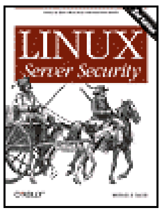
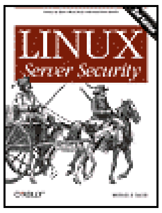


Linux Server Security, 2nd Edition

By Michael D. Bauer

Publisher: O'Reilly
Pub Date: January 2005
ISBN: 0-596-00670-5
Pages: 542

Linux Server Security, 2nd Edition expertly conveys to administrators and c
serious security breaches. It covers both background theory and practical
Linux. Packed with examples, this must-have book lets the good guys stay



Linux Server Security, 2nd Edition

By Michael D. Bauer

Publisher: O'Reilly
Pub Date: January 2005
ISBN: 0-596-00670-5
Pages: 542

- Table of Contents
- Index
- Reviews
- Examples
- Reader Reviews
- Errata
- Academic

Team LiB

◀ PREVIOUS NEXT ▶

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

Dedication

To Felice

Preface

Computer security can be both discouraging and liberating. Once you get past the horror that comes with fully grasping its futility (a feeling identical to the one that young French horn players get upon realizing no matter how hard they practice, their instrument will continue to humiliate them periodically without warning), you realize that there's nowhere to go but up. But if you approach system security with:

- Enough curiosity to learn what the risks are
- Enough energy to identify and take the steps necessary to mitigate (and thus intelligently assume) those risks
- Enough humility and vision to plan for the possible failure of even your most elaborate security measures

you *can* greatly reduce your systems' chances of being compromised. At least as importantly, you can minimize the duration of and damage caused by any attacks that *do* succeed. This book can help, on both counts.

What This Book Is About

Acknowledging that system security is, on some level, futile is my way of admitting that this book isn't really about "Linux server security,"[1] at least not in any absolute sense. Clearly, the only way to make a computer *absolutely* secure is to disconnect it from the network, power it down, repeatedly degauss its hard drive and memory, and pulverize the whole thing into dust. This book contains very little information on degaussing or pulverizing. However, it contains a great deal of practical advice on the following:

[1] My original title was *Attempting to Enhance Certain Elements of Linux System Security in the Face of Overwhelming Odds: Yo Arms Too Short to Box with God*, but this was vetoed by my editor (thanks, Andy!).

- How to think about threats and risks, and the appropriate responses to them
- How to protect publicly accessible hosts via good network design
- How to "harden" a fresh installation of Linux and keep it patched against newly discovered vulnerabilities with a minimum of ongoing effort
- How to make effective use of the security features of some particularly popular and securable server applications
- How to implement some powerful security applications, including Nessus and Snort

In particular, this book is about "bastionizing" Linux servers. The term *bastion host* can legitimately be used several ways, one of which is as a synonym for firewall. (This book *is not* about building Linux firewalls, though much of what I cover can and should be done on firewalls.) My definition of *bastion host* is a carefully configured, closely monitored host that provides restricted but publicly accessible services to nontrusted users and systems. Since the biggest, most important, and least trustworthy public network is the Internet, my focus is on creating Linux bastion hosts for Internet use.

I have several reasons for this seemingly narrow focus. First, Linux has been particularly successful as a server platform: even in organizations that otherwise rely heavily on commercial operating systems such as Microsoft Windows, Linux is often deployed in "infrastructure" roles, such as SMTP gateway and DNS server, due to its reliability, low cost, and the outstanding quality of its server applications.

Second, Linux and TCP/IP, the *lingua franca* of the Internet, go together. Anything that can be done on a TCP/IP network can be done with Linux, and done extremely well, with very few exceptions. There are many, many different kinds of TCP/IP applications, of which I can only cover a subset if I want to do so in depth. Internet server applications are an important subset.

Third, this is my area of expertise. Since the mid-90s my career has focused on network and system security; I've spent a lot of time building Internet-worthy Unix and Linux systems. By reading this book, you will hopefully benefit from some of the experience I've gained along the way.

The Paranoid Penguin Connection

Another reason I wrote this book has to do with the fact that I write the monthly "Paranoid Penguin" security column in *Linux Journal Magazine*. Several years ago, I realized that all my pieces so far had something in common: each was about a different aspect of building bastion hosts with Linux.

By then, the column had gained a certain amount of notoriety, and I realized that there was enough interest in this subject to warrant an entire book on Linux bastion hosts. *Linux Journal* generously granted me permission to adapt my columns for such a book, and under the foolish belief that writing one would amount mainly to knitting the columns together, updating them, and adding one or two new topics, I proposed this book to O'Reilly, and they accepted.

Predictably, the book project was exponentially more work than I could have imagined. I spent a great deal of effort re-researching and expanding all of it, including retesting all examples and procedures. I added entire (lengthy) chapters on topics I hadn't yet covered at all in the magazine, and I more than doubled the size and scope of others. In short, I allowed this to become The Book That Ate My Life in the hope of reducing the number of ugly security surprises in my readers' lives.

The Second Edition

I'd be out of character if I started doing things the smart and easy way, like writing a second edition by simply updating the old material and fixing the errata. No, besides changing the title and updating and revalidating the old material, I've added:

- An all-new chapter on using LDAP for authentication services
- An all-new chapter by Bill Lubanovic on database security
- Lengthy sections in Chapter 9 on LDAP and Cyrus-Imapd, plus an introduction to email encryption
- Comprehensive coverage of the popular *vsftpd* FTP server
- Coverage throughout the book of Fedora Linux

Audience

Who needs to secure their Linux systems? Arguably, anybody who has one connected to a network. This book should therefore be useful both for the Linux hobbyist with a web server in the basement and for the consultant who audits large companies' enterprise systems.

Obviously, the stakes and the scale differ greatly for those two types of users, but the problems, risks, and threats they need to consider have much in common. The same buffer overflow that can be used to "root" a host running "Foo-daemon Version X.Y.Z" is just as much of a threat to a 1,000-host network with 50 Foo-daemon servers as it is to a 5-host network with one.

This book is addressed, therefore, to all Linux system administratorswhether they administer 1 or 100 networked Linux servers, and whether they run Linux for love or for money.

What This Book Doesn't Cover

This book covers general Linux system security, perimeter (Internet-accessible) network security, and server-application security. Specific procedures, as well as tips for specific techniques and software tools, are discussed throughout, and differences between the Red Hat Enterprise Linux, Fedora, SUSE 9, and Debian 3 GNU/Linux distributions are addressed in detail.

