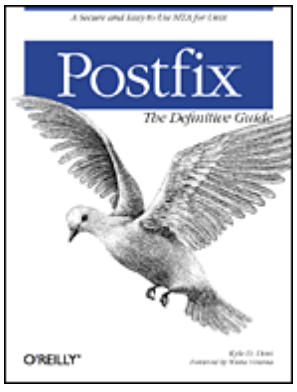
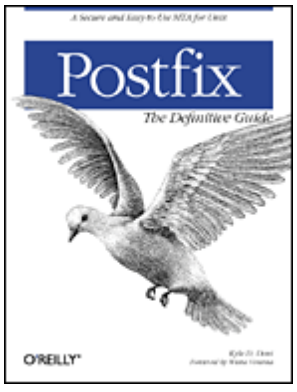


Postfix: The Definitive Guide

By Kyle D. Dent

Publisher: O'Reilly
Pub Date: December 2003
ISBN: 0-596-00212-2
Pages: 264

Postfix: The Definitive Guide eases readers from the basic configuration to the full power of Postfix. It discusses the interfaces to various tools that round out a fully scalable and highly secure email system. These tools include POP, IMAP, LDAP, MySQL, Simple Authentication and Security Layer (SASL), and Transport Layer Security (TLS, an upgrade of SSL). A reference section for Postfix configuration parameters and an installation guide are included.



Postfix: The Definitive Guide

By Kyle D. Dent

Start Reading

Publisher: O'Reilly
Pub Date: December 2003
ISBN: 0-596-00212-2
Pages: 264

Copyright

Printed in the United States of America.

Published by O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly & Associates books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://safari.oreilly.com). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Foreword

All programmers are optimists—these words of wisdom were written down almost thirty years ago by Frederick P. Brooks, Jr.[1] The Postfix mail system is a fine example of this. Postfix started as a half-year project while I was visiting the network and security department at IBM Research in New York state. Although half a year was enough time to replace the mail system on my own workstation, it was not nearly enough to build a complete mail system for general use. Throughout the next year, a lot of code was added while the software was tested by a closed group of experts. And in the five years that followed the public release, Postfix more than doubled in size and in the number of features. Meanwhile, active development continues.

[1] Frederick P. Brooks, Jr.: The Mythical Man-Month: Essays on Software Engineering, Addison Wesley, 1975.

One of the main goals of Postfix is wide adoption. Building Postfix was only the first challenge on the way to that goal. The second challenge was to make the software accessible. While expert users are happy to Read The Friendly Manual that accompanies Postfix, most people need a more gentle approach. Truth be told, I would not expect to see wide adoption of Postfix without a book to introduce the concepts behind the system, and which gives examples of how to get common tasks done. I was happy to leave the writing of this book to Kyle Dent.

Just like Postfix, I see this book as a work in progress. In the time that the first edition of the book was written, Postfix went through several major revisions. Some changes were the result of discussions with Kyle in order to make Postfix easier to understand, some changes added functionality that was missing from earlier versions, and some changes were forced upon Postfix by the big bad ugly world of junk email and computer viruses. Besides the changes that introduced new or extended features, many less-visible changes were made behind the scenes as part of ongoing maintenance and improvement.

This book describes Postfix Version 2.1, and covers some of the differences with older Postfix versions that were widely used at the time of publication. As Postfix continues to evolve, it will slowly diverge from this book, and eventually this book will have to be updated. While it is a pleasure for me to welcome you to this first edition, I already look forward to an opportunity to meet again in the near future.

—Wietse VenemaHawthorne, New YorkSeptember 19, 2003

Preface

I'm always astounded when I think about the early designers of Internet technologies. They were (and many still are) an amazing group of people who developed software and technologies for a network that was minuscule, by comparison with today's Internet. Yet their work scaled and has continued to function in not only a much larger but in a very different environment. The expansion hasn't been completely without growing pains, but that doesn't diminish this amazing feat. Sendmail is an example of one of the early technologies that was written for a different universe, yet is still relevant and handles a large portion of email today.

Postfix has an advantage in that it was built with an awareness of the scope and hostile environment it would have to face. In fact, its creation was motivated by the need to overcome some of the problems of software written in a more innocent age. What a difference a little hindsight can make.

I first started using Postfix when I was working with systems in a security-sensitive environment. The promise of more flexibility and better security caught my interest as soon as I heard about it. I was not disappointed. It didn't take long before I was hooked, and preferred using Postfix everywhere. This book is my attempt to create a reference and a guide to understanding how Postfix works. Its main goal is to explain the details and concepts behind Postfix. It also offers instructions for accomplishing many specific tasks.

Documenting a piece of software that is still under active development is a bit like trying to stop running water. Sadly, this book will be incomplete even before it is out. I've tried to structure the information in the book in such a way as to exclude things that might become irrelevant or quickly out-of-date, so that what you find in the book will be good information for a long time to come. However, you may have to supplement this book with online documentation, web sites, and the Postfix mailing list for coverage of the latest features.

Audience

Postfix is a network application written for Unix. The more you know about networking and Unix, the better equipped you will be to manage a Postfix server. This book tries to explain things in such a way as to be understandable to users new to Unix, but it is unrealistic to think that you could learn to administer a Postfix server without having (or at least acquiring) some Unix knowledge. The book focuses on Postfix itself. Other concepts are explained as needed to understand the functions and configuration of Postfix. If you're new to Unix, you should certainly consult other texts for general Unix information. Unix System Administration Handbook by Evi Nemeth, et al. (Prentice-Hall) is an excellent choice, and includes a helpful section on email. The relevant RFCs mentioned in this book can also be very helpful for understanding the details of a subject.

Organization

Chapter 1 through Chapter 3 provide background information on Postfix and email, Chapter 4 through Chapter 7 discuss general aspects of running a Postfix server, and Chapter 8 through Chapter 15 each present a specific topic that you may or may not need, depending on how you use Postfix:

Chapter 1

Introduces Postfix and some general email concepts. Also discusses some of the design decisions that went into Postfix.

Chapter 2

Covers required topics for understanding other concepts in the book. Anyone with a basic understanding of Unix and email can safely skip this chapter.

Chapter 3

Explains the pieces of the modular architecture of Postfix and how Postfix handles email messages.

Chapter 4

Covers a wide range of topics for configuring and managing a Postfix server.

Chapter 5

Explains how the Postfix queue manager works, and presents the tools used to work with the queue.

Chapter 6

Discusses how DNS is used for email routing. Presents considerations for configuring DNS to work with Postfix.

Chapter 7

Covers how Postfix makes local deliveries and how it operates in conjunction with POP and IMAP servers.

Chapter 8

Conventions Used in This Book

Items appearing in this book are sometimes given a special appearance to set them apart from the regular text. Here's how they look:

Italic

Used for commands, email addresses, URIs, filenames, emphasized text, first references to terms, and citations of books and articles.

Constant width

Used for literals, constant values, code listings, and XML markup.

Constant width italic

Used for replaceable parameter and variable names.

`Constant width bold`

Used to highlight the portion of a code listing being discussed.

These icons signify a tip, suggestion, or general note.

These icons indicate a warning or caution.

Comments and Questions

Please address comments and questions concerning this book to the publisher:
O'Reilly & Associates, Inc.1005 Gravenstein Highway NorthSebastopol, CA 95472(800) 998-9938 (in the United States or Canada)(707) 829-0515 (international or local)(707) 829-0104 (fax)

O'Reilly maintains a web page for this book, that lists errata, examples, and any additional information. You can access this page at:
http://www.oreilly.com/catalog/postfix/

To comment or ask technical questions about this book, send email to:
bookquestions@oreilly.com

For more information about O'Reilly books, conferences, Resource Centers, and the O'Reilly Network, see O'Reilly's web site at:
http://www.oreilly.com/

Acknowledgments

I would first like to thank Wietse Venema for Postfix, of course, but also for his many contributions to the Internet community. Having had the honor to work with him on this book, it is apparent to me that he brings the same level of intelligence and attention to detail to all of his endeavors. This book has benefited greatly from his considerable input.

I have always admired O'Reilly & Associates as a company. After having had the experience of working with them, my admiration has not diminished in the least. My editor, Andy Oram, excellently personifies the goals of the company. I've enjoyed discussions with him, and his comments were always very helpful. I appreciate his enormous patience. Lenny Muellner helped me get going with text-processing tools and I'd like to thank David Chu for his timely assistance when needed. I would also like to thank Robert Romano for turning my crude diagrams into the professional figures you find in the book, and Reg Aubry for guiding the book through the production process.

Several technical reviewers assisted me not only in staying honest and correct in the details, but they also often offered useful stylistic suggestions. Thanks to Rob Dinoff, Viktor Dukhovni (a.k.a. Victor Duchovni), Lutz Jänicke, and Alan Schwartz. I wish I had such a team looking over my shoulder for everything I do.

I would also like to acknowledge the many members of the postfix-users@postfix.org list. It is an active list with a low noise-to-signal ratio, populated by a group of remarkably capable and helpful people. Its members not only help the user community, but have contributed through their comments and discussions to the evolution of Postfix itself.

Finally, I owe a large debt of gratitude to my wife and first editor, Jackie. She subjected my initial drafts to scrupulous tests for lucidity and sanity (shocking how often they failed). This book is much improved from her patient and valuable input. She is an all-around good egg who remained cheerful even when faced with reading and rereading several rewrites.

Chapter 1. Introduction

Internet email history goes back as far as the early 1970s, when the first messages were sent across the Arpanet, the predecessor of today's Internet. Since that time, email has been, and continues to be, the most widely used application on the Internet. In the olden days, email delivery was relatively simple, and generally consisted of moving mail files from one large host to another large host that served many users. As the Internet evolved and the network itself became more complex, more flexible tools were needed to move mail between different networks and different types of networks. The Sendmail package, released in the early 1980s, was designed to deal with the many variations among mail systems. It quickly assumed a dominant role for mail delivery on the Internet.

Today, most Internet sites use the SMTP mail protocol to deliver and receive mail messages. Sendmail is still one of the most widely deployed SMTP servers, but there have been problems with it. Sendmail's monolithic architecture has been the primary cause of numerous security issues, and it can be difficult to configure and maintain.

Postfix was originally conceived as a replacement for the pervasive Sendmail. Its design eliminates many opportunities for security problems. Postfix also eliminates much of the complexity that comes with managing a Sendmail installation. Postfix administration is managed with two straightforward configuration files, and Postfix has been designed from the beginning to handle unexpected hardware or software problems gracefully.

1.1 Postfix Origins and Philosophy

Postfix was written by Wietse Venema, who is widely known for his security tools and papers. It was made available as open source software in December 1998. IBM Research sponsored the initial release and has continued to support its ongoing development. (IBM calls the package Secure Mailer.) There were certain goals from the beginning that drove the design and development of Postfix:

Reliability

Postfix shows its real value when operating under stressful conditions. Even within simple environments, software can encounter unexpected conditions. For example, many software systems behave unpredictably when they run out of memory or disk space. Postfix detects such conditions, and rather than make the problem worse, gives the system a chance to recover. Regardless of hazards thrown its way, Postfix takes every precaution to function in a stable and reliable way.

Security

Postfix assumes it is running in a hostile environment. It employs multiple layers of defense to protect against attackers. The security concept of least privilege is employed throughout the Postfix system, so that each process, which can be run within an isolated compartment, runs with the lowest set of privileges it needs. Processes running with higher privileges never trust the unprivileged processes. Likewise, unneeded modules can be disabled, enhancing security and simplifying an installation.

Performance

Postfix was written with performance in mind and, in fact, takes steps to ensure that its speed doesn't overwhelm other systems. It uses techniques to limit both the number of new processes that have to be created and the number of filesystem accesses required in processing messages.

Flexibility

The Postfix system is actually made up of several different programs and subsystems. This approach allows for great flexibility. All of the pieces are easily tunable through straightforward configuration files.

Ease-of-use

Postfix is one of the easier email packages to set up and administer, as it uses straightforward configuration files and simple lookup tables for address translations and forwarding. The idea behind Postfix's configuration is the notion of least surprise, which means that, to the extent it's possible, Postfix behaves the way most people expect. When faced with design choices, Dr. Venema has opted for the decision that seems most reasonable to most humans.

Compatibility with Sendmail

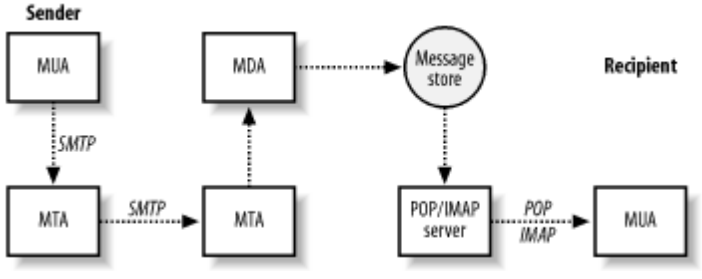
1.2 Email and the Internet

Unlike most proprietary email solutions, where a single software package does everything, Internet email is built from several standards and protocols that define how messages are composed and transferred from a sender to a recipient. There are many different pieces of software involved, each one handling a different step in message delivery. Postfix handles only a portion of the whole process. Most email users are only familiar with the software they use for reading and composing messages, known as a *mail user agent* (MUA). Examples of some common MUAs include mutt, elm, Pine, Netscape Communicator, and Outlook Express. MUAs are good for reading and composing email messages, but they don't do much for mail delivery. That's where Postfix fits in.

1.2.1 Email Components

When you tell your MUA to send a message, it simply hands off the message to a mail server running a *mail transfer agent* (MTA). Figure 1-1 shows the components involved in a simple email transmission from sender to recipient. MTAs (like Postfix) do the bulk of the work in getting a message delivered from one system to another. When it receives a request to accept an email message, the MTA determines if it should take the message or not. An MTA generally accepts messages for its own local users; for other systems it knows how to forward to; or for messages from users, systems, or networks that are allowed to relay mail to other destinations. Once the MTA accepts a message, it has to decide what to do with it next. It might deliver the message to a user on its system, or it might have to pass the message along to another MTA. Messages bound for other networks will likely pass through many systems. If the MTA cannot deliver the message or pass it along, it bounces the message back to the original sender or notifies a system administrator. MTA servers are usually managed by Internet Service Providers (ISPs) for individuals or by corporate Information Systems departments for company employees.

Figure 1-1. Simple Internet message flow



Ultimately, a message arrives at the MTA that is the final destination. If the message is destined for a user on the system, the MTA passes it to a *message delivery agent* (MDA) for the final delivery. The MDA might store the message as a plain file or pass it along to a specialized database for email. The term *message store* applies to persistent message storage regardless of how or where it is kept.

Once the message has been placed in the message store, it stays there until the intended recipient is ready to pick it up. The recipient uses an MUA to retrieve the message and read it. The MUA contacts the server that provides access to the message store. This server is separate from the MTA that delivered the message and is designed specifically to provide access for retrieving messages. After the server successfully authenticates the requester, it can transfer that user's messages to her MUA.

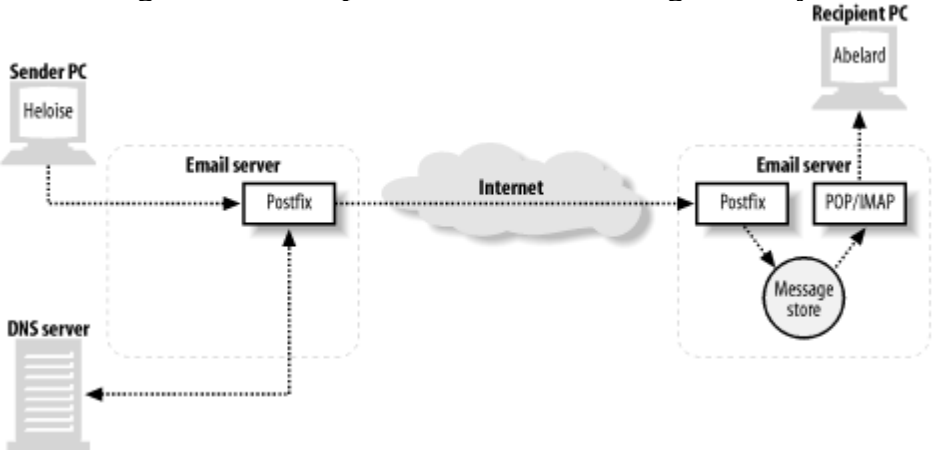
Because Internet email standards are open, there are many different software packages available to handle Internet email. Different packages that implement the same protocols can interoperate regardless of who wrote them or the type of system they are running on. If you are putting together a complete email system, most likely the software that handles SMTP will be a different package than the software that handles POP/IMAP, and there are many different software choices for each aspect of your complete email system.

1.3 The Role of Postfix

Postfix is an MTA and handles the delivery of messages between servers and locally within a system. It does not handle any POP or IMAP communications.

Figure 1-2 illustrates a simple example of message transmission where Postfix handles the responsibilities of the MTA and local delivery. As the MTA, Postfix receives and delivers email messages over the network via the SMTP protocol. For local delivery, the Postfix local delivery agent can deposit messages directly to a message store or hand off a message to a specialized mail delivery agent.

Figure 1-2. Example network email message delivery



This example shows Postfix as the SMTP server at both ends of the email transaction; however, since Postfix is based on Internet standards, the other email server in this example could easily be any other standards-compliant server. Postfix can communicate with any other server that speaks SMTP (and even some that are not quite fluent). In our example, Heloise wants to send a message to Abelard from her address (heloise@oreilly.com) to his address (abelard@postfix.org.) Heloise uses her email client to compose her message, which passes it to her MTA (using SMTP). As it happens, her MTA is a Postfix server that allows her to relay messages. After accepting the message from Heloise's email client, the Postfix server determines where Heloise's message needs to go, based on Abelard's email address. Using DNS (see Chapter 6 for more information on DNS and email) it figures out which SMTP server should accept messages for Abelard's domain (postfix.org) and contacts that server (using SMTP). Abelard's Postfix server accepts the message and stores it until Abelard is ready to pick it up. At this point Postfix's job is done. When Abelard is ready to retrieve his messages, his email client, using POP or IMAP, picks up Heloise's message.

This example leaves out the details of the complicated tasks involved when Postfix delivers mail. In the case of messages with multiple recipients, Postfix has to figure out where to deliver copies for each recipient. In case one or more recipients cannot receive mail due to a networking or systems problem, Postfix has to queue the message and retry delivery periodically. From a user's point of view, the Postfix piece of the operation is nearly invisible. From the Internet mail system's point of view, Postfix handles most aspects of email message delivery.

1.4 Postfix Security

Email systems are necessarily exposed to possible attacks because their function requires that they accept data from untrusted systems. The challenge is to build systems that are resistant to attack, and any good security strategy includes multiple layers of protection. This is particularly true for public systems in a potentially hostile environment. Postfix takes a proactive and multilayered approach to security. The Postfix architecture limits the severity of vulnerabilities, even if there are design or coding errors that might otherwise create major vulnerabilities in a monolithic privileged program.

1.4.1 Modular Design

The modular architecture of Postfix forms the basis for much of its security. Each Postfix process runs with the least amount of privilege necessary to get its particular job done. Many of Sendmail's security problems were exacerbated because Sendmail ran as a privileged process most of the time. Postfix operates with the minimum privilege necessary to accomplish a particular task. Postfix processes that are not needed on a system can be turned off, making it impossible to exploit them. For example, a network firewall system that only relays mail and does not need local delivery can have all the Postfix components for local delivery turned off. Postfix processes are insulated from each other and depend very little on any interprocess communication. Each process determines for itself what it needs to know.

1.4.2 Shells and Processes

In most cases, the delivery of mail does not require a Unix shell process, but when a configuration does make use of one, Postfix sanitizes information before placing it into environment variables. Postfix tries to eliminate any harmful characters that might have special meaning to a shell before making any data available to the shell.

