to the task. This section walks through the most common services you'll find used on all sizes of local networks.

File Servers

These days, the sharing of files has become a necessity in any business environment. Allowing multiple employees to create and edit files in a common folder can greatly improve collaboration efforts in any project.

While sharing files via a web server is common in a wide area network environment, there are easier ways to do that within a local network. There are two basic methods for sharing files in a local network environment:

- Peer-to-peer
- Client-server

In a peer-to-peer network, one workstation enables another workstation to access files stored locally on its hard drive. This method allows collaboration between two employees on a small local network but becomes somewhat difficult if you need to share data between more than two people.

The client-server method of file sharing utilizes a centralized *file server* for sharing files that multiple clients can access and modify as needed. However, with the centralized file server, an administrator must control who has access to which files and folders, protecting them from unauthorized access.

In the Linux world, there are two common server software packages used for sharing files: NFS and Samba.

NFS

The *Network File System (NFS)* is a protocol used to share folders in a network environment. With NFS, a Linux system can share a portion of its virtual directory on the network to allow access by clients as well as other servers.

In Linux, the software package used to accomplish this is *nfs-utils*. The nfs-utils package provides the drivers to support NFS as well as the underlying client and server software to both share local folders on the network and connect to remote folders shared by other Linux systems on the local network. Using nfs-utils, your Linux system can mount remotely shared NFS folders almost as easily as if they were on a local hard drive partition.

Samba

These days, Microsoft Windows workstations and servers have become the norm in many business environments. While Windows workstations and servers can use NFS, the default file sharing method used in Windows is the *System Message Block (SMB)* protocol, created by Microsoft. Although Microsoft servers use proprietary software, Microsoft has released the SMB protocol as a network standard, so it's possible to create open source software that can interact with Windows servers and clients using SMB.

The *Samba* software package (note the clever use of embedding SMB in the name) was created to allow Linux systems to interact with Windows clients and servers. With Samba, your Linux system can act either as a client, connecting to Windows server shared folders, or as a server, allowing Windows workstations to connect to shared folders on the Linux system. Samba does take some work in configuring the correct parameters to manage access to your shared folders.

Print Servers

In a business environment, having a printer for every person in the office is somewhat of a wasted expense. The ability to share network printers has become a requirement for most offices and has also become popular in many home environments.

The standard Linux print sharing software package is called the *Common Unix Printing System (CUPS)*. The CUPS software allows a Linux system to connect to any printer resource, either locally or via a network, by using a common application interface that operates over dedicated printer drivers. The key to CUPS is the printer drivers. Many printer manufacturers create CUPS drivers for their printers, so Linux systems can connect with their printers. For connecting to network printers, CUPS uses the *Internet Printing Protocol (IPP)*.

Besides connecting to a network printer, the CUPS system also allows you to share a locally attached printer with other Linux systems. This allows you to connect a printer to a Linux server and share it among multiple users in a local network.

The Samba software package can also interact with printers shared on Microsoft networks. You can connect your Linux workstation to printers shared on Windows networks using Samba, or you can even share your own locally attached Linux printer with Windows workstations.

Network Resource Servers

Running a local network requires quite a few different resources to keep clients and servers in sync. This is especially true for larger networks where network administrators must manage many different types of clients and servers.

Fortunately, Linux provides a few different service packages that network administrators can use to make their lives easier. The following sections walk through some of the basic network-oriented services that you may see on a Linux server.

IP Addresses

Every device on a local network must have a unique IP address to interact with other devices on the network. For a small home network, that may not be too difficult to manage, but for large business networks, that task can be overwhelming.

To help simplify that requirement, developers have created the *Dynamic Host Configuration Protocol (DHCP)*. Clients can request a valid IP address for the network from a DHCP server. A central DHCP server keeps track of the IP addresses assigned, ensuring that no two clients receive the same IP address.

These days you can configure many different types of devices on a network to be a DHCP server. Most home broadband routers provide this service, as well as most server-oriented operating systems, such as Windows servers and, of course, Linux servers.