This book does *not* cover the following topics explicitly or in detail:

- Linux distributions besides Red Hat, Fedora, SUSE, and Debian, although with regard to application security (which amounts to the better part of the book), this shouldn't be a problem for users of Slackware, Turbolinux, etc.
- Other open source operating systems such as OpenBSD (again, much of what is covered *should* be relevant, especially application security)
- Applications that are inappropriate for or otherwise unlikely to be found on publicly accessible systems (e.g., Samba)
- Desktop (non-networked) applications
- Dedicated firewall systems (this book contains a *subset* of what is required to build a good firewall system)
- Physical security, which admittedly is extremely important but is not in any way unique to Linux systems

Assumptions This Book Makes

While security itself is too important to relegate to the list of "advanced topics" that you'll get around to addressing at a later date, this book does not assume that you are an absolute beginner at Linux or Unix. If it did, it would be twice as long: for example, I can't give a very focused description of setting up *syslog*'s startup script if I also have to explain in detail how the System V *init* system works.

Therefore, you need to understand the basic configuration and operation of your Linux system before my procedures and examples will make much sense. This doesn't mean you need to be a grizzled veteran of Unix who's been running Linux since kernel Version 0.9 and who can't imagine listing a directory's contents without piping it through impromptu *awk* and *sed* scripts. But you should have a working grasp of the following:

- Basic use of your distribution's package manager (*rpm*, *apt-get*, etc.)
- Linux directory system hierarchies (e.g., the difference between */etc* and */var*)
- How to manage files, directories, packages, user accounts, and archives from a command prompt (i.e., without having to rely on X)
- How to compile and install software packages from source
- Basic installation and setup of your operating system and hardware

Notably absent from this list is any specific *application* expertise: most security applications discussed herein (e.g., OpenSSH, Swatch, and Tripwire) are covered from the ground up.

I do assume, however, that with the non-security-specific applications covered in this book, such as Apache and BIND, you're resourceful enough to get any information you need from other sources. In other words, if you're new to these applications, you shouldn't have any trouble following my procedures on how to harden them. But you'll need to consult their respective manpages, HOWTOs, etc. to learn how to fully configure and maintain them.

Organization of This Book

This book provides a comprehensive approach to security by giving you guidelines for securing a system along with configuration details for particular services.

Chapter 1, *Threat Modeling and Risk Management*, introduces the proper attitude and mental habits for thinking securely, including two systematic ways to assess risk: Annualized Loss Expectancies and Attack Trees.

Chapter 2, *Designing Perimeter Networks*, describes where in your network topology to place firewalls and bastion hosts.

Chapter 3, *Hardening Linux and Using iptables*, is a major chapter that shows you how to close up security holes on the operating system level, check your work with nmap and Nessus port scans, create firewalls for servers, and run Bastille.

Chapter 4, *Secure Remote Administration*, covers secure logins, including *ssh* and an introduction to encryption.

Chapter 5, *OpenSSL and Stunnel*, is an in-depth discussion of setting up a certificate authority and creating virtual private network connections.

Chapter 6, *Securing Domain Name Services (DNS)*, gives comprehensive guidelines for securing both BIND and the most popular alternative, djbdns.

Chapter 7, *Using LDAP For Authentication*, introduced OpenLDAP and explains its place in user authentication.

Chapter 8, *Database Security*, covers general considerations for running a database securely, along with details on the MySQL database.

Chapter 9, *Securing Internet Email*, covers the extensive security-related options in Sendmail, Postfix, and Cyrus IMAP. SASL, SMTP AUTH, and email encryption are covered.

Chapter 10, *Securing Web Servers*, is an in-depth approach to the many risks and solutions involved in running Apache, Perl and PHP CGI scripts, and other dynamic features of web sites.

Chapter 11, *Securing File Services*, explains how to configure the ProFTPD and vsftpd FTP servers and how to use *rsync*.

Chapter 12, *System Log Management and Monitoring*, covers the use of syslog and Syslog-ng for logging and Swatch for automated logfile monitoring.

Chapter 13, *Simple Intrusion Detection Techniques*, introduces the complex field of intrusion detection and offers in-depth coverage of Tripwire and Snort.

The Appendix, *Two Complete iptables Startup Scripts*, provides models for creating firewalls.

Conventions Used in This Book

This book uses the following typographical conventions:

Italic

Indicates Unix pathnames, filenames, commands, and packages and program names; Internet addresses, such as domain names and URLs; account usernames; and new terms where they are defined.

`Constant Width`

Indicates command lines and options that should be typed verbatim, as well as names and keywords in system scripts, including commands, parameter names, and variable names.

`Constant Width Bold`

Used in examples and tables to show commands or other text that should be typed literally by the user.

`Constant Width Italic`

Used in examples and tables to show text that should be replaced with user-supplied values.

This icon indicates a tip, suggestion, or general note.

This icon indicates a warning or caution.

Safari® Enabled

When you see a Safari® Enabled icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf. Safari offers a solution that's better than e-Books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it free at http://safari.oreilly.com.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

There is a web page for this book, which lists errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/linuxss2/

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Linux Server Security*, by Michael Bauer. Copyright 2005 O'Reilly Media, Inc., 0-596-00670-5."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

Acknowledgments

For the most part, my writing career has centered on describing how to implement and use software that I didn't write. I am therefore much indebted to and even a little in awe of the hundreds of outstanding programmers who create the operating systems and applications I use and write about. They are the rhinoceroses whose backs I peck for insects.

As if I weren't beholden to those programmers already, I routinely seek and receive first-hand advice and information directly from them. Among these generous souls are Jay Beale of the Bastille Linux project, Ron Forrester of Tripwire Open Source, Balazs "Bazsi" Scheidler of Syslog-ng and Zorp renown, and Renaud Deraison of the Nessus project.

Special thanks go to Dr. Wietse Venema of the IBM T.J. Watson Research Center for reviewing and helping me correct the SMTP chapter. Not to belabor the point, but I find it remarkable that people who already volunteer so much time and energy to create outstanding free software also tend to be both patient and generous in returning email from complete strangers.

Bill Lubanovic wrote the section on djbdns in Chapter 6, *Securing Domain Name Services (DNS)*; all of the new Chapter 8, *Database Security*; and all of Chapter 10, *Securing Web Servers* brilliantly, in my humble opinion. In addition, Bill has taken over and revised Chapter 13, *Simple Intrusion Detection Techniques*. He's brought a great deal of real-world experience, skill, and humor to these four chapters. I could not have finished this book on schedule (and its web security chapter, in particular, would be less convincing!) without Bill's contributions.