Most Postfix processes are executed by a trusted master daemon. They do not run as user child processes, so they are immune to any of the security problems that rely on parent-child inheritance and communications. These attacks that use signals, shared memory, open files, and other types of interprocess communication are essentially useless against Postfix.

1.4.3 Security by Design

A buffer overflow is another common type of attack against applications. In this type of attack, crackers cause a program to write to memory where it is not supposed to. Doing so might allow them to change the path of execution in order to take control of the process. I've already mentioned that Postfix processes run with as little privilege as possible, so such an attack would not get very far; moreover, Postfix avoids using fixed-size buffers for dynamic data, making a successful buffer overflow attack highly unlikely.

An important security protection available on Unix systems is the ability to *chroot* applications. A chroot establishes a new root directory for a running application such as /var/spool/postfix. When that program runs, its view of the filesystem is limited to the subtree below /var/spool/postfix, and it cannot see anything else above that point. Your critical system directories and any other programs that might be exploited during an attack are not accessible. Postfix makes it very simple to cause its processes to run within a chroot (see more about chrooting in Chapter 4). By doing so, you cause Postfix to run in its own separate compartment. Even if Postfix is somehow subverted, it will not provide access to many of the methods an attacker typically employs to compromise a system.

1.5 Additional Information and How to Obtain Postfix

You can get more information about Postfix at the official web site: The Postfix Home Page (http://www.postfix.org/). The site contains the source code, documentation, links to add-on software, articles, and additional information about Postfix. There is also information about joining an active mailing list that discusses all aspects of Postfix.

If you don't have a copy of Postfix already, you can obtain the source code from the Postfix web site. It is, however, quite possible that there is a precompiled package for your platform that may be more convenient for you. If that is the case, you can obtain the Postfix package for your operating system and use your system's normal tools for software installation and configuration. You should check the normal repositories you use to get software for your system.

There are many good reasons to build Postfix for yourself: there may not be a pre-packaged bundle for your platform, you might not trust the packager of the bundle to have done everything correctly for your environment, you might need support for add-ons that are not built into a package, you might need a more current version than is available in packages, or you might just enjoy the task. If you have any experience compiling software, you'll have no trouble building Postfix. It's one of the easier open source packages to compile.

The Postfix web site has a download link that displays a list of mirrors from which you can get the software. You should select the mirror that is closest to you. Postfix is available as either an Official Release package or as an Experimental Release package. Even though it's called experimental, you should consider the code to be very stable. Experimental releases contain new features that might still change before they become official. Some new features are available only in an experimental release, but you should feel comfortable using them. Just be aware that they may evolve slightly in later releases until their feature sets are considered stable enough for the official release. No Postfix software is released that hasn't gone through extensive testing and review. Read through the *RELEASE_NOTES* file that comes with the package to learn what the differences are between the current official and experimental releases.

Chapter 2. Prerequisites

This chapter presents some basic Unix and email concepts that you need in order to follow explanations and examples presented later in the book. If you are already familiar with email administration, you can safely skip the material here and move on to the next chapter. This chapter does not give a systematic or comprehensive overview of either email or Unix administration. There is already an enormous amount of information available on both topics. This chapter simply presents an assortment of items that are referred to later in the book, with the expectation that readers already understand them.

2.1 Unix Topics

There's no question that the more familiar you are with Unix, the better a Postfix administrator you'll be. Postfix is very much a Unix program working in conjunction with the underlying operating system for many of its functions. If you're new to Unix, you should study an introductory text. In the meantime, this section presents some fundamental concepts that you will need to understand to follow explanations in the book.

2.1.1 Login Names and UID Numbers

The list of recognized users on a system is stored in the */etc/passwd* file. Every user should have a unique login name and user ID number (commonly written as uid or UID). The UID, not the user's login name, is the important attribute for identity and ownership checks. The login name is a convenience for humans, and the system uses it primarily to determine what the UID is. Some Postfix configuration parameters require UIDs rather than login names when referring to user accounts. Postfix sometimes takes on the identify of different users. A process is said to be using the rights or privileges of that account when assuming its identity.

2.1.2 Pseudo-Accounts

A pseudo-account is a normal Unix system account except that it does not permit logins. These accounts are used to perform administrative functions or to run programs under specific user privileges. Your system most likely came installed with several pseudo-accounts. Account names such as bin and daemon are common ones. Generally, these accounts prevent logins by using an invalid password and nonexistent home directories and login shells. For Postfix administration, you need at least one pseudo-account for Postfix processes to run under. You may need additional ones for other functions, such as mailing-list programs and filters.

2.1.3 Standard Input/Standard Output

Nearly all processes on a Unix system have a standard input stream and a standard output stream when they start. They read data on their standard input and write data on their standard output. Normally, standard input is the keyboard and standard output is the monitor, which is how users interact with running programs. Standard input and output can be redirected so that programs can get input from, and send output to, a file or another program. This is often how batch mode programs operate. For the purpose of email, you should be aware of standard input and output because your mail system may have to interact with other programs over their standard inputs and outputs. A mail filter program, for example, might accept the contents of an email message on its standard input and send the revised contents to its standard output. Programs usually also have a standard error stream that, like standard output, is normally a user's monitor, but it can also be redirected. Standard input/output/error are often written as stdin, stdout, and stderr. For more information, consult an introductory book on Unix.

2.1.4 The Superuser

The administrative login on Unix systems is the root account. It is also referred to as the superuser account, and you should treat it carefully. You should log in as the root user only when its privileges are required to accomplish a particular task. Administering Postfix sometimes requires root privileges. If you do not have superuser access on your system, you cannot administer Postfix.

2.1.5 Command Prompts

When working with an interactive shell, you are normally greeted with a command prompt that indicates the system is

2.2 Email Topics

Internet email is a complex subject with many aspects. There are important principles that apply when administering an email system regardless of the MTA you are working with. This section presents a few concepts that will help in understanding later explanations in the book, but you are urged to learn as much about Internet email as possible from the many resources available in books and online.

2.2.1 RFCs

RFCs, or Request for Comments documents, define the standards for the Internet. There are several RFCs relating to Internet email, all of which are relevant to you if you are administering an email system on the Internet. The two most commonly referenced RFCs for email are RFC 821 and RFC 822, which deal with how email messages are transferred between systems, and how email messages should appear. These documents were put into effect more than 20 years ago. They were updated in April 2001 with the proposed standards RFC 2821 and RFC 2822, although you will still see many references to the original documents. RFC documents are maintained by the Internet Engineering Task Force, whose site is available at http://www.ietf.org/.

2.2.2 Email Agents

Chapter 1 introduced several of the email agents involved in message composition to final delivery. For convenience, Table 2-1 contains a summary of these agents.

Table 2-1. Email agents

Agent	Name	Purpose
MUA	Mail User Agent	Email client software used to compose, send, and retrieve email messages. Sends messages through an MTA. Retrieves messages from a mail store either directly or through a POP/IMAP server.
MTA	Mail Transfer Agent	Server that receives and delivers email. Determines message routing and possible address rewriting. Locally delivered messages are handed off to an MDA for final delivery.
MDA	Mail Delivery Agent	Program that handles final delivery of messages for a system's local recipients. MDAs can often filter or categorize messages upon delivery. An MDA might also determine that a message must be forwarded to

Chapter 3. Postfix Architecture

You can easily manage and operate Postfix without understanding everything about how it works. If you're ready to dive right in, you can skip this section and start at the beginning of the next chapter. It might be difficult to digest all of the material here if you don't have much experience with Postfix yet, but this chapter will give you an overview of the various pieces, which might come in handy as you start to work with Postfix. Later, after you have more experience with Postfix, you might want to return to this chapter to try to absorb more of the details.

3.1 Postfix Components

The architecture of Postfix is quite different from that of a monolithic system such as Sendmail, which traditionally uses a single large program for its handling of email messages. Postfix breaks down tasks into separate functions using individual programs that each perform one specific task. Most of these programs are daemons, which are processes that run in the background on your system. The master daemon is started first, and it invokes most other processes, as needed. Postfix daemons that are invoked by the master daemon process their assigned tasks and terminate. They might also terminate after a configured amount of time or after handling a maximum number of requests. The master daemon is resident at all times, and gets its configuration information at startup from both *main.cf* and *master.cf*. See Chapter 4 for more information on Postfix configuration files.

Figure 3-1 depicts a high-level picture of the Postfix architecture. Broadly speaking, Postfix receives messages, queues them, and finally delivers them. Each stage of processing is handled by a distinct set of Postfix components. After a message is received and placed into the queue, the queue manager invokes the appropriate delivery agent for the final disposition of the message. The next few sections in this chapter discuss the details of each of the stages.

Figure 3-1. Broad view of the Postfix architecture

3.2 How Messages Enter the Postfix System

Messages come into Postfix in one of four ways:

1. A message can be accepted into Postfix locally (sent from a user on the same machine).
2. A message can be accepted into Postfix over the network.
3. A message that was already accepted into Postfix through one of the other methods is resubmitted for forwarding to another address.
4. Postfix generates messages itself when it has to send notifications of undeliverable or deferred delivery attempts.

There is always the possibility that a message is rejected before it enters the Postfix system, or that some messages are deferred for later delivery.

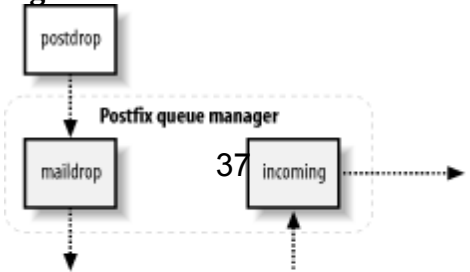
3.2.1 Local Email Submission

The various Postfix components work together by writing messages to and reading messages from the queue. The queue manager has the responsibility of managing messages in the queue and alerting the correct component when it has a job to do.

Figure 3-2 illustrates the flow when a local email message enters the Postfix system. Local messages are deposited into the *maildrop* directory of the Postfix queue by the *postdrop* command, usually through the *sendmail* compatibility program. The pickup daemon reads the message from the queue and feeds it to the cleanup daemon. Some messages arrive without all of the required information for a valid email message. So in addition to sanity checks on the message, the cleanup daemon, in conjunction with the trivial-rewrite daemon inserts missing message headers, converts addresses to the user@domain.tld format expected by other Postfix programs, and possibly translates addresses based on the canonical or virtual lookup tables (see Chapter 4 for more information on lookup tables).

The cleanup daemon processes all inbound mail and notifies the queue manager after it has placed the cleaned-up message into the incoming queue. The queue manager then invokes the appropriate delivery agent to send the message to its next hop or ultimate destination.

Figure 3-2. Local email submission



3.3 The Postfix Queue

The Postfix queue manager does the bulk of the work in processing email. Postfix components that accept mail have the ultimate goal of getting the email message to the queue manager. This is done through the cleanup daemon, which notifies the queue manager when it has placed a new message into the incoming mail queue. Once the queue manager has a new message, it uses trivial-rewrite to determine the routing information: the transport method to use, the next host for delivery, and the recipient's address.

The queue manager maintains four different queues: incoming, active, deferred, and corrupt. After the initial cleanup steps, the incoming queue is the first stop for new messages. Assuming system resources are available, the queue manager then moves the message into the active queue, and calls on one of the delivery agents to deliver it. Messages that cannot be delivered are moved into the deferred queue.

The queue manager also has the responsibility of working with the bounce and defer daemons to generate delivery status reports for problem messages to be sent back to the sender, or possibly the system administrator, or both. In addition to the message queue directories, the Postfix spool directory contains *bounce* and *defer* directories. These directories contain status information about why a particular message is delayed or undeliverable. The bounce and defer daemons use the information stored in these directories to generate their notifications. See Chapter 5 for more detailed information on how the queue manager works.

3.4 Mail Delivery

Postfix uses the concept of address *classes* when determining which destinations to accept for delivery and how the delivery takes place. The main address classes are local, virtual alias, virtual mailbox, and relay. Destination addresses that do not fall into one of these classes are delivered over the network by the SMTP client (assuming it was received by an authorized client). Depending on the address class, the queue manager calls the appropriate delivery agent to handle the message.

3.4.1 Local Delivery

The local delivery agent handles mail for users with a shell account on the system where Postfix is running. Domain names for local delivery are listed in the mydestination parameter. Messages sent to a user at any of the mydestination domains are delivered to the individual shell account for the user. In the simple case, the local delivery agent deposits an email message into the local message store. It also checks aliases and users' *.forward* files to see if local messages should be delivered elsewhere. See Chapter 7 for more information on local delivery.

When a message is to be forwarded elsewhere, it is resubmitted to Postfix for delivery to the new address. If there are temporary problems delivering the message, the delivery agent notifies the queue manager to mark the message for a future delivery attempt and store it in the deferred queue. Permanent problems cause the queue manager to bounce the message back to the original sender.

3.4.2 Virtual Alias Messages

Virtual alias addresses are all forwarded to other addresses. Domain names for virtual aliasing are listed in the virtual_alias_domains parameter. Every domain has its own set of users that do not have to be unique across domains. Users and their real addresses are listed in lookup tables specified in the virtual_alias_maps parameter. Messages received for virtual alias addresses are resubmitted for delivery to the real address. See Chapter 8 for more information on virtual aliases.

3.4.3 Virtual Mailbox Messages

The virtual delivery agent handles mail for virtual mailbox addresses. These mailboxes are not associated with particular shell accounts on the system. Domain names for virtual mailboxes are listed in the virtual_mailbox_domains parameter. Every domain has its own set of users that do not have to be unique across domains. Users and their mailbox files are listed in lookup tables specified in the virtual_mailbox_maps parameter. See Chapter 8 for more information on virtual mailboxes.

3.4.4 Relay Messages

The smtp delivery agent handles mail for relay domains. Email addresses in relay domains are hosted on other systems, but Postfix accepts messages for the domains and relays them to the correct system. Relay configurations are common when Postfix accepts mail over the Internet and passes it to systems on an internal network. Domain names for relay domains are listed in the relay_domains parameter. See Chapter 9 for more information on relaying.

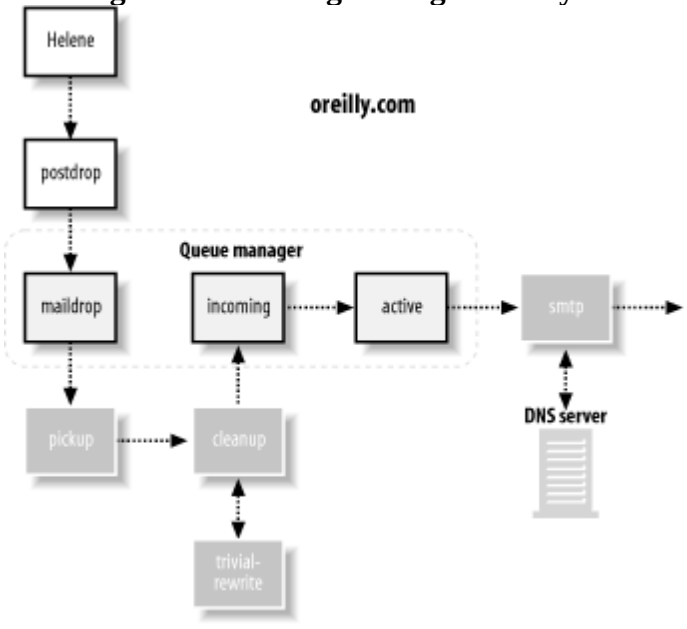
3.4.5 Other Messages

Messages that do not fit into one of the address classes are generally destined for other domains hosted elsewhere on

3.5 Tracing a Message Through Postfix

Let's follow a typical message through the Postfix system. Figure 3-4, Figure 3-5, and Figure 3-6 illustrate the process as the message goes from the originating system to a destination MTA, which, in turn, forwards it to the final MTA, where it is held until the user is ready to read it. In Figure 3-4, Helene (helene@oreilly.com) wants to send a message to Frank (frank@postfix.org). Helene has an account on a system that runs Postfix. Her email client lets her compose the message, and then it calls the Postfix *sendmail* command to send it. The Postfix *sendmail* command receives the message from Helene's email software and deposits it into the *maildrop* directory. The pickup daemon then retrieves the message, performs its sanity checks, and feeds the message to the cleanup daemon, which performs the final processing on the new message. If Helene's email client did not include a From: address, or did not use a fully-qualified hostname in the address, cleanup makes the necessary fixes to the message.

Figure 3-4. Tracing message delivery 1

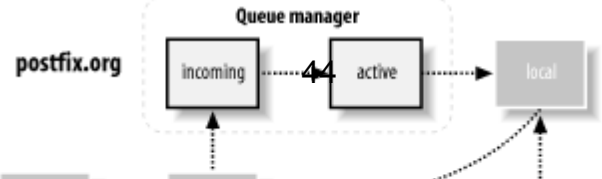


Once finished, cleanup places the message into the incoming queue and notifies the queue manager that a new message is ready to be delivered. If the queue manager is ready to process new messages, it moves the message into the active queue. Because this message is destined for a user on an outside system, the queue manager has to alert the smtp agent to handle the delivery of the message.

The smtp agent uses DNS (see Chapter 6) to get a list of email systems that can accept mail for the domain postfix.org. The smtp delivery agent selects the most preferred MX host from the list and contacts it to deliver Helene's message.

Figure 3-5 shows Frank's email server at postfix.org also running Postfix, although the system could be using any other standards-compliant MTA. The Postfix smtpd on Frank's server takes the message from Helene's smtp delivery agent. After the smtpd daemon verifies that it should, in fact, accept this message, it passes the message through to the cleanup daemon, which performs its checks before depositing the message into the incoming queue.

Figure 3-5. Tracing message delivery 2



Chapter 4. General Configuration and Administration

One of the truly remarkable things about Postfix is that, in many cases, it works as soon as you install it, with little or no change to its configuration. In the first section of this chapter, we'll walk through checking the configuration and starting Postfix for the first time. Later sections discuss Postfix configuration details.

By default, Postfix is configured as a traditional Unix mail server, sending and receiving messages for all the accounts on the system. Your users can send and receive messages using any email client software available on your system.

In most environments, Postfix works in conjunction with a variety of other software systems. You should build each piece of your email system and test each one as a separate module before trying to integrate them all together. As you add each module, test the system before moving on to the next piece.

At this point you should have Postfix installed on your system. You might install Postfix from a packaged bundle for your platform or compile it yourself. See Appendix C for help with compiling Postfix, if you're building it yourself. Check your normal software sources for any Postfix packages that might be available. If you haven't yet installed Postfix, either get a package for your system or follow the instructions in Appendix C to build it. When you have finished with the installation, come back to this chapter for the final configuration.

I will assume, in examples throughout the book, that your installation of Postfix uses the default directories:

/etc/postfix

Configuration files and lookup tables

/usr/libexec/postfix

Postfix daemons

/var/spool/postfix

Queue files

/usr/sbin

Postfix commands

I will also assume that you or your installer created a postfix user and postdrop group. This user and group should

4.1 Starting Postfix the First Time

There are two important issues to deal with before starting Postfix for the first time. The first is how your system identifies itself. Postfix uses a configuration parameter called myhostname, which must be set to the fully qualified hostname of the system Postfix is running on. Once Postfix knows the fully qualified hostname, it can use that hostname to set default values for other important parameters, such as mydomain. If the parameter myhostname is not set, Postfix defaults to the hostname reported by the system itself. There is a complete discussion of myhostname later in the chapter. You can see what name your system reports with the Unix *hostname* command:

```
$ hostname
```

4.2 Configuration Files

The directory */etc/postfix* contains Postfix configuration files. The two most important files used in the configuration of Postfix are *master.cf* and *main.cf*. These files should be owned by, and only writable by, the root user. They should be readable by everyone. Whenever you make changes to these files, you have to reload Postfix for your changes to go into effect:[1]

[1] If you change the inet_interfaces parameter, you must stop and start Postfix.

```
# postfix reload
```

The *master* daemon is the overall process that controls other Postfix daemons for mail handling. The *master* daemon uses the *master.cf* file for its configuration information. The *master.cf* file contains a line for each Postfix service or transport. Each line has columns that specify how each program should run as part of the overall Postfix system. See Chapter 3 for information on Postfix's architecture and how various components interact with each other. In many installations, you will never have to change the default *master.cf* file. See Section 4.5 later in the chapter for information on when and how to make changes to *master.cf*.