The most popular Linux DHCP server package is maintained by the Internet Systems Consortium (ISC) and is called *DHCPd*. Just about all Linux server distributions include this in their software repositories.

Once you have the DHCPd server running on your network, you'll need to tell your Linux clients to use it to obtain their network addresses. This requires a DHCP client software package. For Linux DHCP clients, there are three popular packages that you can use:

- dhclient
- dhcpcd
- pump

Most Debian- and Red Hat–based distributions use the *dhclient* package and even install it by default when a network card is detected during the installation process. The dhcpcd and pump applications are less known, but you may run into them.

Logging

Linux maintains log files that record various key details about the system as it runs. The log files are normally stored locally in the `/var/log` directory, but in a network environment it can come in handy to have Linux servers store their system logs on a remote logging server.

The remote logging server provides a safe backup of the original log files, plus a safe place to store logs in case of a system crash or a break-in by an attacker.

There are two main logging packages used in Linux, and which one a system uses depends on the startup software it uses (see Chapter 6):

- *rsyslogd*: The SysVinit and Upstart systems utilize the rsyslogd service to accept logging data from remote servers.
- *journald*: The Systemd system utilizes the journald service for both local and remote logging of system information.

Both rsyslogd and journald use configuration files that allow you to define just how data is logged and what clients the server accepts log messages from.

Name Servers

Using IP addresses to reference servers on a network is fine for computers, but humans usually require some type of text to remember addresses. Enter the Domain Name System (DNS). DNS maps IP addresses to a host naming scheme on networks. A DNS server acts as a directory lookup to find the names of servers on the local network.

Linux servers use the *BIND* software package to provide DNS naming services. The BIND software package was developed in the very early days of the Internet (the early 1980s) at the University of California, Berkeley, and is released as open source software.

The main program in BIND is *named*, the server daemon that runs on Linux servers and resolves hostnames to IP addresses for clients on the local network. The beauty of DNS is that one BIND server can communicate with other DNS servers to look up an address on remote networks. This allows clients to point to only one DNS name server and be able to resolve any IP address on the Internet!

The DNS protocol is text based and is susceptible to attacks, such as hostname spoofing. The DNSSEC protocol incorporates a layer of encryption around the standard DNS packets to help provide a layer of security in the hostname lookup process. Ensure that your BIND installation supports DNSSEC for the proper security.

Network Management

Being responsible for multiple hosts and network devices for an organization can be an overwhelming task. Trying to keep up with what devices are active or which servers are running at capacity can be a challenge. Fortunately for administrators, there's a solution.

The *Simple Network Management Protocol (SNMP)* provides a way for an administrator to query remote network devices and servers to obtain information about their configuration, status, and even performance. SNMP operates in a simple client/server paradigm. Network devices and servers run an SNMP server service that listens for requests from SNMP client packages. The SNMP client sends requests for data from the SNMP server.

The SNMP standards have changed somewhat drastically over the years, mainly to help add security and boost performance. The original SNMP version 1 (called SNMPv1) provided for only simple password authentication of clients and passed all data as individual

plaintext records. SNMP version 2 (called SNMPv2) implemented a basic level of security and provided for the bulk transmission of monitoring data to help reduce the network traffic required to monitor devices. The current version (SNMPv3) utilizes both strong authentication and data encryption capabilities and provides a more streamlined management system.

The most popular SNMP software package in Linux is the open source *net-snmp* package. This package has SNMPv3 compatibility, allowing you to securely monitor all aspects of a Linux server remotely.

Time

For many network applications to work correctly, both servers and clients need to have their internal clocks coordinated with the same time. The *Network Time Protocol (NTP)* accomplishes this. It allows servers and clients to synchronize on the same time source across multiple networks, adding or subtracting fractions of a second as needed to stay in sync.

For Linux systems, the *ntpd* program synchronizes a Linux system with remote NTP servers on the Internet. It's common to have a single Linux server use ntpd to synchronize with a remote time standard server and then have all other servers and clients on the local network sync their times to the local Linux server.