Linux Journal and its publisher, Specialized Systems Consultants Inc., very graciously allowed me to adapt a number of my "Paranoid Penguin" columns for inclusion in this book; Chapters Chapter 1 through Chapter 7, plus Chapters Chapter 11, Chapter 12, and Chapter 13 contain (or are descended from) such material. It has been and continues to be a pleasure to write for *Linux Journal*, and it's safe to say that I wouldn't have had enough credibility as a writer to get this book published had it not been for them.

My approach to security lately has been strongly influenced by Yuemei Zhang and Bill Wurster, both of whom have been not only outstanding role models but valued friends. Dr. Martin R. Carmichael's infectious passion for information security has also been a major influence.

It should but won't go without saying that I'm very grateful to Andy Oram and O'Reilly for this opportunity and for their marvelous support, guidance, and patience. The impressions many people have of O'Reilly being stupendously savvy, well organized, technologically superior, and in all ways hip are completely accurate.

A number of technical reviewers also assisted in fact checking and otherwise keeping me honest. Rik Farrow, Bradford Willke, Steve Beaty, Stephen J. Lombardo, Ivan Ristic, and Joshua Ball helped immensely to improve the book's accuracy and usefulness.

In creating and testing code and configuration samples for three different Linux distributions, I benefited enormously from the donation of two copies of VMWareWorkstation 4.5 from VMWare, Inc. Their generosity and the quality of their software are greatly appreciated.

Finally, in the inevitable amorphous list, I want to thank the following valued friends and colleagues, all of whom have aided, abetted, and encouraged me as both a writer and as a "netspook": Dr. Dennis R. Guster at St. Cloud State University; KoniKaye and Jerry Jeschke at Upstream Solutions; Steve Rose at Vector Internet Services (who hired me *way* before I knew anything useful); David W. Stacy of St. Jude Medical; Marty J. Wolf at Bemidji State University; John B. Weaver of the JBW Group, without whose support I honestly could not have finished the second edition; the Reverend Gonzo at Musicscene.org; Richard Vernon and Don Marti at *Linux Journal*; Jay Gustafson of Ingenious Networks; Ray Kaplan, whose talent is surpassed only by his character; brothers-in-arms Tim Shea, Tony Bautts, Wayland Shiu, Nate Duzenberry, Tim Warner, Bob Gleason, and Andy Smith; and, of course, my dizzyingly adept pals Paul Cole, Tony Stieber, and Jeffrey Dunitz.

Chapter 1. Threat Modeling and Risk Management

Since this book is about building secure Linux Internet servers from the ground up, you're probably expecting system-hardening procedures, guidelines for configuring applications securely, and other very specific and low-level information. And indeed, subsequent chapters contain a great deal of this.

But what, really, are we hardening against? The answer to that question is different from system to system and network to network, and in all cases, it changes over time. It's also more complicated than most people realize. In short, threat analysis is a moving target.

Far from a reason to avoid the question altogether, this means that threat modeling is an absolutely essential first step (a recurring step, actually) in securing a system or a network. Most people acknowledge that a sufficiently skilled and determined attacker[1] can compromise almost any system, even if you've carefully considered and planned against likely attack vectors. It therefore follows that if you *don't* plan for even the most plausible and likely threats to a given system's security, that system will be *particularly* vulnerable.

[1] As an abstraction, the "sufficiently determined attacker" (someone theoretically able to compromise any system on any network, outrun bullets, etc.) has a special place in the imaginations and nightmares of security professionals. On the one hand, in practice such people are rare: just like "physical world" criminals, many if not most people who risk the legal and social consequences of committing electronic crimes are fairly predictable. The most likely attackers therefore tend to be relatively easy to keep out. On the other hand, if you *are* targeted by a skilled and highly motivated attacker, especially one with "insider" knowledge or access, your only hope is to have prepared for the worst, and not just the most likely threats.

This chapter offers some simple methods for threat modeling and risk management, with real-life examples of many common threats and their consequences. The techniques covered should give enough detail about evaluating security risks to lend context, focus, and the proper air of urgency to the tools and techniques the rest of the book covers. At the very least, I hope it will help you to think about network security threats in a logical and organized way.

1.1. Components of Risk

Simply put, risk is the relationship between your *assets*, the *vulnerabilities* characteristic of or otherwise applicable to those assets, and *attackers* who wish to steal those assets or interfere with their intended use. Of these three factors, you have some degree of control over assets and their vulnerabilities. You seldom have control over attackers.

Risk analysis is the identification and evaluation of the most likely permutations of assets, known and anticipated vulnerabilities, and known and anticipated types of attackers. Before we begin analyzing risk, however, we need to discuss the components that it comprises.

1.1.1. Assets

Just what are you trying to protect? Obviously you can't identify and evaluate risk without defining precisely what is *at* risk.

This book is about Linux security, so it's safe to assume that one or more Linux systems are at the top of your list. Most likely, those systems handle at least some data that you don't consider to be public.

But that's only a start. If somebody compromises one system, what sort of risk does that entail for other systems on the same network? What sort of data is stored on or handled by these *other* systems, and is any of *that* data confidential? What are the ramifications of somebody tampering with important data versus their simply stealing it? And how will your reputation be impacted if news gets out that your data was stolen?

Generally, we wish to protect data and computer systems, both individually and network-wide. Note that while computers, networks, and data are the information assets most likely to come under direct attack, their being attacked may also affect other assets. Some examples of these are customer confidence, your reputation, and your protection against liability for losses sustained by your customers (e.g., e-commerce-site customer credit card numbers) and for losses sustained by the victims of attacks originating from your compromised systems.

The asset of "nonliability" (i.e., protection against being held legally or even criminally liable as the result of security incidents) is especially important when you're determining the value of a given system's integrity (*system integrity* is defined in the next section).

For example, if your recovery plan for restoring a compromised DNS server is simply to reinstall Red Hat with a default configuration plus a few minor tweaks (IP address, hostname, etc.), you may be tempted to think that that machine's integrity isn't worth very much. But if you consider the inconvenience, bad publicity, and perhaps even legal action that could result from your system being compromised and then used to attack someone else's systems, it may be worth spending some time and effort protecting that system's integrity after all.

In any given case, liability issues may or may not be significant; the point is that you need to think about whether they are and must include such considerations in your threat analysis and threat management scenarios.

1.1.2. Security Goals

Once you've determined what you need to protect, you need to decide what levels and types of protection each asset requires. I call the types *security goals*. They fall into several interrelated categories: data confidentiality and integrity, system integrity, and system/network availability.

1.1.2.1 Data confidentiality

Some types of data need to be protected against eavesdropping and other inappropriate disclosures. *End-user* data such as customer account information, trade secrets, and business communications are obviously important; *administrative* data such as logon credentials, system configuration information, and network topology are sometimes less obviously important but must also be considered.