4.2.1 The main.cf Configuration File

The *main.cf* file is the core of your Postfix configuration. Nearly all configuration changes occur in this file. The default *main.cf* file lists only a portion of the nearly 300 Postfix parameters. Most Postfix parameters do not need to be changed, but the flexibility is there when it's required. All Postfix parameters are listed and described in the various sample configuration files. The sample files are located in the directory specified by the sample_directory parameter, which is usually the same directory as your *main.cf* file. Both the *main.cf* file and the sample files that come with the Postfix distribution contain comments that explain each of the parameters.

Throughout this book, when the text says to modify a parameter, it always refers to a parameter in your *main.cf* unless a different file is indicated.

You can edit *main.cf* with the *postconf* command, as you saw earlier in the chapter, or you can change the file directly with any text editor[2] (such as *vi* or *emacs*). The file contains blank lines, comment lines, and lines that assign values to parameters. Comment lines start with the # character and continue to the end of the line. Blank and comment lines are ignored by Postfix. Parameters can appear in any order within the file, and are written as you would expect:

[2] Postfix expects configuration files to contain normal Unix-style line endings. If you edit your configuration files from another platform, such as Windows or Mac, make sure that your editor uses the correct line endings for Unix.

```
parameter = value
```

A parameter definition must start in the first column of the line. The spaces around the equals sign are optional.

Here is an example parameter assignment with a comment:

```
# The myhostname value must be a fully qualified hostname.
```

4.3 Important Configuration Considerations

We saw at the beginning of this chapter how Postfix requires only minimal configuration changes to work. Depending on how you plan to use your Postfix system, you may want to consider some of the more common options. This section discusses how your system identifies itself, and then covers the very important topic of relay control.

4.3.1 Configuring Your MTA Identity

There are four parameters dealing with your system's hostname and domain that you want to consider, no matter how you use Postfix: myhostname, mydomain, myorigin, and mydestination.

4.3.1.1 myhostname and mydomain

We discussed the purpose and importance of the myhostname parameter earlier in this chapter. If myhostname is not specified, Postfix uses the function *gethostname* to determine what your system's hostname is. If your system correctly reports the fully qualified hostname, you can leave myhostname unspecified in the configuration file. Some systems may not be configured correctly or may not report the fully qualified version of the hostname. In these cases, you can set either myhostname to the fully qualified hostname or mydomain to your system's domain. If mydomain is explicitly set, Postfix automatically sets myhostname to the domain name specified and the local hostname reported by *gethostname* to create the fully qualified hostname.

If you set myhostname to the system's fully qualified hostname but omit mydomain, Postfix uses the value of myhostname, minus the first component of the fully qualified hostname, to automatically set mydomain. A value of mail.example.com for myhostname causes mydomain to be example.com unless you explicitly set it to something else. Similarly, a hostname of mail.ny.example.com causes the value to be ny.example.com. If your system does not report its fully qualified name, and you have not set either the mydomain or myhostname parameters, Postfix reports the problem in your log file. See Section 4.4.1 later in this chapter.

4.3.1.2 myorigin

When your users send or receive mail through the Postfix system with no domain name specified in the envelope or header addresses, the parameter myorigin determines what domain name should be appended. The default is to use the value of myhostname. If Postfix is running on a system whose hostname is mail.example.com, messages from the user kdent have a From: address of kdent@mail.example.com. However, frequently users want their mail to be sent from the domain name without any extra host information (kdent@example.com instead of kdent@mail.example.com). If that is the case, set myorigin to $mydomain:

```
myorigin = $mydomain
```

4.3.1.3 mydestination

The mydestination parameter lists all the domains your Postfix system should accept mail for and deliver to local users. By default Postfix accepts mail destined for $myhostname and localhost.$mydomain. If you want your system to accept mail for your entire domain and not just the single host it is running on, add $mydomain to the list:

```
mydestination = $myhostname, localhost.$mydomain, $mydomain
```

Now your mail server can act as a gateway receiving all mail for the domain.

4.3.2 Relay Control

4.4 Administration

Running a mail server is an ongoing task. You cannot start it and forget about it. There are periodic administrative tasks, and you should regularly check for any problems your system might have. This section discusses many of those tasks and how to accomplish them with Postfix.

Postfix provides a utility through the *postfix* command to validate many aspects of your installation. The command checks for configuration problems, looks at directory and file ownership, and creates any missing directories. Executing:

```
# postfix check
```

should report no messages on a correctly installed system. If there are any problems, the command reports them to you both on the screen and in your log file.

4.4.1 Logging

Since Postfix is a long-running program, you should regularly check your system's log file for warnings or messages. Things can change on your system that might impact Postfix. Almost all Postfix activity, successful or not, is logged. Whenever you start or reload Postfix, it is a good idea to check your log file for messages.

Postfix logging is accomplished by using your system's *syslog* daemon. System log files are an aspect of system administration that vary across versions of Unix, so you may have to consult your own system documentation to fully understand Postfix logging.

In general, the syslog daemon (syslogd) receives messages from various system processes and writes them to their final destination (often a file). syslogd organizes messages according to their importance and the application or facility that generated the message. The file */etc/syslog.conf* tells syslogd where to write each type of message. The logging facility used by Postfix is mail. If you don't know where to find messages logged by Postfix, the file */etc/syslog.conf* should point you in the right direction. Some operating systems, by convention, log nearly everything to a single file, such as */var/log/syslog*, while others prefer to separate messages by applications or services, so that Postfix messages go to a file like */var/log/maillog*. For the latter type of systems, you might find an entry like the following in */etc/syslog.conf*:

```
mail.*          -/var/log/maillog
```

Once you locate your mail log file, check it regularly. You'll probably want to check it at least daily, but decide for yourself, depending on the volume of mail your server handles and your existing log rotation scheme. You can use the following command to find Postfix messages that might be of interest:

```
$ egrep '(reject|warning|error|fatal|panic):' /var/log/maillog
```

assuming that your log file is */var/log/maillog*. If not, substitute the name of your own mail log file.

4.4.2 Starting, Stopping, and Reloading Postfix

You saw earlier in the chapter how to use the *postfix* command to start Postfix:

```
# postfix start
```


Once Postfix is running, if you make any changes to *main.cf* or *master.cf*, have Postfix reread its configuration by executing *postfix* with the reload argument:

```
# postfix reload
```

4.5 master.cf

The Postfix *master* daemon launches all of the other Postfix services as they are needed. The various services, and how they are run, are specified in the *master.cf* file.

The master configuration file works like other Postfix configuration files. A comment is marked by a # character at the beginning of a line. Comments and blank lines are ignored. Long lines can continue onto subsequent lines by starting the carry-over lines with whitespace.

Example 4-2 shows a sample file. Each column contains a specific configuration option. A dash in a column indicates the default setting for that column. Some default values come from parameters in the *main.cf* file.

Example 4-2. Sample master.cf file

```
#==========================================================================
```

4.6 Receiving Limits

The *smtpd* daemon can enforce a number of limits on incoming mail. The limits are configurable through several parameters in the *main.cf* file. You can limit the size of messages, the number of recipients for a single delivery, and the length of lines in a message. You can also limit the number of errors to allow from a single client before breaking off communications.

To limit the number of recipients for a single message, use the smtpd_recipient_limit parameter. The default is 1,000 recipients, and it should be adequate for normal operation.

The message_size_limit parameter limits the size of any message your system will accept. The default is 10 MB. If you have limited disk space or memory, you might want to lower the value. On the other hand, if your users commonly receive large attachments, you may have to increase it.

Increasingly frequent errors from the same client might indicate a problem or an attack. Postfix keeps a counter of errors, and handles potential problem clients by introducing delays with each error. The delays can help protect your system from misconfigured or malignant clients. As the number of errors increases so does the length of each delay. The length of the initial delay is specified by smtpd_error_sleep_time with a default of one second. After the number of errors exceeds the value set for smtpd_soft_error_limit, Postfix increases the delay by one second for every error, so that with each error, there is a slightly longer delay. Finally, when the error count hits the value set in smtpd_hard_error_limit, Postfix gives up on the client and disconnects.

If a malicious program connects to your mail server and sends garbage commands, attempting to crash your server, the bogus commands appear to Postfix as errors from a misbehaving client. Assume the following values for the delay parameters:

```
smtpd_error_sleep_time = 1s
```

4.7 Rewriting Addresses

Postfix tries to make sense of addresses in email and writes them using the standard RFC 2822 format. Certain address rewriting occurs automatically.

You saw earlier in the chapter how Postfix appends myorigin to a local name that has no domain part. Postfix also appends the value of mydomain to addresses that include only the host portion without the domain name. This fixes addresses that look like kdent@host so they become kdent@host.example.com.

Turning Off Address Completion

Postfix's expansion of incomplete email addresses is sometimes the source of confusion for end users. If your system is hosting the domain example.com and receives an email message where the From: message header contains an incomplete address like:

```
From: Marketing
```

4.7.1 Canonical Addresses

Postfix provides another type of address rewriting that lets you map disparate addresses into a standard format for your entire site. The canonical_maps parameter points to a lookup table of address mappings. (While the word canonical has many meanings, among computer professionals it means "the usual, standard, or normal.") If different mail systems on your network create addresses in different ways, you can relay them all through your Postfix gateway and have it fix up the addresses into your standard format. Canonical maps are often used to change addresses from an internal format to a public one. Include entries like the following in your canonical table:

#

4.8 chroot

Postfix provides multiple layers of security. One such layer is the option to permit most Postfix services to run within a *chroot* environment. The Unix chroot function allows a process to change its view of, and access to, its filesystem by changing its root directory to a new path other than the normal /.

The chroot feature is particularly beneficial for processes that must communicate with external, potentially hostile clients. If an attacker somehow manages to subvert the *smtpd* daemon, for example, the attacker gains only very limited access to the filesystem. Configuring for a chroot environment is an advanced Postfix feature that adds a layer of complexity that you or your administrators may not want to deal with. Generally, chroot is not needed, except for sites that use Postfix in a highly secure environment or on particularly exposed servers, such as dedicated firewall systems and bastion hosts.

All of the Postfix processes that use chroot change their root directory to the directory specified in the queue_directory parameter, which is normally */var/spool/postfix*. When a process runs chrooted, the directory */var/spool/postfix/pid*, for example, becomes */pid* to that process, and the process cannot access any files other than those below its new root.

To chroot individual components, edit your *master.cf* file. Change the fifth column to y. The chroot option is possible with all components except the *pipe*, *virtual*, *local*, and *proxymap* services. In Example 4-1, chroot is enabled for the SMTP clients and server.

Since chroot changes the environment of the process, all of the resources the chrooted daemon needs must be available below the new root directory. Unfortunately, the specific resources Postfix daemons might need depend on your platform. In general, Postfix might require resources that provide user information (*/etc/passwd*), name resolution configuration (*nsswitch.conf* or *resolv.conf*), timezone information, or shared libraries. Some platforms also require certain device files. There are platform-specific scripts that come with the Postfix distribution. They're available in the *examples/chroot-setup/* subdirectory below the main distribution directory.

Executing the correct script should be sufficient to set up the chroot environment on your system. If there is not a script for your platform, you may have to experiment a little to find everything you need. Consider all of the resources mentioned above and review the example scripts for other platforms. Watch your logs for error messages after you chroot a process. An entry like the following:

```
postfix/smtp[1575]: fatal: unknown service: smtp/tcp
```

shows that Postfix cannot determine what port the smtp service uses. This problem is fixed by placing the */etc/services* file into the chroot, by copying it to */var/spool/postfix/etc/services*. Other symptoms show up in the log complaining of similar types of problems.

If the normal Postfix log doesn't give enough information, you may have to run a trace to see where the program fails. Look for tools such as *truss*, *strace*, and *tusc* on your system. These tools can be used to see where a service fails when it tries to run in a chroot. If you discover the failure is due to a missing component, copy the component into the chrooted environment. See the *DEBUG_README* file that comes with Postfix for instructions on attaching tracing tools to Postfix.

Once you have Postfix running in a chroot, you need to make sure you keep your chroot resources in sync with the normal system files. If your chroot requires */etc/passwd*, for example, whenever the system */etc/passwd* changes, the

4.9 Documentation

The Postfix distribution ships with a lot of documentation. Depending on your installation package, you may or may not have all of the documents. You should have at least the manpages and sample configuration files. The sample files are located in the directory specified by the sample_directory parameter, which is usually the same directory in which your *main.cf* file resides. All of the Postfix parameters are documented in one or more of the sample files.

When Postfix was installed, the manpages should have been installed in a sensible place on your system. If they are in a directory where your system expects to find them, you only have to type, for example:

```
$ man postfix
```

to have the manpage displayed on your screen. If your system replies with an error message such as:

```
$ man postfix
```

Chapter 5. Queue Management

The queue manager daemon *qmgr* is in many ways the heart of your Postfix system.[1] All messages, both outbound and inbound, must pass through the queue. It's a good idea to understand the queue and how Postfix uses it in case you have to troubleshoot a problem.

[1] You may see references to *nqmgr* in older configuration files and documentation. Earlier Postfix versions shipped with two queue manager daemons, *qmgr* and *nqmgr*. The original *qmgr* was replaced by the current one, which has a better scheduling algorithm. *nqmgr* was the name of the current queue manager daemon while it coexisted with the original. Once it was ready for promotion as sole queue manager for Postfix, it was renamed *qmgr*.

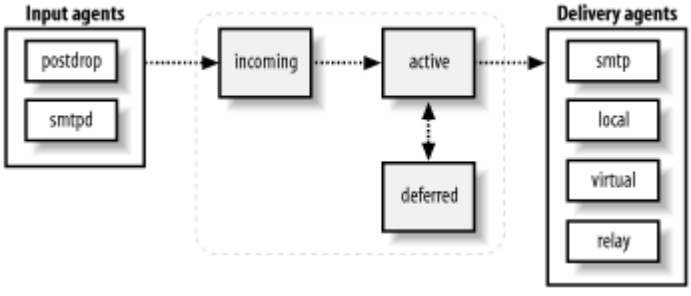
The queue manager maintains five different queues: incoming, active, deferred, hold, and corrupt. Postfix uses a separate directory for each queue below the path specified in the queue_directory parameter. By default the path is */var/spool/postfix*, which gives you a directory structure like the following:

```
/var/spool/postfix/active
```

5.1 How qmgr Works

Figure 5-1 illustrates how messages move through the queue. The incoming queue is where messages first enter Postfix. The queue manager provides protection for the queue filesystem through the queue_minfree parameter. The default value is 0. You can make sure the disk that stores your queue doesn't run out of space by setting a limit.

Figure 5-1. Message movement in the queue.



From the incoming queue, the queue manager moves messages to the active queue and invokes the appropriate delivery agent to handle them. For the most part, if there are no problems with delivery, movement through the queue is so fast that you won't see messages in the queue. If Postfix is trying to deliver to a slow or unavailable SMTP server, you may see messages in the active queue. Postfix waits 30 seconds to decide if a remote system is unreachable.

A message that cannot be delivered is placed in the deferred queue. Messages are deferred only when they encounter a temporary problem in delivery, such as a temporary DNS problem or when a destination mail server reports a temporary problem. Messages that are rejected, or encounter a permanent error, are immediately bounced back to the sender in an error report and don't stay in the queue.

5.1.1 Deferred Mail

Messages in the deferred queue stay there until they are either delivered successfully or expire and are bounced back to the sender. The bounce_size_limit parameter determines how much of a message that could not be delivered is bounced back to the sender in the error report. The default is 50,000 bytes.

Once a message has failed delivery, Postfix marks it with a timestamp to indicate when the next delivery attempt should occur. Postfix keeps a short-term list of systems that are down to avoid unnecessary delivery attempts. If there are deferred messages scheduled for a redelivery attempt, and there is space available in the active queue, the queue manager alternates between taking messages from the deferred and incoming queues, so that new messages are not forced to wait behind a large backlog of deferred ones.

5.1.2 Queue Scheduling

Postfix periodically scans the queue to see if there are deferred messages whose timestamps indicate they are ready for another delivery attempt. Subsequent failed attempts at delivery cause scheduled delays to double, so Postfix waits longer each time before it attempts to deliver a message. You can configure the maximum delay with the maximal_queue_lifetime parameter. When the time has expired, Postfix gives up trying to deliver the message and bounces it back to the sender. By default the period is five days (5d). You can set it to any length of time, or to 0 to have undeliverable mail returned immediately.

5.2 Queue Tools

Postfix provides command-line tools for displaying and managing the messages in your queue. The primary commands are *postsuper* and *postqueue*. You can perform the following tasks on messages in the queue:

- Listing messages
- Deleting messages
- Holding messages
- Requeuing messages
- Displaying messages
- Flushing messages

Each of the tasks, and the commands to accomplish them, are explained in the sections that follow.

5.2.1 Listing the Queue

The queue display contains an entry for each message that shows the message ID, size, arrival time, sender, and recipient addresses. Deferred messages also include the reason they could not be delivered. Messages in the active queue are marked with an asterisk after the Queue ID. Messages in the hold queue are marked with an exclamation point. Deferred messages have no mark.

You can list all the messages in your queue with the *postqueue -p* command. Postfix also provides the *mailq* command for compatibility with Sendmail. The Postfix replacement for *mailq* produces the same output as *postqueue -p*.

A typical queue entry looks like the following:

```
$ postqueue -p
```

Chapter 6. Email and DNS

The *Domain Name System* (DNS) is a vast distributed database whose main job is to map hostnames to IP addresses. It also has an important role in email routing. In this chapter we'll look at how MTAs in general use DNS and some of the DNS issues that relate to Postfix and its configuration. Keep in mind that there are two important aspects to your mail servers and DNS:

- For sending mail, the system running your Postfix mail server must have access to a reliable DNS server to resolve hostnames and email-routing information.
- For receiving mail, your domains must be configured correctly to route messages to your mail server.

Misconfiguration of DNS servers is a common source of problems in setting up email servers.

6.1 DNS Overview

At one time, hostname to IP address mapping was handled by one large, centrally managed text file that contained an entry for every host accessible on the Internet. Each site downloaded a copy of the file periodically to get the latest hostname information. That scheme quickly became unwieldy, and the DNS service was conceived. It was defined in RFC 882 in 1983, and introduced two key ideas: the data is distributed and the naming of hosts is hierarchical. Making the data distributed means that every site updates its own information, and the updates become available almost immediately. Hierarchical naming prevents hostname conflicts and gives us the current domain-naming system that we are all very familiar with today. Each site obtains at least one domain name, and all of the hosts at that site are named by prefixing the simple hostname to the site's domain name. For example, a site that controls the domain name example.com might have any number of hosts with names like server1.example.com, hp4100.example.com, or www.example.com.

Each domain has at least two domain nameservers that are considered *authoritative* for the domain. Authoritative nameservers should have direct access to the database that contains all the information about a domain.