Implementing Security

These days, security is at the top of every system administrator's list of worries. With a seemingly endless supply of people trying to break into servers, implementing security is a vital role for every Linux administrator.

Fortunately, Linux provides several layers of security that you can implement in your Linux server environment. The following sections walk through the server software packages that you may run into as you implement security in your Linux servers.

Authentication Server

The core security for Linux servers is the standard userid and password assigned to each individual user on the system and stored in either the `/etc/passwd` (on non-secure legacy systems) or the `/etc/shadow` file. Each Linux server maintains its own list of valid user accounts that have access on that server.

For a large network environment where users may have to access resources on multiple Linux servers, trying to remember multiple userids and passwords can be a challenge. Fortunately, there are a few different methods for sharing user account databases across multiple Linux servers on a network.

NIS

The *Network Information System (NIS)* is a directory service that allows both clients and servers to share a common naming directory. The NIS naming directory is often used as a common repository for user accounts, hostnames, and even email information on local networks. The NIS+ protocol expands on NIS by adding security features.

The *nis-utils* package is an open source project for implementing an NIS or NIS+ directory in a Linux environment. It's included in most Linux distribution repositories.

The NIS system was originally designed at Sun Microsystems and released under the clever name Yellow Pages (YP). However, due to trademark infringement, the name had to be changed to the more boring NIS.

Kerberos

Kerberos was developed at MIT as a secure authentication protocol. It uses symmetric-key cryptography to securely authenticate users with a centralized server database. The entire authentication process is encrypted, making it a secure method of logging into a Linux server.

You can use the Kerberos authentication system for more than simple server logins. Many common Linux server applications provide plug-in modules to interface with a Kerberos database for authenticating application users.

LDAP

Network authentication systems have taken off in the commercial networking world. Microsoft's Active Directory system is by far the most popular network authentication system used today. However, the open source world isn't too far behind, creating its own network directory system.

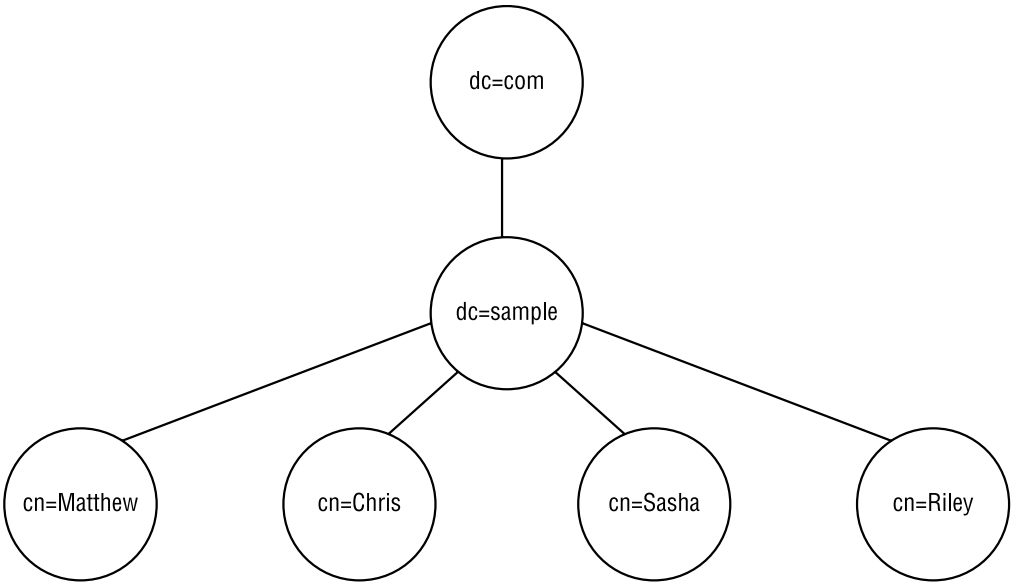
The *Lightweight Directory Access Protocol (LDAP)* was created at the University of Michigan to provide simple network authentication services to multiple applications and devices on a local network. The most popular implementation of LDAP in the Linux world is the *OpenLDAP* package.