The ramifications of disclosure vary for different types of data. In some cases, data theft

1.2. Simple Risk Analysis: ALEs

Once you've identified your electronic assets, their vulnerabilities, and some attackers, you may wish to correlate and quantify them. In many environments, it isn't feasible to do so for more than a few carefully selected scenarios. But even a limited risk analysis can be extremely useful in justifying security expenditures to your managers or putting things into perspective for yourself.

One simple way to quantify risk is by calculating Annualized Loss Expectancies (ALEs).[3] For each vulnerability associated with each asset, you must do the following:

[3] Ozier, Will, Micki Krause, and Harold F. Tipton (eds). "Risk Analysis and Management." *Handbook of Information Security Management*, CRC Press LLC.

1. Estimate the cost of replacing or restoring that asset (its Single Loss Expectancy)
2. Estimate the vulnerability's expected Annual Rate of Occurrence
3. Multiply these to obtain the vulnerability's Annualized Loss Expectancy

In other words, for each vulnerability, we calculate:

Single Loss Expectancy (cost)	x	Expected Annual Rate of Occurrences	=	Annualized Loss Expectancy (cost/year)

For example, suppose your small business has an SMTP (inbound email) gateway and you wish to calculate the ALE for Denial of Service (DoS) attacks against it. Suppose further that email is a critical application for your business: you and your nine employees use email to bill clients, provide work estimates to prospective customers, and facilitate other critical business communications. However, networking is not your core business, so you depend on a local consulting firm for email-server support.

Past outages, which have averaged one day in length, tend to reduce productivity by about 1/4, which translates to two hours per day per employee. Your fallback mechanism is a facsimile machine, but since you're located in a small town, this entails long-distance telephone calls and is therefore expensive.

All this probably sounds more complicated than it is; it's much less imposing when expressed in spreadsheet form (Table 1-1).

Table 1-1. Itemized single-loss expectancy

Item description	Estimated cost
Recovery: consulting time from third-party firm (4 hrs @ $150/hr)	$600.00
Lost productivity (2 hrs per 10 workers @ avg. $17.50/hr)	$350.00
Fax paper, thermal (1 roll @ $16.00)	$16.00
Long-distance fax transmissions (20 @ avg. 2 min @ $.25 /min)	$10.00
Total SLE for one-day DoS attack against SMTP server	$976.00

To a small business, $976 per incident is a significant sum; perhaps it's time to contemplate some sort of defense mechanism. However, we're not done yet.

The next thing to estimate is this type of incident's Expected Annual Occurrence (EAO). This is expressed as a number or fraction of incidents per year. Continuing our example, suppose

1.3. An Alternative: Attack Trees

Bruce Schneier, author of *Applied Cryptography*, has proposed a different method for analyzing information security risks: attack trees.[4] An attack tree, quite simply, is a visual representation of possible attacks against a given target. The attack goal (target) is called the *root node*; the various subgoals necessary to reach the goal are called *leaf nodes*.

[4] Schneier, Bruce. "Attack Trees: Modeling Security Threats." *Dr. Dobbs' Journal*: Dec 1999.

To create an attack tree, you must first define the root node. For example, one attack objective might be "Steal ABC Corp.'s Customers' Account Data." Direct means of achieving this could be as follows:

- Obtain backup tapes from ABC's file server.
- Intercept email between ABC Corp. and their customers.
- Compromise ABC Corp.'s file server from over the Internet.

These three subgoals are the leaf nodes immediately below our root node (Figure 1-3).

Figure 1-3. Root node with three leaf nodes

Next, for each leaf node, you determine subgoals that achieve that leaf node's goal. These become the next "layer" of leaf nodes. This step is repeated as necessary to achieve the level of detail and complexity with which you wish to examine the attack. Figure 1-4 shows a simple but more or less complete attack tree for ABC Corp.

Figure 1-4. More detailed attack tree

No doubt, you can think of additional plausible leaf nodes at the two layers in Figure 1-4, and additional layers as well. Suppose for the purposes of our example, however, that this environment is well secured against internal threats (which, incidentally, is seldom the case) and that these are therefore the most feasible avenues of attack for an outsider.

In this example, we see that backup media are most feasibly obtained by breaking into the office. Compromising the internal file server involves hacking through a firewall, but there are three different avenues to obtain the data via intercepted email. We also see that while compromising ABC Corp.'s SMTP server is the best way to attack the firewall, a more direct route to the end goal is simply to read email passing through the compromised gateway.

This is extremely useful information: if this company is considering sinking more money into its firewall, it may decide based on this attack tree that their money and time is better spent securing their SMTP gateway (although we'll see in Chapter 2 that it's possible to do both without switching firewalls). But as useful as it is to see the relationships between attack goals, we're not done with this tree yet.

After an attack tree has been mapped to the desired level of detail, you can start quantifying the leaf nodes. For example, you could attach a "cost" figure to each leaf node that represents your guess at what an attacker would have to spend to achieve that leaf node's particular goal. By adding the cost figures in each attack path, you can estimate relative costs of different attacks. Figure 1-5 shows our example attack tree with costs added (dotted lines indicate attack paths).

Figure 1-5. Attack tree with cost estimates

In Figure 1-5, we've decided that burglary, with its risk of being caught and being sent to jail, is an expensive attack. Nobody will perform this task for you without demanding a significant sum. The same is true of bribing a system administrator at the ISP: even a corruptible ISP

1.4. Defenses

This is the shortest section in this chapter, not because it isn't important but because the rest of the book concerns specific tools and techniques for defending against the attacks we've discussed. The whole point of threat analysis is to determine what level of defenses are called for against the various things to which your systems seem vulnerable.

There are three general means of mitigating risk. A risk, as we've said, is a particular combination of assets, vulnerabilities, and attackers. Defenses, therefore, can be categorized as means of the following:

- Reducing an asset's value to attackers
- Mitigating specific vulnerabilities
- Neutralizing or preventing attacks

1.4.1. Asset Devaluation

Reducing an asset's value may seem like an unlikely goal, but the key is to reduce that asset's value to attackers, not to its rightful owners and users. The best example of this is encryption: all the attacks described in the examples earlier in this chapter (against poor ABC Corp.'s besieged email system) would be made largely irrelevant by proper use of email encryption software.

If stolen email is effectively encrypted (i.e., using well-implemented cryptographic software and strong keys and pass phrases), it can't be read by thieves. If it's digitally signed (also a function of email encryption software), it can't be tampered with either, regardless of whether it's encrypted. (More precisely, it can't be tampered with without the recipient's knowledge.)