The data is comprised of different types of records called *resource records*. Different resource records provide different kinds of information, such as IP addresses, nameservers, hostname aliases, and mail routing. The resource records you need to know about for this discussion are the following:

A

The mapping of names to IP addresses is handled by A records. These records contain a hostname and its IP address. The names that people use to refer to hosts have to be converted to IP addresses used for Internet routing. A records provide this name-to-address translation.

CNAME

Some hostnames are aliases that point to other hostnames, rather than to IP addresses. This can be useful for directing requests to services (such as HTTP or POP) that might reside on systems generally known by a different name. The CNAME record provides the "real," or *canonical*, name that an alias hostname points to. For example, an administrator might publicize the hostname www.example.com, which is really a CNAME record pointing to server1.example.com most of the time. But during periods of maintenance on server1.example.com, for example, www.example.com could temporarily point to server2.example.com.

MX

MX records provide mail-routing information. They specify *mail exchangers* for domains—that is, the names of the mail hubs that handle all the mail for a domain name. The MX records tell MTAs where to send messages. Since a domain can have multiple mail exchangers, MX records include a preference value to designate the order of priority when selecting a mail exchanger to deliver messages to.

PTR

6.2 Email Routing

Let's consider for a moment one way that email routing might work. A user horatio in the domain example.com has a workstation named denmark. He could receive mail by using the email address horatio@denmark.example.com. An MTA with a message to deliver would simply look up the IP address for denmark.example.com and deliver it to that system for the user horatio. This scenario requires that Horatio's workstation is always turned on, that it has a functional MTA running at all times to receive messages, and that it is accessible by unknown MTAs from anywhere on the Internet. Rather than manage hundreds or thousands of MTAs on workstations and expose them to the Internet, nearly all sites make use of mail hubs that receive all the mail for a domain. MTAs such as Postfix need a way to determine which host or hosts are the mail hubs for a domain. DNS MX records provide this information.

A mail exchanger either delivers mail it receives or forwards it to another mail system. A domain may have multiple mail systems for reliability, and therefore multiple MX records. Generally, one host is the primary mail server and the others serve as backup or secondary mail servers. Each MX record in DNS contains a preference value that orders mail systems from most preferred to least preferred.

BIND is one of the most common DNS server applications. (O'Reilly's DNS and BIND by Paul Albitz and Cricket Liu fully explains the DNS system and documents the BIND software.) A simple BIND configuration file for the domain example.com looks like the following:

```
example.com. IN SOA ns.example.com. kdent.example.com. (
```

6.3 Postfix and DNS

When sending mail, Postfix uses system *resolvers*, which are programs or libraries that make requests for DNS information. To receive mail, the DNS for your domain must be configured to route messages to your Postfix server. This section looks at DNS issues both for sending and receiving mail.

6.3.1 DNS and Sending Mail

The Postfix SMTP delivery agent must be able to obtain IP address and MX records for mail-routing information. Postfix must make at least two DNS lookups: one to get the MX hostname and one to get the IP address for that hostname. Since Postfix uses the normal operating system resolver libraries for its DNS queries, the system that runs Postfix must have access to a DNS server. The DNS server does not have to be on the same system, although for most circumstances it should be.

If your system does not seem to be resolving domain names correctly, there are three common command-line tools that you can use to troubleshoot the problem: *nslookup*, *dig*, and *host*. You should check your system documentation to see which of these tools is available on your server and how to use them. You can use these tools to query all types of resource records for a domain, including the MX record that Postfix needs in order to successfully deliver mail to a domain.

DNS problems might stem from your own system's configuration or a problem with the DNS server configuration for the domain Postfix is trying to send mail to. When you are troubleshooting a problem, it is very important to remember that Postfix first looks for MX records and not A records. Even if you can resolve a domain to an IP address, Postfix may not be able to deliver mail for that domain if there is a problem in retrieving MX information.

6.3.1.1 Configuration options

When delivering mail, Postfix performs a DNS lookup to retrieve all of the MX records for the destination domain. It sorts them in order of preference and tries each one in priority order. Once Postfix has established a connection with an SMTP server, the server replies to Postfix requests with a status code. Codes within the 2xx range indicate that everything is okay. Error codes in the 4xx range indicate a temporary problem, and those in the 5xx range indicate a permanent problem. See Chapter 2 for more information on SMTP reply codes.

To provide compatibility with Sendmail, Postfix, by default, treats SMTP servers that respond with 4xx or 5xx reply codes as if the servers had not responded at all. If you prefer that Postfix react to the error codes returned by the MX server rather than ignore them, set the smtp_skip_5xx_greeting and smtp_skip_4xx_greeting parameters:

```
smtp_skip_4xx_greeting = no
```

6.4 Common Problems

The following error messages in the mail log files indicate host lookup problems:

mail for *domain* loops back to myself

This is one of the most common errors related to DNS. It happens when you have configured your Postfix server as an MX host in your DNS server, but you have not told your Postfix server that it is the final destination for the domain. Add the domain in question to the mydestination parameter, or configure it as a virtual domain or a relay domain. If your Postfix server is behind a proxy or NAT device, it may not realize that it is an MX host for the domain. In that case, add the proxy device's IP address to proxy_interfaces. Log entries for this error resemble the following:

```
postfix/qmgr[3981]: 2CC3B229: from=<heloise@ora.com>, \
```

Chapter 7. Local Delivery and POP/IMAP

Chapter 1 explained that POP and IMAP are protocols that deal with how users retrieve their email messages from message stores. Postfix is a mail transfer agent and does not implement POP or IMAP. This chapter looks at how Postfix delivers messages and how they are read by POP/IMAP servers. There are many POP/IMAP servers available, and the information presented here should be applicable to any standards-conforming server. The last part of this chapter deals with configuring Postfix to work with the Cyrus IMAP server. Before we look at local delivery, we'll first discuss more broadly the different delivery transports Postfix uses. Transports other than local are discussed in subsequent chapters.

7.1 Postfix Delivery Transports

Postfix offers delivery for four different classes of recipient addresses: local, relay, virtual alias, and virtual mailbox. How you configure the domains you accept mail for determines the delivery method used by Postfix. The following are the delivery transports used by Postfix:

local

Delivers mail on the local system. Each address has an account on the system or comes from the local aliases file (historically */etc/aliases*). Delivered messages go to the system's mail spool or mail files in individual home directories. Deliveries are handled by the local delivery agent or passed to a custom delivery program. Lists local domains in the mydestination parameter.

relay

Delivers mail to other systems, usually on the same network. Relay domains are generally configured on gateway systems when Postfix accepts mail for an entire network. The gateway system relays messages to the correct internal mail system. Deliveries are handled by the relay transport, which is simply a clone of the smtp agent, but it is optimized for making deliveries to internal systems on a local network. Lists relay domains in the relay_domains parameter. Mail relaying is discussed in Chapter 9.

virtual

Delivers mail for virtual mailbox domains. Virtual mailbox domains are used for hosting multiple domains using a separate mail spool that contains mailboxes for many separate domains. Email users typically do not have system accounts on the mail server. Lists virtual mailbox domains in the virtual_mailbox_domains parameter. Virtual hosting is discussed in Chapter 8.

Deliveries to nonlocal domains are handled by the *smtp* transport. It determines where to deliver messages for any nonlocal domain through DNS lookups. Virtual alias addresses are resubmitted to Postfix for delivery to the new address, at which point they'll be handled by one of the above transports.

The rest of the chapter discusses the details of local delivery.

7.2 Message Store Formats

When Postfix makes local deliveries it transfers the contents of messages to the local message store. The most common types of message stores are the traditional *mbox* format and the newer *maildir* style. Both use regular files to store messages, but they are structured in different ways. In Postfix, you specify maildir style by including a trailing slash when you configure any mail file or directory parameters (see configuration information later in this chapter).

7.2.1 The Mbox Format

Historically, Unix systems have used a single file to store each user's email messages. This type of message store format is commonly referred to as *mbox*. Each message within the file starts with a line that begins with the word From. It is important that the string start on the first character of the line, and that there is a space after the end of the word. The From line is commonly referred to as From_ with an underscore character to indicate the space following the word. Don't confuse the From_ line used for separating messages within an mbox file with the From: line included in email message headers. The last line of a message is always a blank line.

A complete From_ line looks like the following:

```
From jmbrown@example.com Sun Feb  3 16:54:01 2002
```

As described, the line starts with the word From followed by a space. Following the space is an email address that is usually the envelope address of the message. Following the envelope address is the date of delivery in the common Unix date format occupying 24 characters. The mbox format allows for an optional comment string following the date, but it is generally not used.

When Postfix delivers a message to an mbox file, it first creates the From_ line using the envelope sender and the current date. Postfix then copies the contents of the delivered message into the mbox file. If Postfix encounters any lines that begin with From followed by a space, it has to quote them by adding a > to the beginning of the line, so that they won't be confused with the start of the next message.

When a POP/IMAP server reads messages from the mbox file, it scans the file, looking for From_ lines, which mark the beginning of each message. It can read to the next From_ line (or the end of the file) to know when a message is finished. The POP/IMAP server may unquote any of the ">From" quoted lines, or they may remain in the quoted form.

Since both Postfix and the POP/IMAP servers access the mailbox file, they must use file locking. Postfix must obtain an exclusive lock on the file when it is delivering a message, so that it can write the message to the file. Postfix offers a variety of locking mechanisms, depending on the platform. You can use the *postconf -l* command to see which mechanisms Postfix can use on your system:

```
$ postconf -l
```

7.3 Local Delivery

All destination domains that should be handled by the local transport should be listed in the mydestination parameter. You can list as many domains as you like, but individual local users receive mail at all of the domains listed. For example, if both ora.com and oreilly.com are listed in mydestination, then messages to either kdent@ora.com or kdent@oreilly.com go to the same local mailbox.

All local recipients should be listed in tables configured in the local_recipient_maps parameter to avoid accepting messages for unknown users. By default, local_recipient_maps is set to the system password file and alias maps, so you normally don't have to make any changes. Once Postfix has determined that it is the final destination for a message, and that the message should be delivered locally, it has to decide what to do with the message.

Before looking for a user account that matches the local part of the email address, Postfix consults its alias maps (see Chapter 4). If there is a forwarding alias that matches the recipient address, Postfix resubmits the message as a new delivery, based on the forwarding information from the alias lookup. Otherwise, it tries to deliver the message to a user on the system. Postfix first checks for the existence of a *.forward* file for the local user, and may resubmit the message based on information there. If no *.forward* exists for the user, Postfix delivers the message to the user's mailbox.

7.3.1 .forward Files

.forward files allow local users to set up their own aliases. The contents of the *.forward* file are the same as the righthand side of an alias entry. When an alias entry has multiple values on the righthand side, they are separated by commas; while *.forward* files use the same convention, they also allow multiple entries to be entered on multiple lines.

.forward files must be owned by the recipient, and are normally found in users' home directories. You can specify different locations with the forward_path parameter. When specifying a path for the parameter, there are eight variables whose values are expanded at delivery time:

$user

Recipient username as specified in */etc/passwd*

$home

Recipient home directory as specified in */etc/passwd*

$shell

Recipient shell as specified in */etc/passwd*

$recipient

7.4 POP and IMAP

After Postfix has delivered a message, users need a way to read it. Many sites provide a POP/IMAP server for users to retrieve their email messages over the network. In most cases Postfix works seamlessly with POP/IMAP servers, so that no special configuration is required on either side.

7.4.1 POP Versus IMAP

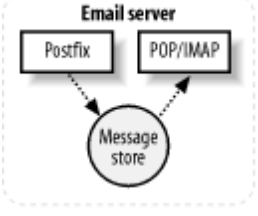
The POP protocol works best when you have limited, or less than full-time, network access because it allows you to connect to your mail server, fetch all of your messages, and disconnect from the network. You now have local copies that you can read offline. Most POP clients have a configuration option to delete your messages from the server when you retrieve them, since you then have the local copies. If you don't delete them at some time, the messages accumulate, taking up more and more space on your mail server. POP was designed to be easy to implement, but the major problem with the POP protocol is that if you ever work from more than one computer, your messages may not be where you need them. It also does not handle multiple mailboxes very well, and it forces you to download complete messages. There is no option to retrieve just the subject, for example, to decide if you want the complete message.

The IMAP protocol was designed to overcome some of POP's shortcomings. It keeps all messages on the server. You have to be connected while working with your email messages, but you can manage them as if they were local. Since everything happens on the server, it doesn't matter if you work from your desktop computer at home, another machine at work, and even on a laptop while traveling. IMAP still allows for saving messages locally, if necessary, and it also provides much more flexibility than POP. You can download just the headers from your messages and then decide to retrieve the rest of a message if you want to read it. You don't have to be stuck downloading a huge message or attachment that you might not be interested in. You can maintain multiple mailboxes and folders on the IMAP server.

7.4.2 Postfix and POP/IMAP Servers

The cooperation between Postfix and POP/IMAP servers is simple. When Postfix accepts delivery of an email message, it places it in the message store. The POP/IMAP server simply retrieves messages from the same store when a user requests them. Figure 7-1 shows how simple the cooperation is between Postfix and POP/IMAP servers. Postfix and the POP/IMAP server must agree on the type of mailbox format and the style of locking. Postfix should work with any standards-compliant POP/IMAP server that uses one of the traditional message stores. You may have to adjust the mail_spool_directory parameter, as described earlier in the chapter, but for most POP/IMAP servers, you can simply follow the standard installation instructions and start the server. For POP/IMAP servers that don't use a traditional message store, Postfix can still deliver messages using the Local Mail Transfer Protocol, which is discussed in the next section.

Figure 7-1. Postfix and POP/IMAP servers



7.5 Local Mail Transfer Protocol

Some POP/IMAP servers use nonstandard message stores. Since it would be unreasonable to expect MTAs such as Postfix to understand many different proprietary formats, the *Local Mail Transfer Protocol* (LMTP) provides a way to pass email messages from one local mail service to another without depending on a common message store. LMTP is based on, and is a simplified version of, SMTP. With LMTP, the server can either accept an email message immediately or it cannot accept it at all. There is no attempt by the LMTP server to queue or redeliver a message that cannot be delivered immediately.

When an MTA makes a delivery to an SMTP server, where the message is destined for multiple recipients, and one or more recipients cannot accept the message for some reason, the SMTP server takes the responsibility of queuing the message to deliver it later, and reports an overall successful delivery to the MTA. LMTP servers do not queue messages, so they must return an individual status reply for every recipient of a particular email message. For those recipients that could not be delivered, the MTA, and not the LMTP server, takes the responsibility of queuing the message and attempting redelivery.

LMTP conversations can occur between mail subsystems on the same machine or on different machines on a local area network. It is not recommended for wide area networks, since the protocol depends on a quick response to indicate whether the message was delivered. With SMTP there is a recognized synchronization problem between sending and receiving mail systems that sometimes causes duplicate messages to be delivered. It is believed that LMTP over wide area networks would make the problem worse.

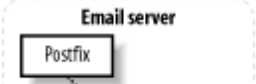
Apart from delivery to nonstandard message stores, a real benefit of the LMTP protocol is that it allows for a highly scalable and reliable mail system. One or more Postfix servers can receive mail from the public Internet and make deliveries to multiple LMTP backend systems. As the load increases, it is a simple matter to add more boxes to the front- or backend systems.

The most common implementation of LMTP delivery is the Cyrus IMAP server from Carnegie Mellon University. It is available from the Project Cyrus web page at http://asg.web.cmu.edu/cyrus/. Cyrus IMAP uses its own message store, as shown in Figure 7-2. This section looks at how Postfix can use the LMTP protocol to hand off messages to Cyrus IMAP. For more information about configuring Cyrus IMAP, see Managing IMAP by Dianna Mullet and Kevin Mullet (O'Reilly).

7.5.1 Postfix and Cyrus IMAP

Cyrus IMAP is intended to run on servers that provide POP/IMAP access only, where users do not need a shell account. If you are creating a mail server for existing users on a system, you will probably want to use another simpler POP/IMAP solution, such as Qualcomm's Qpopper (POP access only) or the University of Washington's IMAP Toolkit, which doesn't require any special configuration to work with Postfix. This section deals with configuration issues for getting Postfix to work together with Cyrus IMAP.

Figure 7-2. Postfix and Cyrus IMAP



Chapter 8. Hosting Multiple Domains

It is very common these days for a single system to host many domains. For instance, oreillynet.com and onlamp.com might run on a single host, but act as if they were two totally different hosts. A system usually has a canonical domain, which is considered its usual or common domain name. Additional domains are configured as virtual domains. Each virtual domain can host services such as web sites and email as if it were the only domain on a server. This chapter explains several different mechanisms for hosting multiple domains. The techniques are explained separately, but it is possible to mix techniques if you must handle different domains in different ways.

To determine which technique or techniques you need, you must decide how Postfix should deliver messages for virtual domains. There are two important considerations that influence how you should configure Postfix for hosting multiple domains:

- Should your domains have separate namespaces? For example, should mail for the two addresses info@ora.com and info@oreilly.com go to the same mailbox or separate ones? We'll refer to the same mailbox scenario as shared domains, and the other as separate domains.
- Does every user require a system account? We'll make the distinction between system accounts that are real Unix accounts on your system and virtual accounts. With virtual accounts, users can have mailboxes on your server, but don't otherwise log in to the system and don't require an entry in */etc/passwd*.

We'll consider four different ways Postfix can handle mail for virtual domains:

- Shared domains with system accounts
- Separate domains with system accounts
- Separate domains with virtual accounts
- Virtual domains with a proprietary message store not managed by Postfix

Your POP/IMAP server will be a major factor in deciding which technique you need. If your POP/IMAP server does not understand virtual domains, then it will most likely require that you have system accounts for all addresses. Some POP/IMAP servers inherently support multiple domains, and deliver messages into a particular directory structure on the local filesystem. Other POP/IMAP servers use their own proprietary message store. Postfix can hand off messages to them using LMTP.

Regardless of the technique you use, all of your virtual domains must be configured correctly in DNS. You should configure DNS for virtual domains the same way you do for your system's canonical domain. See Chapter 6 for information on Postfix and DNS.

8.1 Shared Domains with System Accounts

Accepting mail for multiple domains where every user can receive mail for every domain is the simplest configuration of virtual domains. Simply add your virtual domains to the mydestination parameter. Create user accounts as you normally would, and they can start receiving mail addressed to any of the domains. This technique uses the local delivery agent, providing all of the same features as your normal canonical domain hosting. Users can create their own *.forward* files, and local aliases are available. On a system whose canonical name is oreillynet.com, hosting two virtual domains, ora.com and oreilly.com, the mydomain parameter is set as if oreillynet.com were the only domain, and mydestination is set as follows:

```
mydomain = oreillynet.com
```

8.2 Separate Domains with System Accounts

If you require separate namespaces for each of your virtual domains, the configuration is only slightly more complicated. With separate domains, mail to info@ora.com should go to a different mailbox than mail to info@oreilly.com. In this case, do not list the additional domains in the mydestination parameter. Instead, use virtual_alias_domains:

```
virtual_alias_domains = ora.com, oreilly.com
```

You must create a user account for every email address that will receive messages on your system. Your system accounts do not have to match the email addresses in any way, since you will be mapping the addresses to the accounts separately, but each account must be unique. If your platform supports long usernames, a good way to create unique account names, and to avoid confusion about which accounts are meant to receive mail at which domains, is to use the domain name itself as part of the account name. One possible naming convention is to create accounts such as info.ora.com and info.oreilly.com.