OpenLDAP allows you to design a hierarchical database to store objects in your network. In the hierarchical database, objects are connected in a treelike fashion to one another, as shown in Figure 2.2.

The hierarchical databases allows you to group objects by types, such as users and servers, or by location, or both. This provides a flexible way of designing authentication for your local network.

The OpenLDAP package consists of both client and server programs. The client program allows systems to access an OpenLDAP server to authenticate requests made by clients or other network objects.

FIGURE 2.2 A sample LDAP directory tree



Certificate Authority

The days of assigning every user on your server a userid and password are nearing an end (if it hasn't already come). The userid/password method of logging into a server is fraught with security issues—sharing user accounts, simple passwords, and even accounts with no passwords assigned.

A better method of authenticating users is using *certificates*. A certificate is an encrypted key that implements a two-factor authentication method. To log into a server, a user must have two things:

- Something they possess, such as the certificate file
- Something they know, such as a PIN

A certificate identifies a specific user on the network. Only one user should have the certificate and know the PIN required to access the certificate. However, it's important that the server trusts the certificate as well. For that, you need a certificate authority.

The *OpenSSL* package provides standard certificate functions for both servers and clients. You can set up your Linux server to create certificates for clients and then authenticate them for network applications.

Access Server (SSH)

Remotely accessing servers in today's environment is risky. There are plenty of people around snooping on networks, looking for information they can steal. Logging into a server from a remote location using a plaintext protocol such as Telnet or FTP is not a good idea anymore.

Instead, you should use a remote access protocol that incorporates encryption between the client and server. The *Secure Shell (SSH)* provides a layer of encryption around data sent across the network.

The most popular software package that implements SSH in the Linux environment is the *OpenSSH* package. The OpenSSH package provides secure Telnet, FTP, and even remote copy features using SSH.

The OpenSSH program also supports a feature called tunneling. With tunneling, you can wrap any type of network transaction with an encryption layer, thus protecting any type of network application even if it's not directly supported by the OpenSSH software.

Virtual Private Networks

Remotely connecting to servers on your local network via the Internet can be a dangerous thing. Your network traffic takes many hops between many intermediary networks before getting to your servers, providing lots of opportunities for prying eyes to snoop on your data.

The solution to remotely connecting to resources on a local network is the *virtual private network (VPN)* protocol. The VPN protocol creates a secure point-to-point tunnel between a remote client or server and a VPN server on your local network. This provides a secure method for remotely accessing any server on your local network.

In Linux, a popular VPN solution is the *OpenVPN* package. The OpenVPN package runs as a server service on a Linux server on your local network. Remote clients can use OpenVPN to connect with the OpenVPN server to establish connectivity to the server and then, once on the server, gain access to the rest of your local network.

Proxy Server

A *web proxy server* allows you to intercept web requests from local network clients. By intercepting the web requests, you have control of how clients interact with remote web servers. The web proxy server can block websites you don't want your network clients to see, and the server can cache common websites so that future requests for the same pages load faster.

The most popular web proxy server in Linux is the *Squid* package. You can configure it to work both as a filter and as a caching server. The nginX web server package discussed earlier also has the ability to work as a web proxy server and is starting to gain in popularity.

Monitoring

If you have multiple Linux servers on your network, trying to keep up with what they're all doing can be a challenge. It's always a good idea for system administrators to peek in on a

server's performance and log files just to be aware if anything bad is about to happen or has already happened.

There are several monitoring tools available in the Linux world. The *Nagios* software package is quickly becoming a popular tool, especially in cloud Linux systems. Nagios uses SNMP to monitor the performance and logs of Linux servers and provide results in a simple graphical window environment.

Improving Performance

Developers and administrators of high-volume Linux applications are always looking for ways to improve the performance of the applications. There are three common methods for improving performance that all Linux administrators should be aware of. This section covers these methods.