A "physical world" example of asset devaluation is a dye bomb: a bank robber who opens a bag of money only to see himself and his loot sprayed with permanent dye will have some difficulty spending that money.

1.4.2. Vulnerability Mitigation

Another strategy to defend information assets is to eliminate or mitigate vulnerabilities. Software patches are a good example of this: every single sendmail bug over the years has resulted in its developers distributing a patch that addresses that particular bug.

An even better example of mitigating software vulnerabilities is "defensive coding"; by running your source code through filters that parse, for example, for improper bounds checking, you can help insure that your software isn't vulnerable to buffer- overflow attacks. This is far more useful than releasing the code without such checking and simply waiting for the bug reports to trickle in.

In short, vulnerability mitigation is simply another form of quality assurance. By fixing things that are poorly designed or simply broken, you improve security.

1.4.3. Attack Mitigation

In addition to asset devaluation and vulnerability fixing, another approach is to focus on attacks and attackers. For better or worse, this is the approach that tends to get the most attention, in the form of firewalls and virus scanners. Firewalls and virus scanners exist to stymie attackers. No firewall yet designed has any intelligence about specific vulnerabilities of the hosts it protects or of the value of data on those hosts, nor does any virus scanner. Their sole function is to minimize the number of attacks (in the case of firewalls, network-based attacks; with virus-scanners, hostile code-based attacks) that succeed in reaching their intended targets.

Access-control mechanisms, such as username/password schemes, authentication tokens, and smart cards, also fall into this category, since their purpose is to distinguish between trusted and untrusted users (i.e., potential attackers). Note, however, that authentication mechanisms can also be used to mitigate specific vulnerabilities (e.g., using SecurID tokens

1.5. Conclusion

This is enough to get you started with threat analysis and risk management. How far you need to go is up to you. When I spoke on this subject recently, a member of the audience asked, "Given my limited budget, how much time can I really afford to spend on this stuff?" My answer was, "Beats me, but I do know that periodically sketching out an attack tree or an ALE or two on a cocktail napkin is better than nothing. You may find that this sort of thing pays for itself." I leave you with the same advice.

Team LiB PREVIOUS NEXT

1.6. Resources

Cohen, Fred et al. "A Preliminary Classification Scheme for Information Security Threats, Attacks, and Defenses; A Cause and Effect Model; and Some Analysis Based on That Model." Sandia National Laboratories: September 1998, http://www.all.net/journal/ntb/cause-and-effect.html

Team LiB PREVIOUS NEXT

Chapter 2. Designing Perimeter Networks

A well-designed perimeter network (the part or parts of your internal network that have direct contact with the outside worlde.g., the Internet) can prevent entire classes of attacks from even reaching protected servers. Equally important, it can prevent a compromised system on your network from being used to attack other systems. Secure network design is therefore a key element in risk management and containment.

But what constitutes a "well-designed" perimeter network? Since perimeter networks always involve firewalls, you might be tempted to think that a well-configured firewall equals a secure perimeter, but there's a bit more to it than that. In fact, there's more than one "right" way to design the perimeter, and this chapter describes several. One simple concept, however, drives all good perimeter network designs: systems that are at a relatively high risk of being compromised should be segregated from the rest of the network. Such segregation is, of course, best achieved (enforced) by firewalls and other network access-control devices.

This chapter, then, is about creating network topologies that isolate your publicly accessible servers from your private systems while still providing those public systems some level of protection. This *isn't* a chapter about how to pull Ethernet cable or even about how to configure firewalls; the latter, in particular, is a complicated subject worthy of its own book (there are many, in fact). But it should give you a start in deciding where to put your servers before you go to the trouble of building them.

By the way, whenever possible, the security of an Internet-connected perimeter network should be designed and implemented *before* any servers are connected to it. It can be extremely difficult and disruptive to change a network's architecture while that network is in use. If you think of building a server as similar to building a house, network design can be considered analogous to urban planning. The latter really must precede the former.

The Internet is only one example of an external network to which you might be connected. If your organization has a dedicated Wide Area Network (WAN) circuit or a Virtual Private Network (VPN) connection to a vendor or partner, the part of your network on which that connection terminates is also part of your perimeter.[1]

[1] Actually, "perimeter" has a much broader definition than it used to. It used to mean "the outer edge of your network," but nowadays it means "any place trusted systems meet untrusted traffic." For example, in many organizations, it's become common for external vendors to support internal systems (e.g., via VPN connections or modems); in that scenario, the perimeter extends as far inside the network as the external vendors go.

Most of what follows in this chapter is applicable to any part of your perimeter network, not just the part that's connected to the Internet.

2.1. Some Terminology

Let's get some definitions cleared up before we proceed. These may not be the same definitions you're used to or prefer, but they're the ones I use in this chapter:

Application gateway (or application-layer gateway)

A firewall or other proxy server possessing application-layer intelligence, e.g., able to distinguish legitimate application behavior from disallowed behavior, rather than dumbly reproducing client data verbatim to servers and vice versa. Each service that is to be proxied with this level of intelligence must, however, be explicitly supported (i.e., "coded in"). Application gateways may use packet filtering or a Generic Service Proxy to handle services for which they have no application-specific awareness.

Bastion host

A system that runs publicly accessible services but is usually not itself a firewall. Bastion hosts are what we put on DMZs (although they can be put anywhere). The term implies that a certain amount of system hardening (see "Hardened system," later in this list) has been done, but sadly, this is not always the case.

DMZ (demilitarized zone)

A network, containing publicly accessible services, that is isolated from the "internal" network proper. Preferably, it should also be isolated from the outside world. (It used to be reasonable to leave bastion hosts outside the firewall but exposed directly to the outside world; as we'll discuss shortly, this is no longer justifiable or necessary.)

Firewall

A system or network that isolates one network from another. This can be a router, a computer running special software in addition to or instead of its standard operating system, a dedicated hardware device, or any other device or network of devices that performs some combination of packet filtering, application-layer proxying, and other network-access control. In this discussion, the term will generally refer to a single multihomed host.

Generic Service Proxy (GSP)

A proxy service (see later in this list) that has no application-specific intelligence. These are nonetheless generally preferable over packet filtering, since proxies provide better protection against TCP/IP stack-based attacks by interrupting and re-initiating each transaction they proxy. Firewalls that use the SOCKS protocol rely heavily on GSPs.

Hardened system

A computer on which all unnecessary services have been disabled or uninstalled, all current OS patches have been applied, and that in general has been configured in as secure a fashion as possible while still providing the services for which it's needed. This is the subject of Chapter 3.