Once Postfix knows which domains to accept mail for, and you have accounts for each address, use virtual_alias_maps to map the email addresses to the accounts you create. In *main.cf*, point the virtual_alias_maps parameter to the virtual alias lookup file. In this example, the file */etc/postfix/virtual_alias* is used:

```
virtual_alias_maps = hash:/etc/postfix/virtual_alias
```

The */etc/postfix/virtual_alias* file contains entries with the email addresses pointing to the system accounts you created, plus any non-local forwarding you need:

```
info@ora.com        helene@localhost
```

8.3 Separate Domains with Virtual Accounts

The drawback for the techniques so far is that you must maintain system accounts for all email addresses on your server. As the number of domains you host increases, so does the effort to maintain all the accounts. In particular, if users only receive email at your server, and don't otherwise log in, you probably don't want to have to create system accounts for each one. Instead, configure Postfix to deliver to a local message store where each virtual email address can have its own mailbox file. Your users then retrieve their messages through a POP/IMAP server.

The local message store works much like normal local delivery, but it doesn't require a one-to-one correspondence between each mail file and a local user account. For this configuration, list each virtual domain in the virtual_mailbox_domains parameter:

```
virtual_mailbox_domains = ora.com, oreilly.com
```

If you have many domains, you can list them in a file and point virtual_mailbox_domains to the file:

```
virtual_mailbox_domains = /etc/postfix/virtual_domains
```

The file */etc/postfix/virtual_domains* then contains a line for each domain:

```
#
```

8.4 Separate Message Store

The last configuration we'll consider is hosting virtual domains with a system using a proprietary message store. To work with these systems, Postfix hands off messages using a protocol like LMTP, letting the proprietary system handle delivery to the correct mail box.

Since Postfix must receive messages before handing them off to the LMTP server, it has to know that it should accept mail for each of the virtual domains. List them in virtual_mailbox_domains:

```
virtual_mailbox_domains = ora.com, oreilly.com
```

You also have to list each email address, so Postfix can accept messages for valid addresses and reject unknown users. Use the virtual_mailbox_maps parameter to point to a lookup file with valid addresses:

```
virtual_mailbox_maps = hash:/etc/postfix/virtual
```

In the */etc/postfix/virtual* file, the righthand value isn't used because all messages are passed along to the POP/IMAP server. You must still include a righthand value because lookup tables must have a key and a value, but the value you use doesn't matter:

```
info@ora.com          General Information Address
```

8.5 Delivery to Commands

As mentioned earlier in the chapter, you can't use local aliases, *.forward* files, and mailing-list programs with virtual domains delivered by the virtual delivery agent. You've seen that you can easily set up aliases through the virtual_alias_maps parameter, but you cannot deliver messages to a command. In this last section, we'll look at working around that issue by demonstrating how to deliver virtual addresses to external programs. The first example sets up delivery to an autoreply program, and the second to a mailing-list manager.

Auto-responders are scripts or programs that process incoming messages and return a reply to the sender of the message without any human intervention. The autoreply program used in this example, *inforeply.pl*, is listed in Example 8-1. This program is meant to handle mail for a dedicated information email address. Users or customers can send a message to the address to request special information. Note that this simple example is inadequate as a general autoreply program, such as the Unix *vacation* command. It does not cache addresses it has already replied to, and it does not do full checking for addresses that should not receive automatic replies (see the sidebar). You might also like to enhance the program to return different types of information, based on the subject or a keyword in the body of the request messages.

Example 8-1. Simple automatic reply program

```
#!/usr/bin/perl -w
```

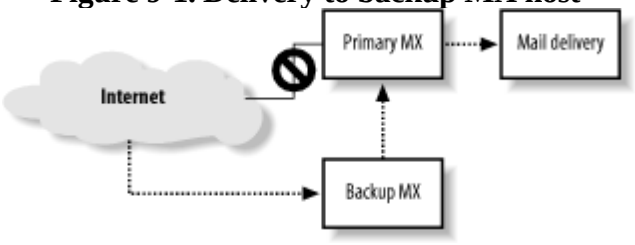
Chapter 9. Mail Relaying

Up until now, we've mostly considered Postfix in its role as the end node for email messages. That is, messages that arrive at the Postfix system are, for the most part, delivered to the local system. But it's also common to find Postfix serving as an intermediate node on the path a message follows to its ultimate destination. In this chapter we'll look at some of the configuration options for Postfix as a client in MTA-to-MTA communications.

9.1 Backup MX

In DNS, MX records refer to *mail exchangers* (see Chapter 6). MX records contain both host and priority (or preference) information for sending mail to a domain. A backup MX server is one that receives mail for a particular domain, but is not the preferred server to receive the mail. If the preferred server or servers are down, the backup MX server receives the mail and queues it until one of the more preferred servers comes back online. Figure 9-1 illustrates delivery to a backup host when the primary host is not available. The backup queues messages until the primary is back online, whereupon the backup can deliver messages to it.

Figure 9-1. Delivery to backup MX host



When your system is configured in DNS as a backup MX host, you don't have to configure any special transport from your system to the primary system. Postfix uses the DNS records to determine how to route mail to the primary MX host. The only configuration required in Postfix is to indicate that it should receive mail for the domain by adding the domain name to the relay_domains parameter. When a sending MTA discovers that the primary mail system for a domain is down, it tries the next preferred one until it finds one that accepts delivery. If your system is a backup MX host, and the destination domain is listed in your relay_domains parameter, Postfix accepts the mail and queues it. Postfix periodically scans its queue and checks for a more preferred system to see if any are able to accept the message. Once a higher priority mail exchanger is back online, Postfix can deliver the message to it.

Postfix continues trying to deliver queued messages for the amount of time specified in the maximal_queue_lifetime parameter, which determines how long deferred messages stay in the queue before they are bounced back to the sender. The default value is five days. If you provide secondary mail service for primary servers that you know will be down longer than the default, you can extend the time.

9.1.1 Relay Recipients

It is highly recommended that you maintain a list of valid recipients for domains you provide backup MX services to. You should develop a regular process for obtaining an updated user list from your primary MX servers. If your system does not know all of the available mailboxes on the primary mail server, it must accept all messages. It's only when your backup MX server tries to deliver them to the primary server that it discovers that a message cannot be delivered. At that point, your server must bounce the message back to the original sender.

Since spammers often send messages to made-up addresses, if your server does not know all the valid email addresses on the primary server, your server will unnecessarily accept a lot of mail that must be bounced. The bounce problem is exacerbated by the spammer tactic of forging sender addresses by using the real email addresses of innocent bystanders. The forged addresses receive all of the error notices for messages they never sent (see Chapter 11). The relay_recipient_maps parameter specifies lookup tables that should contain all of the addresses for domains listed in your relay_domains parameter:

```
relay_recipient_maps = hash:/etc/postfix/relay_recipients
```


The *relay_recipients* file should contain entries with the recipient address on the lefthand side. The righthand side is not used by Postfix, but you must specify a value:

```
#
```

9.2 Transport Maps

Postfix can be configured to relay to any other host, regardless of how DNS MX records are set up. This section discusses the transport_maps parameter in general. Later sections and other chapters in the book present specific configurations that use it.

Conceptually, transport maps override default transport types for delivery of messages. The transport_maps parameter points to one or more transport lookup tables. The following entry sets up */etc/postfix/transport* as a transport map lookup table:

```
transport_maps = hash:/etc/postfix/transport
```

The keys in a transport lookup table are either complete email addresses or domains and subdomains. (Email addresses as lookup keys for transport maps require Postfix 2.0 or later.) When a destination address or domain matches a lefthand key it uses the righthand value to determine the delivery method and destination. Example 9-1 lists some possible transport map entries.

Example 9-1. Transport map entries

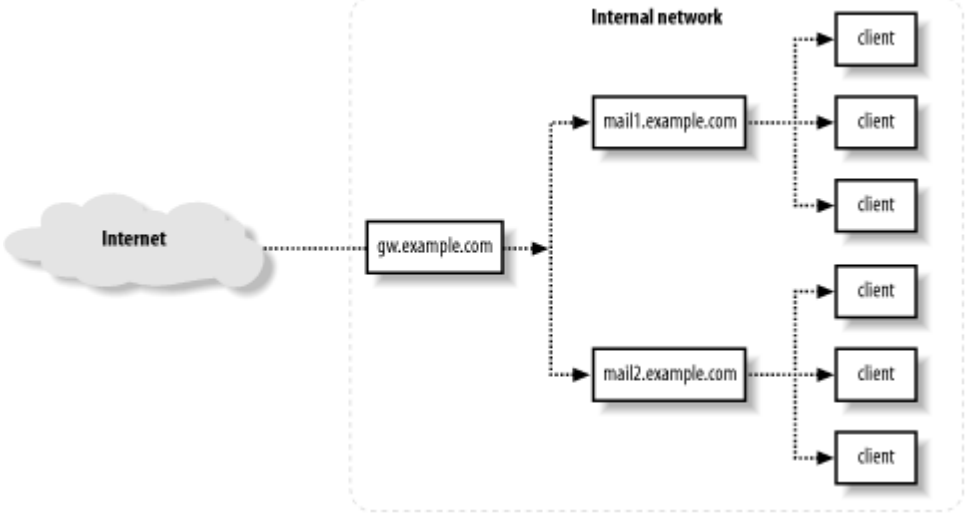
```
example.com         smtp:[192.168.23.56]:20025
```

9.3 Inbound Mail Gateway

A mail gateway is an email system that accepts messages and relays them to another system. Gateways might provide a path from one network to another, or from one protocol to another. A common use of a mail gateway is a server that accepts all the mail for a network from the Internet and relays it to internal mail systems. Mail gateways are commonly set up in conjunction with firewall systems to limit the number of servers that need direct access to the Internet.

Imagine a company network such as the one depicted in Figure 9-2. There are sub-domains for different workgroups at the company, and each workgroup has its own internal mail server. The gateway system gw.example.com receives all the mail for the network. The human resources department gets email addressed as user@hr.example.com, and their mail should go to the server mail1.example.com. The sales department uses user@sales.example.com, and their mail should go to mail2.example.com. The client hosts in each subnet retrieve mail from their respective mail servers. Transport maps are required to set up the mail gateway gw.example.com to relay messages to the correct internal mail servers.

Figure 9-2. Email gateway to internal systems



The following procedure demonstrates how to configure gw.example.com to relay messages to the correct internal systems:

1. Make sure that the DNS has been configured correctly with MX records for hr.example.com and sales.example.com pointing to the gateway gw.example.com.
2. In your *main.cf* file, set relay_domains to include the two internal domains:
   ```
   relay_domains = hr.example.com, sales.example.com
   ```
3. Make sure that the transport_maps parameter points to your transport lookup table:
   ```
   transport_maps = hash:/etc/postfix/transport
   ```
4. Add entries to your *transport* file for each domain pointing to the correct internal mail systems:
   #

9.4 Outbound Mail Relay

When a mail system does not have adequate connectivity or all of the information it needs to relay messages, it can forward them to another system that is in a better position for relaying. Consider the network in Figure 9-2 again. If the internal mail systems don't have direct access to the Internet, they can't deliver messages sent by the users in their subnets. They can, however, pass along all messages to the gateway mail system, which can make the deliveries for them. The following procedure demonstrates setting up Postfix on mail1.example.com to relay all messages it receives to gw.example.com, which can then make the outbound deliveries.

Before configuring the internal mail systems, make sure that the mail gateway is set up to permit relaying from the internal mail systems. The mynetworks parameter (see Chapter 4) should encompass the IP addresses of the internal mail systems, and if you use SMTP UBE restrictions (see Chapter 11), be sure to include permit_mynetworks among the rules to allow relaying:

1. Check the mynetworks (or mynetworks_style) parameter to make sure it includes the client systems.
2. Have the users in the workgroup configure their various mail clients to use mail1.example.com as their SMTP server.
3. In *main.cf*, set the parameter relayhost to point to the gateway system:
   ```
   relayhost = [gw.example.com]
   ```
4. Reload Postfix so that it recognizes the changes in its configuration file:
   ```
   # postfix reload
   ```

Now all messages delivered to mail1.example.com are relayed through gw.example.com.

9.5 UUCP, Fax, and Other Deliveries

The Postfix online documentation describes configuring Postfix for delivery to a FAX system and setting up a gateway for UUCP. These provide good examples for configuring Postfix to work with all kinds of special devices. If you need to create a gateway between different types of systems or different networks, transport maps provide the mechanism for directing mail to the other systems or devices.

Chapter 10. Mailing Lists

Mailing lists provide a convenient way to send a single email message to many recipients. They allow a nearly unlimited number of correspondents to carry on conversations through email. A server-based, centrally managed mailing list has many advantages over other mechanisms to send messages to multiple recipients. If you regularly send email to the same group of people, typing in lists of recipients is too tedious and prone to error to be practical. MUAs usually have a facility that lets you create personal aliases that associate an easily remembered name with a list of email addresses. Personal aliases are fine for an individual, but as soon as the list has to be shared with others, it is no longer a practical solution. Major advantages of centrally managed mailing lists are that changes are made in a single place, and the new information is immediately available to anyone sending messages to the list. Other advantages become evident when you use Mailing List Managers (MLMs) to administer the list, relieving administrators from manually updating addresses.

In this chapter we look at creating simple mailing lists within Postfix itself, and then configuring Postfix to deliver messages to MLMs for more sophisticated list management. In deciding whether or not to create your mailing list within Postfix or to use an MLM, consider how often the list has to be changed, who will make the changes, and whether you need some of the other features of an MLM, such as moderated lists and digest versions. MLMs allow users to subscribe and unsubscribe themselves and to make changes to their addresses, if necessary. If you have relatively static lists or users who come to you for subscribing and unsubscribing anyway, you probably don't need the overhead of an MLM. You can always run both flavors of lists, if that fits your environment.

There are many aspects and nuances to managing a mailing list. If you will be taking on the task, you should consult a text that deals specifically with mailing-list management such as Managing Mailing Lists by Alan Schwartz (O'Reilly).

10.1 Simple Mailing Lists

Postfix provides the means to create simple mailing lists through the normal alias facility (see Chapter 4). Because aliases can point to lists of addresses or files that contain lists of addresses, it is easy to create an alias that points to multiple names. You can create list aliases in the system *aliases* file, or in any other file that you specify in the alias_maps parameter. See more about the alias_maps parameter later in the chapter. The default alias file when you install Postfix is */etc/aliases*.

Let's suppose that you administer mail for the domain example.com, and you want to create a new mailing list for people to discuss needlepoint. You decide to create a mailing list alias needlepoint@example.com to be used for online discussions. Edit your alias file, and add the following line with the email addresses of people who want to subscribe to the list:

```
needlepoint:        rgrier@oreilly.com, gmhopper@onlamp.com,
```

10.2 Mailing-List Managers

Running mailing lists within Postfix is fine for static lists. But lists that change frequently are better handled by a mailing-list manager (MLM). With an MLM, the administrator of the list doesn't have to manually edit the list file to add, delete, or change addresses because list members can subscribe and unsubscribe themselves. MLMs also support other features such as archiving of messages, digests of discussions, and the ability to moderate a list by allowing an administrator to review messages before they are posted to all members.

MLMs work by pointing normal Postfix aliases to commands that handle the distribution of messages and management of lists. MLMs use administrative aliases that point to programs to handle list functions such as subscribing and unsubscribing members from the list, handling bounced messages, and possibly filtering messages sent to the list. The lists themselves actually work the same way as the simple aliases from the last section. Each list has its own file to store list members, but rather than editing the file yourself, you can have the MLM automatically add and remove addresses.

The next two sections look at two popular MLMs: Majordomo and Mailman.

10.2.1 Majordomo

Majordomo is one of the more popular MLMs and has been available since the early 1990's. It offers a complete set of MLM features, and nearly all administration takes place by sending commands through email messages. Little to no intervention is required by a postmaster once a list has been created. There are also web-based administration packages available to work with Majordomo, allowing much of the list administration to take place from a web site.

Majordomo is available at the Majordomo home page (http://www.greatcircle.com/majordomo/.) It requires Perl and works with Perl4 Version 4.036 or Perl5 Version 5.002 or better. Future releases will probably require Perl5. Majordomo also makes use of a small wrapper program written in C. If you are planning to build the package from scratch, you must have an ANSI C compiler.

If you configure Majordomo for moderated lists, where a list administrator approves posts using the Majordomo-supplied *approve*, you have to make an adjustment for Postfix and Majordomo to work together correctly. Postfix prepends a Delivered-To: header to messages it handles. It then uses the header to detect mailer loops. When a Majordomo message is delivered to a moderator for approval who then pipes the message through the *approve* command, it is sent back to the list with all of its original headers intact. When Postfix receives the message again, it recognizes that it has already seen the message and reports a mail delivery loop.

The easiest way to fix this issue is to make a small change to the Majordomo *approve* script (which is written in Perl). You'll have to edit the file, normally located in the */bin* directory located below the main Majordomo installation directory. If you follow the steps in the procedure below, your file will be located at */usr/local/majordomo/bin/approve*. Edit the file and find the subroutine called process_bounce. Within that routine, there is a while loop, as shown below. Insert the emphasized line as shown, save the file, and you're done:

```
while (<$FILE>) {
```

Chapter 11. Blocking Unsolicited Bulk Email

Unsolicited Bulk Email (UBE), also referred to as Unsolicited Commercial Email (UCE), is commonly called spam. Spamming is the practice of sending mass mailings to large numbers of people who have had no prior relationship with the sender and who didn't ask to receive such mail. Spam exists because it's so cheap to send. The incremental cost of adding even hundreds of thousands of recipients to a mailing is relatively small, so spammers target as many email addresses as they possibly can. This chapter looks at the problem of spam and the tools Postfix provides to help limit the consequences.

11.1 The Nature of Spam

There is a decidedly dishonest component to most spam. Spammers make no effort to match message content with a recipient's interests, and their messages frequently lie, claiming that the recipient has an association with the company or its partners or in some way requested information. Messages are sometimes designed to look like an actual exchange between two people that was mistakenly misdelivered in the hopes of sparking interest in some product or service.

Spam frequently offers instructions to *opt out* from receiving more messages; however, in many cases this is simply a subterfuge on the part of the spammer to confirm that your email address is good. By replying to such messages, you confirm that your address is a legitimate one. Following the directions provided will more than likely cause your address to be added to more spammer lists.

Spammers often try to hide their trail so their messages cannot be traced back to them. They purposely use false return addresses and forge header information. They seek out misconfigured systems that allow them to relay anonymously. More recently spammers have broken into systems and installed their own secret relay servers. Spammers commonly encode their messages or insert random letters to circumvent spam filters.

Some of the techniques employed by spammers have sideeffects that make the problem much worse than the act of spamming itself. In their scatter-shot approach, spammers send messages to email addresses they think are likely to exist whether they actually do or not. Some launch dictionary assaults on mail servers where they run through preassembled lists of names hoping to find a match with a user on the mail server.

11.2 The Problem of Spam

While spam may seem like a minor issue on a small scale, it is a significant problem on the Internet. A system hosting hundreds or thousands of users each receiving dozens or hundreds of unwanted messages every day can have substantial difficulties dealing with the onslaught. There is a real cost to the victims of spam. It unfairly uses the bandwidth and disk space of its recipients and their providers.

Other costs brought on by spam include technical support personnel time, when technicians or administrators must help users clean up flooded mailboxes. Sometimes the volume of spam can even make a system unusable for its intended purpose (clogging bandwidth or filling disk space). In such a case the effects of spam are no different from those of a denial-of-service attack. Even in less drastic circumstances, spam interferes with legitimate uses of email. Important messages can easily be overlooked in a flood of spam or mistakenly deleted when littered mailboxes are cleaned up.

A significant issue with spam is dealing with messages addressed to nonexistent users. Some mail systems recognize that a destination address is bogus and can reject mail before it is accepted; other systems must receive the mail first and then bounce it as undeliverable. The volume of bounces can easily clog a queue and interfere with the delivery of legitimate messages. Since the return addresses often don't really exist, the bounces cannot be delivered and sit in the queue undergoing many redelivery attempts until they expire.