Clustering

A computer *cluster* improves application performance by dividing application functions among multiple servers. Each server node in the cluster is configured the same and can perform the same functions, but the cluster management software determines how to split the application functions among the servers. Since each server in the cluster is working on only part of the application, you can use less powerful servers within the cluster than if you had to run the entire application on a single server.

The cluster management software is the key to the performance of the cluster. One of the earliest attempts at creating clusters of inexpensive Linux servers was the Beowulf cluster. The Beowulf cluster relied on parallel processing libraries, such as the *Parallel Virtual Machine (PVM)* library, to distribute an application's library calls between multiple Linux servers.

Newer versions of clustering include the *Apache Hadoop* project and the *Linux Virtual Server (LVS)* project.

Load Balancing

Load balancing is a special application of clustering. A load balancer redirects entire client requests to one of a cluster of servers. While a single server processes the entire request, the client load is distributed among the multiple servers automatically.

Common Linux load-balancing packages include *HAProxy*, the Linux Virtual Server (LVS), and even the nginX web server.

Containers

One of the biggest problems for application developers is creating a development environment that mirrors the actual server environment the applications run in. All too often a

developer will get an application working just fine in a workstation development environment only to see it crash and burn when ported to the server.

Linux *containers* help solve this problem by creating a self-contained environment to encapsulate applications. A container packages all of the necessary application files, library files, and operating system libraries into a bundle that you can easily move between environments.

Several Linux server packages are available that support containers. Currently, the two most popular ones are *Docker* and *Kubernetes*. You can use these packages to easily port application containers to any Linux server, whether it's a physical server, a virtual server, or in the cloud.

Summary

Linux servers provide network applications that support both clients and network devices. Server applications are called services and are launched by the Linux server without human intervention. When a Linux server can launch services directly, they're called daemons. The daemon runs in the background and listens for client connection requests. A super-server runs in the background and listens for client connection requests for multiple services. When a connection request is accepted, the super-server launches the appropriate service.

Linux supports services for all types of applications. The Apache and nginX services provide web server applications for Linux. For database applications, PostgreSQL, MySQL, and MongoDB are the most popular. If you're looking to run an email server, the sendmail, Postfix, or Exim application should do the trick. Linux also works well as a server for a local network environment. There are open source packages for file, print, and network server resources as well as packages for security authentication and certificate applications.

Finally, you can configure Linux servers for fault tolerance by clustering a large group of small servers together to create one large server. The clustering software can either work to split an application to run over several servers simultaneously or assign individual clients to specific servers to implement load balancing. To support application development, Linux servers also support containers. Containers allow developers to migrate the same environment used to develop an application to a production environment, ensuring that applications will work the same in both development and production.

Exam Essentials

Describe the ways to start server programs in Linux. Server programs in Linux can either run continually in the background as a daemon process or be started from a super-server daemon when requested by a client.

Explain how clients know how to contact a server program. Server applications listen for client connections on well-known ports. Clients must send a connection request to the server on the well-known port for the application they want to interact with.

Explain the components commonly used in a LAMP stack. The LAMP stack uses the Linux operating system, the Apache web server, the MySQL database server, and the PHP programming language to provide a platform for web applications.

Describe the difference between a relational database and a NoSQL database. A relational database stores data records in individual data tables. Each data type consists of one or more data fields that contain individual data elements. A data record is an instance of data for each data field in a table. A NoSQL database stores data values in documents. Each document is independent of all of the other documents in the database. Each document can also contain different data elements.

Understand the ways a Linux server can share files in a local network. Linux servers can use the nfs-utils package to communicate with other Linux servers to share folders using NFS. The local Linux server can mount folders from the remote Linux server as if they were local disks. Linux servers can also use the Samba package to share files on Windows local networks with Windows clients and servers as well as map folders located on Windows servers.

Understand which server packages are commonly used to support network features on a local network. The DHCPd package provides DHCP server services to assign IP addresses to clients. The BIND package provides DNS server services to both clients and servers on a local network for hostname resolution. The net-snmp package allows you to implement remote device management using SNMP, and you can use the ntpd package to create an NTP time server for the local network.