Internal network

What we're trying to protect: end-user systems, servers containing private data, and all other systems to which we do not wish the outside world to initiate connections. This is also called the "protected" or "trusted" network.

Multihomed host

Any computer having more than one logical or physical network interface (not counting loopback interfaces).

Packet filtering

Inspecting the IP headers of packets and passing or dropping them based primarily on

2.2. Types of Firewall and DMZ Architectures

In the world of expensive commercial firewalls (the world in which I earn my living), the term "firewall" nearly always denotes a single computer or dedicated hardware device with multiple network interfaces. This definition can apply not only to expensive rack-mounted behemoths, but also to much lower-end solutions: network interface cards are cheap, as are PCs in general.

This is different from the old days, when a single computer typically couldn't keep up with the processor overhead required to inspect all ingoing and outgoing packets for a large network. In other words, routers, not computers, used to be one's first line of defense against network attacks.

This is no longer the case. Even organizations with high-capacity Internet connections typically use a multihomed firewall (whether commercial or open source-based) as the primary tool for securing their networks. This is possible thanks to Moore's law, which has provided us with inexpensive CPU power at a faster pace than the market has provided us with inexpensive Internet bandwidth. It's now feasible for even a relatively slow PC to perform sophisticated checks on a full T1's-worth (1.544 Mbps) of network traffic.

2.2.1. The "Inside Versus Outside" Architecture

The most common firewall architecture one tends to see nowadays is the one illustrated in Figure 2-1. In this diagram, we have a packet-filtering router that acts as the initial, but not sole, line of defense. Directly behind this router is a "proper" firewallin this case, a Sun SparcStation running, say, Debian Linux with iptables. There is no direct connection from the Internet or the "external" router to the internal network; all traffic to or from it must pass through the firewall.

Figure 2-1. Simple firewall architecture

In my opinion, all external routers should use some level of packet filtering, a.k.a. "Access Control Lists" in the Cisco lexicon. Even when the next hop inwards from such a router is a sophisticated firewall, it never hurts to have redundant enforcement points. In fact, when several Check Point vulnerabilities were demonstrated at a recent Black Hat Briefings conference, no less than a Check Point spokesperson mentioned that it's foolish to rely solely on one's firewall, and he was right. At the very least, your Internet-connected routers should drop packets with non-Internet-routable source or destination IP addresses, as specified in RFC 1918 (ftp://ftp.isi.edu/in-notes/rfc1918.txt), since such packets may safely be assumed to be "spoofed" (forged).

What's missing or wrong about Figure 2-1? (I said this architecture is common, not perfect!) Public services such as SMTP (email), Domain Name Service (DNS), and HTTP (WWW) must either be sent through the firewall to internal servers or hosted on the firewall itself. Passing such traffic to an internal server doesn't directly expose other internal hosts to attack, but it does magnify the consequences of the internal server being compromised.

While hosting public services on the firewall isn't necessarily a bad idea on the face of it (what could be a more secure server platform than a firewall?), the performance issue should be obvious: the firewall should be allowed to use all its available resources for inspecting and moving packets.

Furthermore, even a painstakingly well-configured and patched application can have unpublished vulnerabilities. (All vulnerabilities start out unpublished.) The ramifications of such an application being compromised on a firewall are frightening. Performance and security, therefore, are impacted when you run any service on a firewall.

Where, then, to put public services so that they don't directly or indirectly expose the internal network and don't hinder the firewall's security or performance? Answer: in a DMZ (demilitarized zone) network.

2.2.2. The "Three-Homed Firewall" DMZ Architecture

2.3. Deciding What Should Reside on the DMZ

Once you've decided where to put the DMZ, you need to decide precisely what's going to reside there. My advice is to put *all* publicly accessible services in the DMZ.

Too often I encounter organizations in which one or more crucial services are "passed through" the firewall to an internal host despite an otherwise strict DMZ policy; frequently, the exception is made for MS-Exchange or some other application that is not necessarily designed with Internet-strength security to begin with and hasn't been hardened even to the extent that it could be.

But the one application passed through in this way becomes the hole in the dike: all it takes is one buffer-overflow vulnerability in that application for an unwanted visitor to gain access to all hosts reachable by that host. It is far better for that list of hosts to be a short one (i.e., DMZ hosts) than a long (and critical!) one (i.e., all hosts on the internal network). This point can't be stressed enough: the real value of a DMZ is that it allows us to better manage and contain the risk that comes with Internet connectivity.

Furthermore, the person who manages the passed-through service might be different from the one who manages the firewall and DMZ servers, and he might not be quite as security-minded. If for no other reason, all public services should go on a DMZ so that they fall under the jurisdiction of an organization's most security-conscious employees; in most cases, these are the firewall/security administrators.

But does this mean corporate email, DNS, and other crucial servers should all be moved from the inside to the DMZ? Absolutely not! They should instead be "split" into internal and external services. (This is assumed to be the case in Figure 2-2).

DNS, for example, should be split into "external DNS" and "internal DNS": the external DNS zone information, which is propagated out to the Internet, should contain only information about publicly accessible hosts. Information about other, nonpublic hosts should be kept on separate "internal DNS" zone lists that can't be transferred to or seen by external hosts.

Similarly, internal email (i.e., mail from internal hosts to other internal hosts) should be handled strictly by internal mail servers, and all Internet-bound or Internet-originated mail should be handled by a DMZ mail server, usually called an SMTP gateway. (For more specific information on Split-DNS servers and SMTP gateways, as well as how to use Linux to create secure ones, see Chapter 6 and Chapter 9, respectively.)

Thus, almost any service that has both "private" and "public" roles can and should be split in this fashion. While it may seem like a lot of added work, it need not be, and, in fact, it's liberating: it allows you to optimize your internal services for usability and manageability while optimizing your public (DMZ) services for security and performance. (It's also a convenient opportunity to integrate Linux, OpenBSD, and other open source software into otherwise commercial-software-intensive environments.)

Needless to say, any service that is strictly public (i.e., not used in a different or more sensitive way by internal users than by the general public) should reside solely in the DMZ. In summary, public services, including the public components of services that are also used on the inside, should be split, if applicable, and hosted in the DMZ.

The primary exception to this rule is databases used by web applications: it isn't a good idea to store critical data in untrusted networks such as DMZs, so the best place for databases is the internal network. The tradeoff is that you must then allow inbound queries from your DMZed web servers to your internal database servers, but it's possible to mitigate this risk through careful design and hardening of those servers.