Another spamming trick is to use a legitimate return address that belongs to an innocent third party. The target or relay systems that receive the spam send bounce messages to the supposed sender, helpfully letting that person know that the recipient does not exist. In this case, thousands or millions of bounce messages will be delivered to the unfortunate victim in a phenomenon referred to as *backscatter*. This victim isn't involved in any way in the original delivery of the spam. In most cases, the only solution for these completely innocent bystanders is to abandon the victimized address and start using a new one.

11.3 Open Relays

If you operate an email server on the Internet, you have a responsibility to make sure that you do not create an *open relay* that spammers can use as a launching point for their activities. An open relay is a mail system that permits outside systems to send mail to other outside systems, passing the messages along so that the originating system does not have to deliver directly to its target. Spammers constantly scan for misconfigured systems that permit them to relay mail. Before spam became such a problem on the Internet, mail administrators often operated open relays because it made their systems convenient for their users. Now nearly all SMTP software systems are configured by default not to be open relays. Postfix is no exception.

If your system is abused as an open relay, it will most likely be so bogged down with sending spam that its performance will be hindered for your legitimate users. If you choose to accept spam into your own system that is, of course, up to you, but you must take steps to ensure that your system is not used to abuse other systems. There is a good possibility that if spammers use your system to relay mail, your network will end up on a blacklist. Once your site is blacklisted, many sites will reject all messages from your network, both relayed spam and legitimate messages from your users. Chapter 4 discusses safely configuring Postfix to prevent your system from being abused.

11.4 Spam Detection

As long as you're not operating an open relay, you can be confident that your systems are not being used to harm other systems. Your next consideration is to protect yourself and your users by limiting the spam your network receives. Ideally, your mail server could simply reject any message that looks like spam. Unfortunately, whereas humans can look at a message and know instantly that it's spam, computers have a tougher time detecting it without making mistakes. The ugly truth is that once you start to reject spam, there is always a risk that you will block legitimate correspondence.

Misidentifying a legitimate message as spam is referred to as a *false-positive* identification. Your anti-spam efforts are an attempt to detect as much spam as you can with the fewest possible false-positives. You have to weigh the size of your spam problem against the possibility of rejecting real email when deciding how aggressive to be in implementing your anti-spam measures. The extremes range from permitting all spam to accepting mail only from preapproved individuals. Preapproval may seem severe, but the problem is getting bad enough for some people that *whitelist* applications, where any correspondent you receive mail from must be identified ahead of time, are becoming more common.

There are two primary ways of detecting spam: identifying a known spamming client and inspecting the contents of a message for tell-tale phrases or other clues that reveal the true nature of a spam message. Despite the difficulties, postmasters can achieve some success with minimal false-positives by implementing various spam-detection measures.

11.4.1 Client-Based Spam Detection

Client-blocking techniques use IP addresses, hostnames, or email addresses supplied by clients when they connect to deliver a message. Each piece of information supplied can be compared to lists of items from known spamming systems. Spamming systems might be owned by actual spammers, but they might also be unintentionally open relays managed by hapless, (almost) innocent mail administrators. In either case, if a system is regularly sending you spam, you will probably decide to block messages from it. One problem with identifying spam by IP address, hostname, or email address is that these items are easily forged. While the IP address of the connecting system requires some sophistication to spoof, envelope email addresses are trivial to fake.

11.4.1.1 DNS-based blacklists

In a grass-roots effort to stem the tide of spam on the Internet, various anti-spam services, generally called DNS-based Blacklists (DNSBL) or Realtime Blacklists, have developed. These services maintain large databases of systems that are known to be open relays or that have been used for spam. A newer, increasingly more common problem is with systems that have been hijacked by spammers who install their own proxy software that allows them to relay messages. These hijacked systems can also be used in distributed denial-of-service attacks. There are DNSBL lists that are dedicated to listing these unwitting spam relays. The idea is that by pooling the information from hundreds or thousands of postmasters, legitimate sites can try to stay ahead of spammers.

Usually, these systems work by adding a DNS entry to their domain space for each of the IP addresses in their database that have been identified as spam-friendly open relays. For example, if the host at IP address 192.168.254.31 has been identified as an open relay, the (fictitious) DNSBL service No Spam Unlimited using a domain name of nospam.example.com creates a DNS entry like 31.254.168.192.nospam.example.com. When a client connects to your Postfix system, Postfix can check the No Spam DNS server to see if there is an entry for the client's IP address. If the IP address has been identified as an open relay system, Postfix can reject the message.

11.5 Anti-Spam Actions

Broadly speaking you have a few choices once you have detected spam:

- Reject spam immediately during the SMTP conversation. Rejecting spam outright is an attractive idea because you never have to store a copy of the message and worry about what to do with it. The sender of the message is responsible for handling the error. If your site has a low tolerance for rejecting legitimate messages, you might prefer to accept suspect messages and develop a process to review them periodically to make sure that there are no good messages in with the bad.
- Save spam into a suspected spam repository. If you save the suspect messages and review them periodically, you can be sure that you don't miss any legitimate mail. The task is cumbersome and usually requires frequent reviews, so you may not gain much over allowing suspect messages into users' mail boxes.
- Label spam and deliver it with some kind of spam tag. This option provides users with flexibility in determining their own tolerance for spam versus their sensitivity to missing real messages. Postfix doesn't currently have a built-in mechanism for labeling spam. You can easily have Postfix work with an external content filter to handle the labeling (see Chapter 14). If the content filter delivers tagged messages to individual users, they can configure their email software to deal with it according to their own preferences.

When using an MTA for spam detection, the rejection option is usually best. If you want more flexibility, consider using options that filter spam at the MDA or MUA level. A combination of spam filtering is also a good alternative. You can configure Postfix to reject the obvious spam, allowing suspicious messages through to the next level where another agent can perform the most appropriate action.

Postfix really excels in its tools to help you identify spam clients and reject them. Rejecting messages with Postfix requires fewer system resources than invoking external filters after the message has been accepted. If you are concerned about losing legitimate mail, there are still a couple of safety measures available that we'll look at when configuring Postfix.

11.6 Postfix Configuration

The rest of this chapter discusses the various types of UBE checks Postfix provides. It considers four different categories of spam detection which are listed below.

Client-detection rules.

Four parameter rules that work with pieces of the client identity. Each rule is assigned a list of one or more restrictions that can explicitly reject or accept a message or take no position one way or the other (commonly indicated as DUNNO). For example, you can configure a rule that includes a restriction to reject a particular client IP address.

Syntax-checking parameters.

Parameters that check for strict adherence to the standards. Since spammers often don't follow the published standards, you can reject messages that come from misconfigured or poorly implemented systems. Some of the client restrictions also fall under this category.

Content checks.

You can check the headers and the body of each message for tell-tale regular expressions that indicate probable spam.

Restriction classes

You can define complex client-detection rules with restriction classes. These allow you to combine restrictions into groups to form new restrictions.

When configuring Postfix to detect spam, you also specify what to do with messages identified as spam. In general, Postfix can reject them outright, separate them into a different queue, or pass them along to an external filter.

11.7 Client-Detection Rules

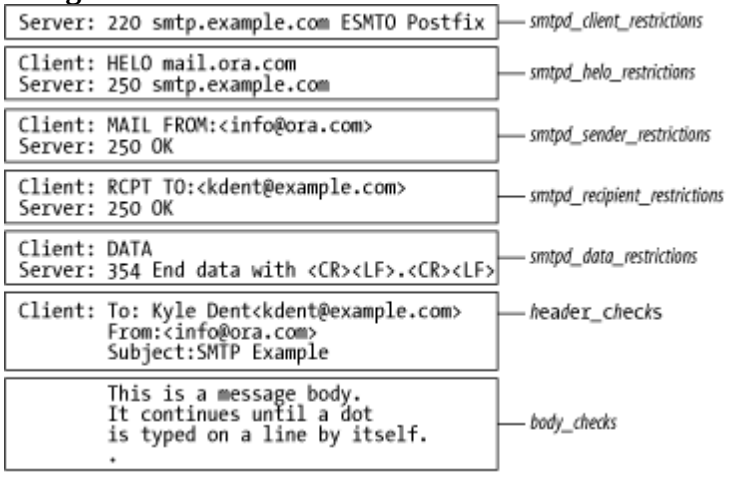
Postfix provides the following rules that are assigned restrictions based on client information:

- smtpd_client_restrictions
- smtpd_helo_restrictions
- smtpd_sender_restrictions
- smtpd_recipient_restrictions
- smtpd_data_restrictions

Each one corresponds to a step of the SMTP transaction. At each step, the client provides a piece of information. Using the client-supplied information, Postfix considers one or more restrictions that you assign to each rule. Figure 11-1 shows an SMTP conversation along with the client rule applied at each step. The header_checks and body_checks are discussed later in the chapter.

Let's review the SMTP conversation to see where each of the parameters fits in.

Figure 11-1. SMTP conversation with client rules



11.7.1 The SMTP Conversation (Briefly)

The SMTP conversation in Figure 11-1 should be familiar to you from Chapter 2. Example 11-1 shows the log entries for the transaction. First, an SMTP client connects to Postfix over a socket. Because of the way sockets function, Postfix learns the IP address of the client when it establishes the connection. You don't see the client IP address in the figure, but it is logged by Postfix. You can accept or reject a message based on the client hostname or IP address, thus blocking specific hostnames or IP and network addresses.

Example 11-1. SMTP logging

11.8 Strict Syntax Parameters

There are two parameters configured in *main.cf* that require strict adherence to Internet email standards. Enable the smtpd_helo_required parameter to require that SMTP clients start the conversation with the HELO/EHLO verb, as described in the SMTP RFC. By default Postfix is rather lenient with clients that do not follow the protocol exactly. If you specify smtpd_helo_required = yes, and a client skips this step, Postfix rejects the message. The RFC also specifies exactly how envelope addresses should be formatted. Normally, Postfix accepts nearly any envelope address that it can make sense of, but if you specify strict_rfc821_envelopes = yes, Postfix rejects messages from clients that do not send correctly formatted addresses.

In actual practice, it's probably a good idea to require HELO because most clients at least follow the basic steps of the protocol. On the other hand, there are a number of clients that don't get address formatting correct. Being too strict here might lose legitimate messages.

11.9 Content-Checking

The last chance you have to reject a message from Postfix directly is by checking the contents of the message itself. Postfix offers simple content checking through the parameters:

- header_checks for message headers
- mime_header_checks for MIME headers
- nested_header_checks for attached message headers
- body_checks for the body of a message

These checks are an all-or-nothing feature with Postfix. There is no way to bypass checks for certain senders or recipients. For more sophisticated analysis, you should use a separate content filter specifically designed to detect spam. See Chapter 14 for more information on using filters with Postfix.

Each parameter points to a lookup table containing regular expression patterns and actions. The patterns are compared to strings within email messages. If Postfix finds a match, the specified action is executed. By default regular expression checking is not case-sensitive. See Chapter 4 for information on using regular expressions with Postfix lookup tables.

11.9.1 Content Checking Configuration

By default mime_header_checks and nested_header_checks use the same lookup tables as header_checks. If you want to distinguish checks for each one, you can configure them separately; otherwise, configuring header_checks causes mime_header_checks and nested_header_checks to use the same patterns as header_checks. When you assign the checking parameters, indicate both the lookup table and which type of regular expression you are using (see Chapter 4):

```
header_checks = regexp:/etc/postfix/header_checks
```

11.10 Customized Restriction Classes

Restriction classes provide the last wrinkle in the Postfix anti-spam parameters. They allow you to define a set of restrictions that you can assign to the righthand side of an access table. They cannot be used in header and body checks—only in access tables. Restriction classes let you set up different restrictions for different clients, senders, and recipients. Restriction classes are a powerful tool that can provide great flexibility in Postfix UBE restrictions. If you require any sort of complicated rules to block spam, it is well worth your while to invest the time to understand restriction classes.

Restriction classes are particularly useful when you need to create exceptions to your normal restrictions. To illustrate with an example, let's create two classes of users. One group wants to receive all messages addressed to them whether or not the messages are spam. The other group prefers particularly stringent checks against spam even at the risk of losing some legitimate mail.

11.10.1 Sample Restriction Classes

We'll call the two classes "spamlover" and "spamhater." You must list all of the restriction classes you plan to define in the smtpd_restriction_classes parameter:

```
smtpd_restriction_classes = spamlover, spamhater
```

We've invented the names of the classes, but once listed with smtpd_restriction_classes, they can be treated like any other restriction rule. You can assign a list of restrictions to be considered for the class. Once defined, the restriction class can be used as an action in an access table. When Postfix encounters the class, it steps through the assigned restrictions.

We'll define "spamhater" with several restrictions:

```
spamhater =
```

11.11 Postfix Anti-Spam Example

Now that we've covered the many aspects of Postfix's anti-spam arsenal, we'll finish with an example configuration. Requirements vary considerably from site to site, so it's impossible to make actual recommendations apart from the considerations that have been discussed in this chapter. Example 11-2 can provide a starting point, but you must decide for yourself which restrictions fit your own circumstances.

Example 11-2. Sample restrictions to block UBE

```
smtpd_restriction_classes =
```

Chapter 12. SASL Authentication

The basic SMTP protocol does not provide a mechanism to authenticate users. Since email envelope addresses are so easy to fake, you can't know who is sending mail to your server unless you have a reliable means to authenticate clients. To allow mail relay privileges on your server, you need assurance that senders are who they claim to be, and you cannot rely on the senders' email addresses as identification. In this chapter, we look at using the *Simple Authentication and Security Layer* (SASL) as a means to control mail relaying and generally to identify who is using your mail server.

You might want to provide access to individuals using your mail server as their SMTP server, or to other MTAs that relay through your system. We'll also look at configuring Postfix to provide its own credentials to other MTAs that may require authentication before permitting email delivery or relaying. Chapter 4 discusses the mail relay problem in general, and some other solutions to consider.

Because you lock down your mail servers to prevent unauthorized relaying, some of your users might have trouble sending email when they are not on your network. If you have users that travel with laptops, for example, they will likely connect through a nearby ISP and get an IP address from its dial-up pool. Or perhaps you have users that work from home. In any case, whenever you don't know what users' IP addresses will be, SASL can provide the means to reliably identify them.

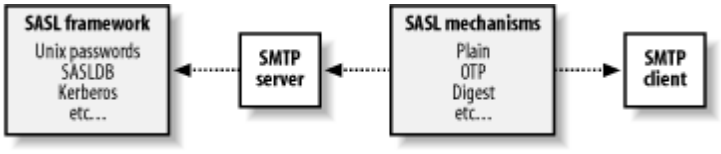
RFC 2554, "SMTP Service Extension for Authentication," provides an extension to the basic SMTP protocol that allows clients to authenticate to an SMTP server using the SASL protocol. We'll show how to use the Cyrus SASL libraries from Carnegie Mellon to add SASL to Postfix. You may optionally also want to add support for TLS (see Chapter 13). TLS (formerly SSL) is most commonly used to encrypt conversations between web browsers and servers, but works equally well for mail servers and clients. Since some of the SASL password mechanisms transmit passwords as plaintext, you can use TLS to make sure your passwords are not sent in the clear.

Adding SASL to Postfix requires that you have the Cyrus libraries on your system and that your Postfix system be compiled with them. Remote users must configure their email clients to send a login and password when they want to relay mail through your system. Most modern email clients make this a fairly easy configuration option.

12.1 SASL Overview

SASL is a general method to add or enhance authentication in client/server protocols. Its primary purpose is to authenticate clients to servers. When you configure SASL, you must decide on both an *authentication mechanism*, for the exchange of authentication information (commonly referred to as user *credentials*), and an *authentication framework* for how user information is stored. The SASL authentication mechanism governs the challenges and responses between the client and server and how they should be encoded for transmission. The authentication framework refers to how the server itself stores and verifies password information. Figure 12-1 illustrates these two processes. Once an authentication is successful, the server knows the user's identity and can determine which privileges the identified user should have. In the case of Postfix, it is the privilege to relay mail. You can also optionally limit identified users to using a particular sender address when they relay mail.

Figure 12-1. SASL authentication frameworks and mechanisms



12.1.1 Choosing an Authentication Mechanism

The client and server must agree on the authentication mechanism they'll use. (See the Cyrus documentation for currently supported mechanisms.) Some of the more common mechanisms are listed below:

PLAIN

The PLAIN mechanism is the simplest to use, but it does not include any encryption of authentication credentials. You may want to use TLS (see TLS information in Chapter 13) in conjunction with the PLAIN mechanism. The login and password are passed to the mail server as a base64 encoded string.

LOGIN

The LOGIN mechanism is not an officially registered or supported mechanism. Certain older email clients were developed using LOGIN as their authentication mechanism. The SASL libraries support it in case you have to support such clients. If you need it, you must specify support for it when you compile the libraries and Postfix. See Appendix C if you are building your own Postfix. If you are using a packaged distribution and you need LOGIN support, check the documentation with your distribution to make sure it includes it. If it is used, the authentication exchange works the same as the PLAIN mechanism.

OTP

OTP is an authentication mechanism using *one-time passwords* (formerly S/Key). The mechanism does not provide for any encryption, but that may not be necessary since any captured password is good for only a single session. SMTP clients must be able to generate OTP authentication credentials.

DIGEST-MD5

12.2 Postfix and SASL

Before getting started with SASL, you should decide which framework and mechanism you will use because it affects your installation and configuration. In order to enable SASL authentication in Postfix, you must have the Cyrus SASL library and a copy of Postfix with SASL support compiled in. Some platforms have precompiled packages available with support for SASL. If you want to use a precompiled Postfix package make sure that it specifically includes support for SASL and has the necessary SASL libraries. Furthermore, make sure that the SASL libraries were compiled with the options you need for your situation. The relevant options are described throughout the rest of this section.

Cyrus SASL library development is currently following two tracks: SASL and SASLv2. The SASL track is being phased out in favor of SASLv2. In the future, you can expect Postfix to include support for SASLv2 only. This chapter discusses SASLv2. You must have the correct combination of versions of both Postfix and the SASL libraries.

You should be able to use the latest stable version of the SASLv2 track of the Cyrus libraries. Postfix support for SASLv2 first appeared in the experimental release Version 1.1.7-20020331 and was included in the official release 2.0. It is very important that you use a version of Postfix that supports SASLv2 to follow the directions in this chapter. When the text mentions SASL, it refers to Version 2 of the library.

12.3 Configuring Postfix for SASL

Before you get started, decide on the authentication mechanisms you plan to support and the authentication framework you want SASL to use with Postfix.

12.3.1 Specifying a Framework

The SASL library uses a separate configuration file for each application it works with. Postfix uses a file named *smtpd.conf* for SASL purposes. This file is usually located at */usr/local/lib/sasl2/smtpd.conf*. At a minimum, *smtpd.conf* contains a line indicating the framework to use. We are going to look at specifying either Unix passwords or separate SASL passwords for Postfix authentication. See the Cyrus documentation to see other options you might include in *smtpd.conf*.

12.3.1.1 Unix passwords

Often, it's most convenient for SASL to use the existing system database to authenticate users. Historically, this meant using the */etc/passwd* file. Today, it's more likely that you use */etc/shadow*, PAM, or some related authentication database. Since these passwords are not available to unprivileged processes, and Postfix purposely runs with limited privileges, it cannot normally authenticate users.

The Cyrus libraries deal with the problem by providing a special authentication server called *saslauthd*. It handles requests on behalf of Postfix. The *saslauthd* daemon requires superuser privileges; however, since it runs as a process distinct from Postfix and does not have to communicate outside of your network, the security impact is minimized. If you are going to use Unix passwords with SASL, you must run the *saslauthd* daemon that ships with the Cyrus distribution. Note that using Unix passwords with *saslauthd* limits you to plaintext passwords because the daemon needs the actual passwords to verify them. See Chapter 13 for using encryption between Postfix and email clients.