Describe how to create a network directory server using Linux. The OpenLDAP package allows you to create an LDAP directory of users and devices on the local network. Clients and other servers can use the LDAP directory to authenticate users and devices on the network.

Explain how to improve the performance of a network application. For network applications in a high-volume environment, you can improve performance by implementing either a cluster or load balancing environment. In a cluster, you can split application functions between multiple servers by using a cluster package such as Apache Hadoop. With load balancing, you can distribute client connections between multiple servers using packages such as HAProxy and Linux Virtual Server (LVS).

Review Questions

1. Which web server is used in the popular LAMP stack?
   - **A.** nginX
   - **B.** Apache
   - **C.** Lighthttpd
   - **D.** PostgreSQL

2. A _____ runs in the background and listens for client connection requests for a single application.
   - **A.** Daemon
   - **B.** Super-server
   - **C.** Shell
   - **D.** Graphical desktop

3. Which open source database provided fast performance and became a popular choice for web applications?
   - **A.** MongoDB
   - **B.** PostgreSQL
   - **C.** MySQL
   - **D.** NoSQL

4. How does a server know what client request is sent to which application daemon?
   - **A.** IP addresses
   - **B.** Ports
   - **C.** Ethernet addresses
   - **D.** Services

5. What popular open source web servers can also perform as a load balancer?
   - **A.** nginX
   - **B.** Apache
   - **C.** PostgreSQL
   - **D.** Lighthttpd

6. What format does MongoDB use to store data elements in the database?
   - **A.** Relational
   - **B.** YaML
   - **C.** JSON
   - **D.** Encrypted

7. Which part of the Linux mail process is responsible for sending emails to remote hosts?
   - **A.** MUA
   - **B.** MTA
   - **C.** MDA
   - **D.** Evolution

8. Which part of the Linux mail process allows you to create filters to automatically redirect incoming mail messages?
   - **A.** MUA
   - **B.** MTA
   - **C.** MDA
   - **D.** Evolution

9. What protocol should you use to mount folders from remote Linux servers on your local Linux server?
   - **A.** SNMP
   - **B.** NTP
   - **C.** DHCP
   - **D.** NFS

10. The _____ software package allows your Windows workstations to mount a folder stored on a Linux server.
    - **A.** ntpd
    - **B.** Samba
    - **C.** DHCPd
    - **D.** Evolution

11. Which two software packages are used in Linux to maintain log files? (Choose two.)
    - **A.** rsyslogd
    - **B.** journald
    - **C.** ntpd
    - **D.** DHCPd

12. Which software program should you load on your Linux server to synchronize its time with a standard time server?
    - **A.** DHCPd
    - **B.** BIND
    - **C.** ntpd
    - **D.** Samba

13. What software package allows a Linux server to print to a network printer?
    A. DHCPd
    B. BIND
    C. ntpd
    D. CUPS

14. If you see the named program running in the background on your Linux server, what service does it provide?
    A. Network time
    B. Hostname resolution
    C. Dynamic IP address allocation
    D. Printing

15. Which authentication package used to be called by the name "Yellow Pages"?
    A. Samba
    B. Kerberos
    C. NIS
    D. BIND

16. What package do you need to install to allow your Linux server to provide IP addresses to clients on your local network?
    A. DHCPd
    B. BIND
    C. ntpd
    D. Evolution

17. The _____ package allows you to create a secure tunnel across a private network to access your local network remotely.
    A. BIND
    B. ntpd
    C. OpenSSH
    D. OpenSSL

18. What server role should you implement to block your local network clients from accessing sports websites during business hours?
    A. A DHCP server
    B. A web server
    C. A web proxy
    D. A container

19. What server role should you implement to increase performance on your company's website?

A. A load balancer

B. A web proxy

C. A DHCP server

D. A container

20. A _____ allows your developers to easily deploy applications between development, test, and production.

A. web proxy

B. DHCP server

C. container

D. cluster