2.4. Allocating Resources in the DMZ

So everything public goes in the DMZ. But does each service need its own host? Can any of the services be hosted on the firewall itself? Should one use a hub or a switch on the DMZ?

The last question is the easiest: with the price of switched ports decreasing every year, switches are preferable on any LAN, and especially so in DMZs. Switches are superior in two ways. From a security standpoint, they're better because it's a bit harder to "sniff" or eavesdrop traffic not delivered to one's own switch port.

Wireless Local Area Networks and Firewalls

Wireless Local Area Networks (WLANs) are increasingly popular, due to their convenience and their low cost (compared to running cable and terminating it to data jacks). But network security professionals nearly unanimously agree that WLAN segments should not be connected directly to trusted/internal networks; they should instead be set up as DMZ networks separated both from the internal network and from other (wired) DMZs by a firewall.

Why? The main reason is because wireless networking is a radio technology: all network traffic in a WLAN is broadcast over radio waves that can be trivially eavesdropped by unauthorized passersby. Besides the obvious privacy problem, this eavesdropping exposure also makes it easier for an attacker to connect to and pretend to be a legitimate user of a WLAN.

Emerging WLAN technologies such as WPA may effectively and transparently encrypt all traffic to mitigate eavesdropping exposures, but as of this writing, the predominant WLAN technology is still 802.11b, a.k.a. "WiFi," typically implemented without WPA (which is backward-compatible with 802.11b). Although 802.11b natively supports encryption via the "Wired Equivalent Privacy" protocol, WEP is not trustworthy: it was found to have fatal flaws very soon after its details were made public.

Even if you use 128-bit WEP keys (the maximum key length WEP supports), an attacker with WEP-cracking software needs only to capture a few hours' worth of your 802.11b WLAN traffic to crack its WEP key and read all your WLAN packets at will (and, potentially, to connect to your WLAN).

Isolating a WLAN segment outside of a firewall mitigates the exposure to unauthorized access to the network, but what about the exposure of data confidentiality? My best advice is not only to DMZ your WLAN but also to run VPN software or to use only encrypted services such as SSH, HTTPS, etc. on it (*in addition* to using 128-bit WEP).

(Unfortunately, this isn't as true as it once was: there are a number of ways that Ethernet switches can be forced into "hub" mode or otherwise tricked into copying packets across multiple ports. Still, some work, or at least knowledge, is required to sniff across switch ports.)

One of our assumptions about DMZ hosts is that they are more likely to be attacked than internal hosts. Therefore, we need to think not only about how to prevent each DMZed host from being compromised, but also what the consequences might be if it is. One possible consequence is the attacker using it to sniff other traffic on the DMZ. We like DMZs because they help isolate publicly accessible hosts, but that does *not* mean we want those hosts to be easier to attack.

Switches also provide better performance than hubs: most of the time, each port has its own chunk of bandwidth rather than sharing one big chunk with all other ports. Note, however, that each switch has a *backplane* that describes the actual volume of packets the switch can handle: a 10-port 100 Mbps hub can't really process 1000 Mbps if it has an 800 Mbps backplane. Nonetheless, even low-end switches disproportionately outperform comparable hubs.

2.5. The Firewall

Naturally, you need to do more than create and populate a DMZ to build a strong perimeter network. What ultimately distinguishes the DMZ from your internal network is your firewall.

Your firewall (or firewalls) provides the first and last word as to which traffic may enter and leave each of your networks. Although it's a mistake to mentally elevate firewalls to a panacea, which can lead to complacency and thus to bad security, it's imperative that your firewalls are carefully configured, diligently maintained, and closely watched.

As I mentioned earlier, in-depth coverage of firewall architecture and specific configuration procedures are beyond the scope of this chapter. What we *will* discuss are some essential firewall concepts and some general principles of good firewall construction.

2.5.1. Types of Firewall

In increasing order of strength, the three primary types of firewall are the simple packet filter, the so-called "stateful" packet filter, and the application-layer proxy. Most packaged firewall products use some combination of these three technologies.

2.5.1.1 Simple packet filters

Simple packet filters evaluate packets based solely on IP headers (Figure 2-5). Accordingly, this is a relatively fast way to regulate traffic, but it is also easy to subvert. Source-IP spoofing attacks generally aren't blocked by packet filters,[2] and since allowed packets are literally passed through the firewall (without being rewritten in any way), packets with "legitimate" IP headers but dangerous data payloads, as in buffer-overflow attacks, can often be sent intact to "protected" targets.

[2] Unless the packet filter uses "interface rules" that filter packets based on which network interface they arrive on, rather than solely based on IP header.

Figure 2-5. Simple packet filtering

An example of an open source packet-filtering software package is Linux 2.2's *ipchains* kernel modules (superseded by Linux 2.4's *netfilter*/*iptables*, which is a stateful packet filter). In both the commercial and open source worlds, simple packet filters are increasingly rare: nowadays all major firewall products and packages have some degree of state-tracking ability.

2.5.1.2 Stateful packet filtering

Stateful packet filtering comes in two flavors: generic and application-aware, notably Check Point. Let's discuss the generic type first.

At its simplest, the term refers to the tracking of TCP connections, beginning with the "three-way handshake" (SYN, SYN/ACK, ACK), which occurs at the start of each TCP transaction and ends with the session's last packet (a FIN or RST). Most packet-filtering firewalls now support some degree of low-level connection tracking.

Typically, after a stateful packet-filtering firewall verifies that a given transaction is allowable (based on source/destination IP addresses and ports), it monitors this initial TCP handshake. If the handshake completes within a reasonable period of time, the TCP headers of all subsequent packets for that transaction are checked against the firewall's "state table" and passed until the TCP session is closedthat is, until one side or the other closes it with a FIN or RST. (See Figure 2-6.) Specifically, each packet's source IP address, source port, destination IP address, destination port, and TCP sequence numbers are kept track of.

Figure 2-6. Stateful packet filtering

This has several important advantages over simple (stateless) packet filtering. The first is bidirectionality: without some sort of connection-state tracking, a packet filter isn't really

Chapter 3. Hardening Linux and Using iptables

There's tremendous value in isolating your bastion (Internet-accessible) hosts in a DMZ network, protected by a well-designed firewall and other external controls. And just as a good DMZ is designed assuming that sooner or later, even firewall-protected hosts may be compromised, good bastion server design dictates that each host should be hardened as though there were *no* firewall at all.

Obviously, the bastion-host services to which your firewall allows access must be configured as securely as possible and kept up to date with security patches. But that isn't enough: you must also secure the bastion host's operating-system configuration and disable unnecessary servicesin short, "bastionize" or "harden" it as much as possible.