To specify that you want Postfix to use the *saslauthd* daemon for authentication, create the *smtpd.conf* with a line like the following:

```
pwcheck_method: saslauthd
```

saslauthd comes with the Cyrus SASL distribution and should be installed in a convenient location. The daemon must be running in the background for Postfix to use it to authenticate clients. When you start *saslauthd*, you tell it what type of password system you are using with the -a option. The most common options are pam, shadow, or getpwent (for the conventional */etc/passwd*). For example, to start the daemon on a system that uses PAM for authentication, type the command:

```
# saslauthd -a pam
```

Consult the Cyrus documentation for other options when using *saslauthd*. Also, you probably want this daemon to start automatically at system initialization so that it is always available for your Postfix server. You can add *saslauthd* to your system's startup processes in the same way you add other daemons such as Postfix.

12.3.1.2 SASL passwords

If you don't want your mail server to use existing system accounts, you can create a separate database of users and passwords that is independent of the system password mechanism. You can create accounts for email users who have mail access only and will not be able to log into the host itself. Include the following line in your *smtpd.conf* file:

```
pwcheck_method: auxprop
```

12.4 Testing Your Authentication Configuration

It's probably best to try authenticating to your SMTP server manually before having your users attempt it with their email clients. By connecting to your SMTP server and manually authenticating, you can see exactly what response you get, and you can immediately check your log file for any other important information.

The easiest way to connect to your SMTP server is to use a Telnet client and then start speaking SMTP to your server. (Chapter 2 shows a sample SMTP session.) The PLAIN mechanism is the easiest to test, so if you have disabled it, you may want to enable it just to confirm that authentication works. You can disable it after you are finished testing.

To authenticate using the PLAIN mechanism, you must send the command AUTH followed by your credentials encoded using base64. Your credentials are a combination of the authorization identity (identity to login as), followed by a null character, followed by the authentication identity (identity whose password will be used), followed by a null character, followed by the password. Usually, the authorization identity is the same as the authentication identity, and we'll assume as much here. Using the credentials for the user kdent, you need to encode the string 'kdent\0kdent\0Rumpelstiltskin'.

The tricky part is to encode your credentials in base64 without including a carriage return character. If your system has the *mmencode* and *printf* commands, it should be simple. The *printf* command prints formatted strings, and does not automatically include a linefeed like the more common *echo* command. The *mmencode* command simply converts strings into various MIME formats and uses base64 by default, which is exactly what we need.

You can get the encoded string you need by executing the following:

```
$ printf 'kdent\0kdent\0Rumpelstiltskin' | mmencode
```

12.5 SMTP Client Authentication

You may want your Postfix server to relay through other servers that require SMTP authentication. In addition to requiring passwords on your own server, you can configure Postfix to provide login names and passwords when relaying mail through other SMTP servers.

You have to provide Postfix with a password file that contains the credentials it should use when authenticating to other servers. Entries in the password file contain a domain or hostname, username, and password in the form: *domain username:password*. For the domain or hostname, Postfix first checks for the destination domain from the recipient address. If it doesn't find the domain, it then checks for the hostname it is connecting to. This allows Postfix to work easily with sites that have multiple MX hosts that share the same user database. Use smtp_sasl_password_maps parameter to specify where your password file is.

The client smtp_sasl_security_options parameter works just like server smtpd_sasl_security_options (discussed earlier in the chapter) for the SMTP servers. If you don't specify any options, the default allows all available mechanisms including plaintext but not anonymous logins.

12.5.1 Procedure to Enable SMTP Client Authentication

Use the following steps to configure Postfix to provide a login and password when relaying mail. In this example, you'll set up two different passwords for Postfix to authenticate when relaying through any server for the domain ora.com and through a host called mail.postfix.org:

1. Create a file called */etc/postfix/sasl_passwd* with entries for each host, login, and password combination you need. Your file should resemble the following:

```
ora.com                 kdent:Rumpelstiltskin
```

Chapter 13. Transport Layer Security

Transport Layer Security, or TLS (formerly known as SSL), enhances TCP communications by adding encryption for privacy and message integrity. RFC 3207 defines an extension to SMTP known as STARTTLS. Its primary purpose is to provide privacy in peer-to-peer communications. It can also give you assurances that your mail is not being delivered to a rogue system posing as the server you think you're sending mail to. Another useful application is in combination with SASL, to protect plaintext passwords that would otherwise be sent in the clear.

One nice benefit of TLS is that you can obtain the privacy and assurances of reliable server identification without a previous arrangement between systems. Strong authentication is also possible if your users' email clients support it. By using client certificates, which are cryptographically signed identifiers (see sidebar), your mail server can be sure that connecting clients are indeed who they claim to be. You can use client certificates in place of or in conjunction with SASL authentication discussed in Chapter 12. There is administrative overhead in managing client certificates and assisting users in configuring their email clients to use them, while using TLS just to encrypt authentication credentials is fairly easy to set up.

It is important to note, however, that TLS is not meant to protect the contents of email messages. When you encrypt the transmission between a client and server, everything (including the message) is encrypted. However, TLS protects only the transmission between the two systems. After the server receives a message, it is probably stored as plaintext. You can't be sure if the message will be encrypted or not when the server forwards it to the next destination, or when the final recipient downloads the message to read it. Unless you can control and encrypt the path all the way from the originating client to the ultimate recipient of the message, it will most likely pass in the clear at some point on its way to delivery. To achieve end-to-end privacy you need a client solution such as PGP or S/MIME.

13.1 Postfix and TLS

Support for TLS in Postfix is provided by a set of patches written by Lutz Jänicke. You can follow the link for Add-on Software from the Postfix home page to download the patches. (See Appendix C for information on building Postfix with the TLS patches.) If you are using a prebuilt Postfix package for your platform, make sure that it has the TLS patches built in.

In addition to compiling Postfix to support TLS, you must also create and configure TLS certificates. You need both a private key and a public key. The public key is a signed certificate identifying your server. It is validated and digitally signed by a certificate authority (CA), which attests that your certificate does, in fact, identify your system (see sidebar in TLS Certificates Brief Overview). In addition to your own certificates, you must also have the public key of the CA that signed your certificate.

You can register with any of the many CAs to obtain a signed certificate, or you can act as your own CA. The clients connecting to your TLS-enabled server must recognize and acknowledge the CA you use and agree to accept it as an authority to attest to your identity. Generally, it is a fairly simple configuration option in email clients to accept a certificate and have the CA public key added to its list of trusted authorities if it isn't listed already.

TLS Certificates Brief Overview

TLS uses public-key cryptography to allow a client and a server to communicate privately. It also provides assurance that no one has tampered with transmitted information and that the information is not forged because the protocol allows for both the client and server to authenticate each other. Always keep in mind, however, that the benefits of TLS are limited to just the end points of a given TLS connection. What happens to any data before or after it passes between the client and server is not protected by TLS.

Public-key cryptography uses a pair of complementary keys. One can be widely distributed and the other is a secret key. Data encrypted with one key can be decrypted with the other key and vice versa. Others can send you data encrypted with your public key that only you can decrypt with your private one. In most implementations, the private key can be used to create a digital signature of a block of data. The public key can then be used to verify that a particular private key created a given signature.

Moreover, your public key is associated with an identifier, referred to as its *common name* (often the hostname of your server). Others can be sure your server is what it claims to be by comparing the common name associated with its public key against its DNS hostname or a name supplied during connection handshaking. In general, you want everyone to have your public key, but your private key must be guarded at all costs.

Public keys are digitally signed by CAs to create certificates. CAs are usually third-party organizations that are trusted by both sides of the exchange. In theory, the CA's digital signature indicates that it has verified the identity of the public-key holder and attests that this public key belongs to this server. Chapter 13 A public key validated by a CA is often referred to as a signed certificate. Your trust in a certificate should extend only as far as your faith in the CA that signed it. The only assurance that exists with

13.2 TLS Certificates

The TLS patches for Postfix were written using the OpenSSL libraries. The libraries come with command-line tools for managing certificates, which you will need to generate certificates. For Postfix purposes, all of your certificates must be in the PEM format, which is base64 encoded data with some additional header lines. The default output for the OpenSSL tools is PEM, so you won't have to convert any certificates you generate to use with Postfix. By default, the OpenSSL tools are installed below */usr/local/ssl*. The *openssl* command is the utility you'll use most often in managing your certificates.

13.2.1 Becoming a CA

Your server certificates have to be signed by a CA. You can easily set yourself up as a CA to sign your own certificates. The OpenSSL distribution includes a script to configure yourself as a CA. From the SSL home directory, type the following:

```
# misc/CA.pl -newca
```

Answer all of the prompts as requested. This sets up all of the necessary CA files below *./demoCA*. Later, when you issue the command to sign a certificate, the *openssl* command will refer to these root certificates.

13.2.2 Generating Server Certificates

You can use the *openssl* command to generate the public and private keys for your server. From the public key, you create a certificate signing request (CSR) to send to a CA for validation. Once signed, your public certificate can be widely distributed, but your private keys must be carefully guarded. In fact, many applications store encrypted private keys and require a pass phrase to access them. You cannot use encrypted keys with Postfix, however, because different components need read access to the keys as they are started by the *master* daemon.

The OpenSSL distribution includes scripts to help you generate keys and certificate-signing requests, but the scripts encrypt the keys by default. Since you want to leave the keys unencrypted, it's just as easy to use the *openssl* command directly. Execute the following command to create a public and private key to be used with Postfix:

```
$ openssl req -new -nodes -keyout mailkey.pem \
```

Chapter 14. Content Filtering

A content filter is a utility that scans the headers and body of an email message, and usually takes some action based on what it finds. The most common examples are anti-virus and anti-spam programs. Viruses are commonly spread within the contents of email messages, and if you cannot detect spam based on the connecting client or envelope information, you might have better luck by inspecting the actual contents of a message. Filters might change messages, redirect them, respond to them, or tag them for later processing by another tool.

In this chapter we'll look at content filtering at your mail server, although that may not always be your best option for filtering. MTA filtering is appropriate for filtering that should occur with all or nearly all messages. If you need filtering that is configurable by user, the MTA is not the best choice for it. Other types of filtering to consider are:

Mail delivery agent (MDA)

Configurable MDAs such as *procmail* or *sieve* allow users to manage their own delivery configuration files. Generally, MDAs expect your users to edit their own configuration files on the mail server system. If they don't have system accounts, you must provide another means for them to configure their filtering, such as through a web-based application.

Mail user agent (MUA)

You might also consider allowing your users to take advantage of filtering capabilities within their email clients. If their client packages support filtering, this is an excellent way to provide per-user filtering for virtual users that don't have system accounts on your mail server. It has the added advantage of moving processor- and memory-intensive scanning from the server out to multiple clients.

Postfix body and header checks

Postfix body and header checks can provide limited filtering. They cannot be configured by the user, but they are probably the simplest to implement. See Chapter 11 for information about setting them up.

A combination of MTA and MUA filters might make a nice compromise. The MTA filter can tag messages with a value to be read by users' MUA filters. Users can then configure their own filters to accept, reject, or categorize messages based on the tagged value.

An anti-virus filter is an excellent choice for MTA filtering. You can maintain it centrally and block viruses before they even enter your network. Actions that should occur for every message that enters your system are best handled by the MTA.

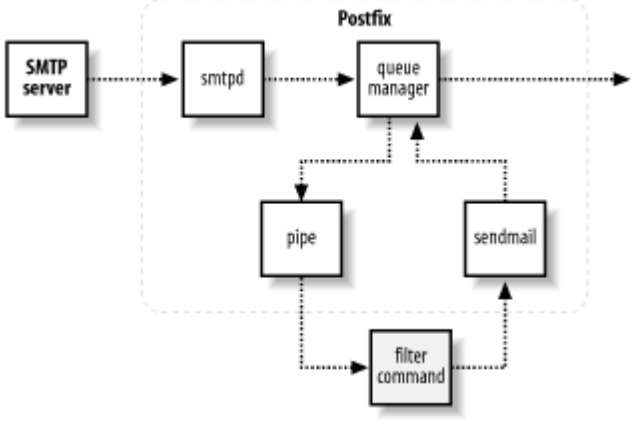
Postfix body and header checks, while powerful, can consider only one line of a message at a time, and they're always applied to all messages. They don't offer a convenient way to set up complex options for rejecting or redirecting messages. Anything more than simple filtering should probably not be handled within a general MTA like Postfix.

14.1 Command-Based Filtering

The simplest way to set up content filtering is to use a program that runs as a command and accepts the contents of a message on its standard input. Postfix delivers messages to your filter command via the pipe mailer. Your filter command performs its checking and then gives the filtered message back to Postfix using the Postfix *sendmail* command.

For this discussion, we'll assume that the filter command operates on mail that comes in through the SMTP daemon but not on mail that is delivered locally (using the *sendmail* command), so that your filter can use *sendmail* to give the message back to Postfix without looping. Figure 14-1 illustrates the path messages follow once you put your filter in place. Rather than passing the message to a delivery agent, the queue manager invokes the filter.

Figure 14-1. Mail-filtering command



Your filter program must be able to accept the message on its standard input and then deliver it to the Postfix *sendmail* command. If you have a filtering program that doesn't handle input and output in this way it should be easy enough to create a shell script wrapper to deal with those details. In the Postfix distribution, the *FILTER_README* file contains an example of such a script.

14.1.1 Configuration

When you configure Postfix to use your filter program, you must specify a user that the program runs as. You should create a pseudoaccount whose sole purpose is to run the filter.

Let's set up an example configuration and assume that you have a filter program named *simple_filt* stored at */usr/local/bin* and that you have created a pseudouser called filter to run it. Edit your *master.cf* file to add an entry for your filter:

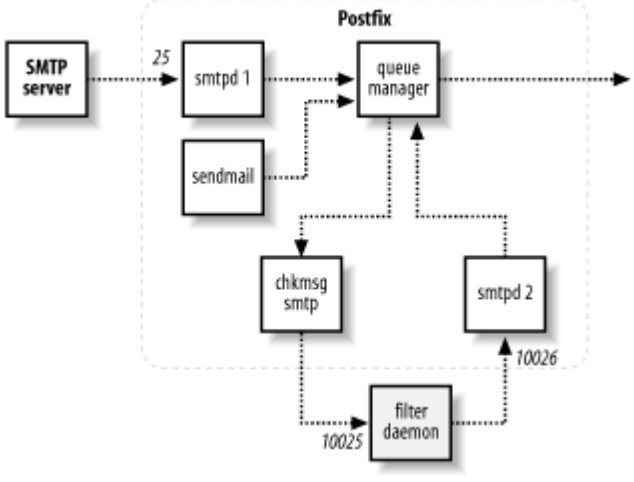
```
filter    unix  -       n       n       -       -       pipe
```

14.2 Daemon-Based Filtering

Daemon-based filtering offers a more advanced architecture over the command-based method with lower cost in I/O and CPU usage. It can provide better error handling than is possible with the command method. If implemented as a resident process, the startup overhead per message is eliminated. A daemon-based content filter can pass email messages back and forth with Postfix using the standard SMTP or LMTP protocol. Such a filter can run as a standalone daemon or it can be started by Postfix if configured to do so in *master.cf*.

In this configuration, we want the content filter to handle all messages, whether delivered locally (via *sendmail*) or to the smtpd daemon. You have to configure Postfix in *master.cf* to use a special smtp client component to deliver the messages to your filter and an additional smtpd daemon to receive messages back from your filter. Figure 14-2 illustrates how a filtered message travels through Postfix to your content filter and back into Postfix for delivery. In this diagram, the filter receives mail via localhost port 10025 from the additional smtp client and submits it back to Postfix via localhost port 10026 to the additional *smtpd* server component.

Figure 14-2. Mail-filtering daemon



If the filter wants to reject a message, it should reply with an SMTP code of 550 along with the reason for the rejection. Otherwise, it should accept the message and perform its operations before passing it back to Postfix. If your filter rejects a message, Postfix bounces it back to the sender address with the message your filter provides.

14.2.1 Configuration

For the purposes of this discussion, I'll assume that you are running a standalone content filter daemon that listens for incoming messages using SMTP. After processing, it sends the message back to Postfix using SMTP. The basic steps to configure this setup are:

1. Create a pseudoaccount for your filter.
2. Install and configure your content filter.
3. Edit *master.cf* to add two additional Postfix components.
4.

14.3 Other Considerations

You can run multiple content filters, if necessary, by chaining them. If, for example, you have both an anti-virus and an anti-spam content filter, simply configure the first one to deliver to the next one rather than immediately back to Postfix. The Postfix configuration doesn't have to change from what's presented here. Only the final filter delivers the message back to Postfix.

Be aware of any email address rewriting that occurs before your filter receives a message. When the filter resubmits a message, if the rewritten address isn't in one of the recipient maps, Postfix will reject it. You may have to turn off address rewriting in your normal SMTP server and configure it instead in your SMTP server that accepts messages back from your filter.

Some filters recommend that you configure them to accept mail in front of your normal MTA, and then they pass the messages on to your MTA after processing. You probably do not want to do this. Postfix is specifically designed to accept messages over an unfriendly network. A content filter is specifically designed to deal with processing the contents of messages and probably isn't optimized for dealing with the load and potential hazards of accepting connections from the outside. Likewise some filters want to handle the final delivery of messages without re-injecting them into Postfix. Again, Postfix offers a lot of flexibility and security in dealing with the final disposition of messages that you might lose by delegating the delivery to another package.

Chapter 15. External Databases

Postfix map files provide an easy and efficient mechanism for the many lookup operations needed when handling email. In some situations, however, it can be more convenient to have the information in a database separate from Postfix. A database can provide a central repository available to many system or network services that need similar information, such as account names and passwords. A database can be useful when redundant systems running Postfix need to share the same configuration information. A central database might also be more convenient when you have multiple people who need access to edit information.

Databases can also slow Postfix performance compared to normal index files. In general, if you don't have a definite need for a database, you're better off with the standard Postfix maps. In many cases you can get the best of both options by storing information in a database and running regular scripts that update your Postfix files from the central data repository. But if your environment requires instant access to revised data, an external database configuration may be your only option.

In this chapter, we'll look at configuring Postfix to work with MySQL and LDAP. (Postfix also has support for PostgreSQL as of Version 2.1.) In either case, Postfix must be compiled with additional libraries to support the mysql and ldap map types. If you are using a prebuilt package, make sure that it has support for the type of database you plan to use. If you built your own Postfix, see Chapter 15 for information on compiling with the additional libraries.

You can easily check if your Postfix installation contains support for LDAP and MySQL with the *postconf -m* command:

```
$ postconf -m
```

15.1 MySQL

MySQL is an open source relational database system that uses Structured Query Language (SQL) for querying and managing its data. You don't have to know SQL to use Postfix with MySQL, but it will help to understand how they interact. Normally, you would use MySQL because you already have a database of information about each user such as a full name, account name, phone numbers, etc. You have to make sure your database includes the information you need to accomplish a particular task with Postfix. A common use is to map an email alias to the local account name. For this to work there must be one database column containing email aliases and another with local account names. Postfix can query your database with the recipient address of an email message as the key to look up the value of the local account for delivery. Any of the Postfix lookup table parameters can work with MySQL queries. You just have to figure out which columns contain the information you need.

15.1.1 MySQL Configuration

MySQL maps are specified like any other map in Postfix. You specify the map type and the file containing the mappings. In the case of MySQL, however, the file you specify is not the lookup map itself, but rather a file that contains configuration information that specifies how to get the desired value from your database:

```
alias_maps = mysql:/etc/postfix/mysql-aliases.cf
```

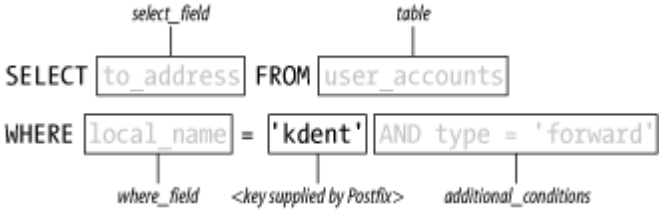
The file *mysql-aliases.cf* contains configuration information that specifies how to get the information from MySQL. The parameters for this file are explained below.

15.1.1.1 MySQL parameters

MySQL parameters provide the information necessary for Postfix to connect to your database server and construct an SQL statement to look up the data it needs. These parameters are placed in a MySQL map configuration file that functions like a Postfix configuration file with blanks and comments ignored. Comments are marked by a # as the first character of a line. You can have as many MySQL configuration files as needed in place of normal Postfix lookup files. All of the MySQL parameters presented here are required except for additional_conditions.

Figure 15-1 shows an SQL statement that Postfix creates using the parameters described.

Figure 15-1. Sample SQL statement



hosts

List of hostnames or IP addresses where a MySQL server is running. You can also indicate a Unix domain socket by preceding a path to a socket with unix:. You should list more than one host or socket only if you have multiple redundant database servers. Each host is tried in the order listed until a successful query can be made. For example:

```
hosts = unix:/tmp/mysql.sock, db.example.com, 192.168.150.15
```


user

15.2 LDAP

LDAP is a protocol that provides access to directories of information. LDAP directories are composed of entries that are organized into hierarchies. You have to understand how LDAP works and how your own directory is organized to use it with Postfix. Many networks are starting to make use of LDAP for user information, which makes it a nice way for Postfix to determine what users and addresses it should accept mail for. If your organization uses an LDAP directory, you can query your existing information for your Postfix configuration.

15.2.1 LDAP Configuration

LDAP maps are specified with the ldap map type and can be listed along with any other maps for a given parameter. Unlike MySQL, LDAP parameters are all listed in *main.cf*. You have to invent a name for the particular LDAP configuration you are creating and specify it with the ldap map type. If you call your LDAP configuration ldapaliases, for example, set your alias maps like this:

```
alias_maps = ldap:ldapaliases
```

The LDAP parameters for this configuration all start with the name you invented followed by the name of the parameter. Thus, the LDAP server is identified by the parameter *name*_server_host, so for the example above, the parameter is called ldapaliases_server_host:

```
ldapaliases_server_host = ldap.example.com
```

The important LDAP parameters are defined below. The complete list is available in the *LDAP_README* file that comes with the Postfix distribution:

*name*_search_base

The base DN from which to start the search. You have to know the naming context for your directory so that you can specify the common container for your entries. Often it is the root of the directory. Example: ldapaliases_search_base = dc=example, dc=com

*name*_scope

The scope of the search. There are three possible options for the scope: sub, base, and one. Your directory hierarchy determines which value you need. The base option is rarely useful. With sub the entire tree under the base is searched, and with one only direct child nodes are searched. The _scope parameter defaults to sub if you don't specify another value. Example: ldapaliases_scope = one

*name*_query_filter

The attributes and values that should form your search filter. The variable %s can be used as a placeholder for the current recipient email address. Example: ldapaliases_query_filter = (mailType=forward)

*name*_result_attribute

The attribute containing the value you want returned for this lookup. You can list multiple attributes in order of preference. Example: ldapaliases_result_attribute = email, rfc822Mailbox.

15.2.2 LDAP Example

Appendix A. Configuration Parameters

This appendix contains an alphabetical listing of parameters normally configured in the Postfix *main.cf* file. The brief descriptions are only meant to give you an idea of the purpose of the parameter. All of the parameters are fully documented in the sample configuration files and manpages that come with the Postfix distribution. This quick reference can point you in the right direction, but you will have to consult the body of this book or the online documentation to understand how each parameter works.

All of the parameters are listed with a type of value that should be assigned to it. Most of the value types are obvious. Those that require some explanation are described here:

Explicit list

The parameter requires one or more items from a specific list of possible values. See the online documentation for a particular parameter to see what the possible values are.

Lookup tables

When a parameter points to lookup tables, the tables are specified with their map type and the table name separated by a colon:

```
transport_map = hash:/etc/postfix/transport
```

Pathname

The complete path to a file.

Template

Some parameter values are specified as strings that contain macros:

```
smtpd_banner = $myhostname ESMTP $mail_name
```

The macros are expanded into their values at the time the parameter is used. See the online documentation to find out what macros are allowed for a particular template parameter.

Time units

Many parameters are specified as an amount of time:

```
queue_run_delay = 1000s
```

They are assigned a value and a time unit abbreviation. Time unit abbreviations are listed in Table A-1. If you leave off the time unit, each time parameter has a default unit that it assumes for the value specified. You can check the online documentation to see what the default unit is for a particular parameter.

A.1 Postfix Parameter Reference

bounce_notice_recipient

"2bounce" is one of several possible error classes. Each class of error can optionally generate an error notice. 2bounce_notice_recipient designates the recipient address for "2bounce" error notices.

Possible values:

email address

Default:

postmaster

Example:

```
2bounce_notice_recipient = postmaster
```

access_map_reject_code

SMTP response code sent when a request is rejected because of an access map restriction.

Possible values:

reply code

Default:

554

Example:

```
access_map_reject_code = 554
```

alias_maps

List of alias databases used by the local delivery agent.

Possible values:

alias maps

Default: hash:/etc/aliases, nis

Appendix B. Postfix Commands

Postfix command-line tools are listed below. Each one is fully documented in a manpage that comes with the Postfix distribution. This appendix is meant to give you an idea of what each command is used for. You should refer to the manpages for complete information about each of the commands:

postalias

Creates or queries alias databases.

postcat

Prints the contents of queue files, allowing administrators to display the text of a message in the queue.

postconf

Displays or changes Postfix parameters. Can display one parameter at a time, or the entire list of parameters.

postdrop

Injects a message into the maildrop directory for delivery by Postfix.

postfix

Starts and stops the Postfix system. Can also be used for other Postfix maintenance, such as checking the configuration and flushing the queue.

postkick

Sends a request to a particular Postfix service. Meant to provide a way for shell scripts to communicate with Postfix services.

postlock

Locks a specified file for exclusive access. Provides a means for shell scripts to use Postfix-compatible locking.

postlog

Appendix C. Compiling and Installing Postfix

The general steps to build Postfix from the source files are to obtain the software bundle, uncompress it, compile it, and install it. The tools you need are common on nearly all distributions of Unix: *gzip*, *tar*, *make*, and a C compiler. Postfix generally expects the GNU gcc compiler, but you can also build it with your platform's native compiler, as long as it supports ANSI C.

C.1 Obtaining Postfix

The official Postfix web site (http://www.postfix.org/) has a download link that displays a list of mirrors from which you can get the software. You should select the mirror that is closest to you. Get the package you want by selecting the "Source code" link under either the Official or Experimental release (see Chapter 1). The examples here assume that you have downloaded a file called *postfix-2.0.10.tar.gz*. If the file you download is different, change the filename accordingly in the commands in the examples.

C.2 Postfix Compiling Primer

Before we move on to the specifics of building Postfix, let's take a look at some of the basics when compiling C code.

The options for a particular build are usually contained within a description file normally called *Makefile*. The *make* utility uses the *Makefile* to determine prerequisites, dependencies, and options to use when building a package. Using this information, *make* calls a compiler to create object files, and then a linker (usually called *ld*) to link them together into executables.

Since the Postfix distribution creates its own *Makefile*, you don't have to worry about editing that (and you shouldn't edit it, since any changes you make would likely get overwritten later). Options that Postfix needs in its *Makefile* are defined in environment variables such as CCARGS. The *INSTALL* file that comes with the Postfix distribution discusses all of the available options. We'll look at some of the more common ones here.

The following environment variables are available to set compile-time options. You should use quotes around the values to retain spaces or other shell metacharacters:

AUXLIBS

Tells the linker where to look for additional libraries that are not in the standard locations. For example, if you build support for an add-on package, you may have to indicate where the libraries are for that package.

CC

Specifies a particular compiler to use. If you want to use a compiler other than the one Postfix selects, set this variable to your compiler. Postfix normally uses *gcc* except on platforms where the native compiler is known to work better. You can check the *makedefs* file to see which compiler Postfix uses by default on your system.

CCARGS

Provides additional arguments to the compiler. If your compiler allows special options or your supporting files are not located in default directories, indicate those options with this variable.

DEBUG

The DEBUG parameter specifies debugging levels for the compiler to use when building the Postfix binaries. Turning on debugging produces extra information that a debugger can use. You can also turn off debugging features completely to build Postfix for a production system.

OPT

C.3 Building Postfix

The source file that you download is in a compressed, tar archive and must be uncompressed using the *gzip* command. In the same directory as the downloaded bundle, type the following:

```
$ gzip -d postfix-2.0.10.tar.gz
```

This uncompresses the file and produces a tar file without the *.gz* extension. Next, untar the file:

```
$ tar -xf postfix-2.0.10.tar
```

This creates a directory called *postfix-2.0.10* below the current directory. Set that directory as your current directory for the rest of the compilation:

```
$ cd postfix-2.0.10
```

If you accept all of the default parameters for building Postfix, compiling is as simple as executing *make* in the top-level directory of the distribution:

```
$ make
```

Executing *make* creates a *Makefile* for your particular platform, which is in turn used to compile Postfix for your system. If you don't need any changes to the default build, you can skip ahead to the Section C.4 section.

C.3.1 Customizing Your Build

The file *makedefs* contains platform-specific information that Postfix uses when configuring the package for your system. If you are curious, you can look at the file to see which parameters Postfix uses for your platform. It identifies your environment and creates the macros and definitions that are used in the *Makefile* for building Postfix on your system. The resultant *Makefile* is invoked by the *make* command which in turn calls your compiler and linker to build the Postfix system. When you type *make* as above, all of this happens automatically, so you don't normally need to worry about this file.

If you want to change any of the parameters for your environment, you can execute the build in two steps. The command *make makefiles* creates a new *Makefile* based on parameters that you specify on the command line. To set specific parameters, simply define variables on the command line. For example, you can use a different compiler from the default that Postfix chooses for your environment. The following example works on an HP-UX system to be sure that *make* finds the correct compiler:

```
$ make makefiles CC="/opt/ansic/bin/cc -Ae"
```

You would, of course, specify the path to your own compiler plus any necessary options. If you need to specify an additional directory for header files on your system, define CCARGS to include your directory:

```
$ make makefiles CCARGS="-I /usr/local/include/"
```

And, of course, you can combine options:

```
$ make makefiles CC="/opt/ansic/bin/cc -Ae" CCARGS="-I /usr/local/include"
```

C.3.2 Modifying Postfix Defaults

Postfix provides a lot of flexibility through its configuration files. Nearly all of Postfix's runtime parameters, including the various directories it uses, can be set in its configuration file except, of course, the location of the configuration file itself. You can change the location by defining DEF_CONFIG_DIR within the CCARGS variable:

```
$ make makefiles CCARGS='-DDEF_CONFIG_DIR=\"/usr/local/etc/postfix\"'
```


The single and double quotation marks and backslashes are important since the value for DEF_CONFIG_DIR should itself be quoted. After compilation, Postfix looks for its *main.cf* configuration file in the directory

C.4 Installation

After you have successfully compiled Postfix, you are ready to install it. You will have to be the root user in order to perform the installation steps.

You need to create a dedicated account that will own the Postfix queue and most of its processes. The account should not permit logins and does not need a shell or a home directory. Use your normal administrative tools to create an account. You can set the password to * and its home directory and shell to invalid paths (something like */bin/false* or */dev/null*). By convention the username should be postfix. The entry in */etc/passwd* should resemble the following:

```
postfix:*:1001:1001:postfix:/no/where:/bin/false
```

You must also create a dedicated group that is not used by any user account, including the postfix account you just created. By convention the group name is postdrop. On most systems you create groups be editing the */etc/group*. Add a line like the following:

```
postdrop:*:1007:
```

Remember that Postfix is a replacement for Sendmail, and in order to maintain compatibility it installs its own *sendmail* binary in place of your existing one. You may want to rename the existing one to save it from being overwritten. Depending on your platform your existing *sendmail* is commonly in */usr/sbin/sendmail* or */usr/lib/sendmail*. You should be able to determine the exact location of your *sendmail* by executing:

```
# whereis sendmail
```

This may list a number of files. You are looking for the binary that has no extension. Once you have found it, rename it to move it out of the way:

```
# mv /usr/sbin/sendmail /usr/sbin/sendmail.orig
```

You will also want to rename two other files that will be replaced by Postfix: *mailq* and *newaliases*. These are commonly found in the */usr/bin* directory, but you can use the *whereis* command to locate them if necessary. These commands might be symbolic links on your systems:

```
# mv /usr/bin/mailq /usr/bin/mailq.orig
```

C.5 Compiling Add-on Packages

This section walks through building Postfix with various add-on packages that are mentioned in the book. Before recompiling Postfix with any additional packages, it is important to first clean up from any previous builds. Execute the following:

```
$ make tidy
```

Now you'll be starting with a clean source tree for your new builds. Each of the examples below takes you through creating a new *Makefile*. Once you've accomplished that, simply type:

```
$ make
```

to rebuild Postfix. If your new build is successful, you can upgrade your currently installed Postfix:

```
# make upgrade
```

If you hadn't previously installed Postfix, use *make install* instead.

C.5.1 Cyrus SASL

See Chapter 12 for information on Cyrus SASL and Postfix. You can download the source for the Cyrus SASL libraries from the Carnegie Mellon web site at http://asg.web.cmu.edu/sasl/sasl-library.html. Note that this book assumes that you are working with SASL Version 2.x libraries. Follow the instructions for building the Cyrus SASL2 libraries. There is also a *SASL_README* file that comes with the Postfix distribution.

One issue when compiling Cyrus SASL that affects Postfix is whether or not to include support for certain Microsoft clients that authenticate using a nonstandard mechanism. The standard plain-text authentication mechanism is identified as PLAIN, but these clients use LOGIN. If you need to support such clients, be sure that the libraries are built with the workaround enabled using the --enable-login option when you run *configure*.

When you install the libraries, be sure to note their location. This example assumes that they are installed in */usr/local/lib* and that the header files are located below */usr/local/include*. If you are using different locations, adjust the examples accordingly.

To build Postfix with SASL support, you must define the USE_SASL_AUTH macro and specify the directories for the libraries and header files. You must also link against the *libsasl2.so* library file. Run *make tidy* if necessary. Build your *Makefile* with the following options:

```
$ make makefiles CCARGS='-DUSE_SASL_AUTH -I/usr/local/include/sasl' \
```

C.6 Common Problems

If you run into problems, check the various README files for information about your build. Frequently, they contain information about problems you might run into. Certainly, if there is a README file specific to your platform, be sure to read it. Some possible problems are mentioned below. Exact messages vary depending on your platform and compiler, so the following are general errors similar to what you might see when building Postfix.

C.6.1 Compile Time

No such file or directory

Make sure that the path to your compiler is correct. If you specified a compiler by setting CC when building your *Makefile* (for example, make makefiles CC="/path"), double-check the path you typed. If the path to your compiler came from the Postfix *makedefs* file, you might need to override it with:

```
$ make makefiles CC="/path/to/your/compiler"
```

Another possibility is to have Postfix call your compiler without a path, assuming its directory is in your environment path:

```
$ make makefiles CC="cc"
```

Could not open source file

Make sure that the path to your include files is correct. The include files are normally stored in */usr/include*. If your system uses a different path for some reason, you will have to specify it with the -I option set in CCARGS:

```
$ make makefiles CCARGS="-I/path/to/include"
```

If you already specified a path with -I double-check your typing.

Unresolved (or undefined) symbol

Make sure that the library paths you specified with the -L option are correct and that you have specified the libraries themselves correctly with the -l option.

Warnings from header files

If you see errors associated with a header file like *mail_conf.h*, you may not be using an ANSI C compiler. Nearly all platforms ship with a compiler that is used to reconfigure the kernel, but they do not all include an ANSI C compiler that you can use for development. You may have to contact your vendor to get an ANSI C compiler if you want to build Postfix. Also, the GNU *gcc* compiler works on nearly all platforms and is available as open source software. If you are using the compiler for HP-UX, you must use the -Ae flag to compile in ANSI mode. Include it in your CCARGS variable:

```
$ make makefiles CCARGS="-Ae"
```

Don't know how to

You have probably lost your *Makefile* or never had one. You can easily create your *Makefile* by executing the

C.7 Wrapping Things Up

You can mix and match any of the options or add on libraries described in this appendix to build Postfix for your environment. If your command line for building the Postfix *Makefile* is getting a little complicated, you should probably create a simple shell script that invokes the options and additional libraries you need. Creating a build script has the added advantage of documenting the options you used when you last built Postfix. Feel free to include plenty of comments to yourself to explain the reasons you are including an option or not, and how you came to that decision. The following is an example of a shell script you might use, although you will certainly need to customize it for your own environment. This example includes all of the add-on libraries we've discussed. You should exclude the ones you don't need:

```
#
```

Appendix D. Frequently Asked Questions

I can't seem to receive messages. What does this error mean: "<test@example.com>: mail for example.com loops back to myself"?

Postfix reports this error when a DNS reply points to your mail server, but Postfix has not been configured to accept mail for the domain. Postfix accepts mail for domains listed in mydestination, relay_domains, virtual_mailbox_domains, virtual_alias_domains, and domains that resolve to IP addresses listed in inet_interfaces and proxy_interfaces. Your domain must be listed in one of these parameters.

When I make changes to configuration files or lookup tables, do I have to reload Postfix?

It depends on the type of file you are changing. Changes in files that Postfix reads into memory at startup require a reload. Examples of such files are *main.cf*, *master.cf*, and any lookup table using regular expressions. DB or DBM files are not read into memory and don't require reloading Postfix when they are changed.

Is there some kind of "include" directive for *main.cf*?

No. Most administrators with complex configurations create a *Makefile* that will *cat* the necessary files together. If you have other regular administrative tasks, add them to your *Makefile* too. Your *Makefile* should have an entry that looks something like this:

```
main.cf: file1 file2 file3
```

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of Postfix: The Definitive Guide is a dove. Doves belong to the class Aves (birds) and the order Columbiformes (doves and pigeons), to which the now-extinct dodo bird (Raphus cucullatus) also belonged. Their family, Columbidae, includes over 300 species of pigeons and doves, including the common rock dove or feral pigeon (Columba livia).

In 1679, the French astronomer Augustin Royer discovered the dove-shaped constellation Columba. A constellation in the southern hemisphere, located near Puppis and Caelum, Columba's stars were originally part of the constellation Canis Major.

Reg Aubry was the production editor and copyeditor, and Matt Hutchinson was the proofreader for Postfix: The Definitive Guide . Colleen Gorman and Claire Cloutier provided quality control. Mary Agner provided production assistance. Ellen Troutman-Zaig wrote the index.

Ellie Volckhausen designed the cover of this book, based on a series design by Edie Freedman. The cover image is an original illustration created by Susan Hart. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font.

David Futato designed the interior layout. This book was converted by Joe Wizda to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read, using Macromedia FreeHand 9 and Adobe Photoshop 6. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Leanne Soylemez and Reg Aubry.

The online edition of this book was created by the Safari production group (John Chodacki, Becki Maisch, and Ellie Cutler) using a set of Frame-to-XML conversion and cleanup tools written and maintained by Erik Ray, Benn Salter, John Chodacki, and Jeff Liggett.

[Team LiB]