If you don't do this, you won't have a bastion server: you'll simply have a server behind a firewallone that's at the mercy of the firewall and the effectiveness of its own applications' security features. But if you do bastionize it, your server can defend itself should some other host in the DMZ be compromised and used to attack it. (As you can see, pessimism is an important element in risk management!)

Hardening a Linux system is not a trivial task: it's as much work to bastionize Linux as Solaris, Windows, and other popular operating systems. This is a natural result of having so many different types of software available for these OSes, and at least as much variation between the types of people who use them.

Unlike many other OSes, however, Linux gives you extremely granular control over system and application behavior, from a high level (application settings, user interfaces, etc.) to a very low level, even as far down as the kernel code itself. Linux also benefits from lessons learned over the three-decade history of Unix and Unix-like operating systems. Unix security is extremely well understood and well documented. Furthermore, over the course of those 30-plus years, many powerful security tools have been developed and refined, including *chroot*, *sudo*, TCPwrappers, Tripwire, and *shadow*.

This chapter lays the groundwork for much of what follows. Whereas most of the rest of this book is about hardening specific applications, this chapter covers system-hardening principles and specific techniques for hardening the core operating system.

3.1. OS Hardening Principles

Operating-system hardening can be time consuming and even confusing. Like many OSes designed for a wide range of roles and user levels, Linux has historically tended to be "insecure by default": most distributions' default installations are designed to present the user with as many preconfigured and active applications as possible. Therefore, securing a Linux system not only requires you to understand the inner workings of your system; you may also have to undo work others have done in the interest of shielding you from those inner workings!

Having said that, the principles of Linux hardening and OS hardening in general can be summed up by a single maxim: "That which is not explicitly permitted is forbidden." As I mentioned in the previous chapter, this phrase was coined by Marcus Ranum in the context of building firewall rules and access-control lists. However, it scales very well to most other information security endeavors, including system hardening.

Another concept originally forged in a somewhat different context is the Principle of Least Privilege. This was originally used by the National Institute of Standards and Technology (NIST) to describe the desired behavior of the "Role-Based Access Controls" it developed for mainframe systems: "a user [should] be given no more privilege than necessary to perform a job" (http://hissa.nist.gov/rbac/paper/node5.html).

Nowadays people often extend the Principle of Least Privilege to include applications; no application or process should have more privileges in the local operating environment than it needs to function. The Principle of Least Privilege and Ranum's maxim sound like common sense (they *are*, in my opinion). As they apply to system hardening, the real work stems from these corollaries:

- Install only necessary software; delete or disable everything else.
- Keep all system and application software painstakingly up to date, at least with security patches, but preferably with *all* package-by-package updates.
- Delete or disable unnecessary user accounts.
- Don't needlessly grant shell access: */bin/false* should be the default shell for *nobody*, *guest*, and any other account used by services, rather than by an individual local user.
- Allow each service (networked application) to be publicly accessible only by design, never by default.
- Run each publicly accessible service in a *chrooted* filesystem (i.e., a subset of */*).
- Don't leave any executable file needlessly set to run with superuser privileges, i.e., with its *SUID* bit set (unless owned by a sufficiently nonprivileged user).
- In general, avoid using *root* privileges unnecessarily, and if your system has multiple administrators, delegate *root*'s authority via *sudo*.
- Configure logging and check logs regularly.
- Configure every host as its own firewall; i.e., bastion hosts should have their *own* packet filters and access controls in addition to (but *not* instead of) the firewall's.
- Check your work now and then with a security scanner, especially after patches and upgrades.
- Understand and use the security features supported by your operating system and applications, *especially* when they add redundancy to your security fabric.
- After hardening a bastion host, document its configuration so it may be used as a baseline for similar systems and so you can rebuild it quickly after a system compromise or failure.

All of these corollaries are ways of implementing and enforcing the Principle of Least Privilege on a bastion host. We'll spend most of the rest of this chapter discussing each in depth with specific techniques and examples. We'll end the chapter by discussing Bastille Linux, a handy

3.2. Automated Hardening with Bastille Linux

The last tool we'll explore in this chapter is Bastille. You might be wondering why I've saved this powerful hardening utility for last: doesn't it automate many of the tasks we've just covered? It does, but with two caveats.

First, the Linux version of Bastille remains somewhat Red Hat-centric. On the one hand, Debian 3.0 includes a deb package for Bastille 1.3, which seems to work pretty well. On the other hand, the Bastille 2.03 RPM included with SUSE 9.0 Enterprise Linux reportedly yields uneven results (though if you're a SUSE user, I certainly encourage you to try it out and provide feedback to the Bastille team). So Bastille still works best if you run a distribution derived from Red Hat, specifically Red Hat itself, Mandrake, or Immunix.

Second, even if you do run a supported distribution, it's extremely important that you use Bastille as a tool rather than a crutch. There's no good shortcut for learning enough about how your system works to secure it.

The Bastille guys (Jay Beale and Jon Lasser) are at least as convinced of this as I am: Bastille has a remarkable focus on educating its users.

3.2.1. Background

Bastille Linux is a powerful set of Perl scripts that both secure Linux systems and educate their administrators. It asks clear, specific questions about your system that allow it to create a custom security configuration. It also explains each question in detail so that by the time you've finished a Bastille session, you've learned quite a bit about Linux/Unix security. If you already understand system security and are interested only in using Bastille to save time, you can run Bastille in an "explain less" mode that asks all the same questions but skips the explanations.

3.2.1.1 How Bastille came to be

The original goal of the Bastille team (led by Jon Lasser and Jay Beale) was to create a new secure Linux distribution based on Red Hat. The quickest way to get their project off the ground was to start with a normal Red Hat installation and then to "Bastille-ify" it with Perl scripts.

Before long, the team had decided that a set of hardening scripts used on different distributions would be less redundant and more flexible than an entirely new distribution. Rather than moving away from the script approach altogether, the Bastille team has instead evolved the scripts themselves.

The Perl scripts comprising Bastille Linux are quite intelligent and make fewer assumptions about your system than they did when Bastille was used only on fresh installations of Red Hat. Your system needn't be a "clean install" for Bastille to work: it transparently gleans a lot of information about your system before making changes to it.

3.2.2. Obtaining and Installing Bastille

To get the latest version of Bastille Linux, point your web browser to http://www.bastille-linux.org/. This page contains links to the Bastille packages and also contains complete instructions on how to install them and the Perl modules that Bastille requires. Unlike earlier versions, Bastille 2.0 is now distributed as a single RPM in addition to its traditional source-code tarball.

In addition to Bastille itself, RPM-based Linux[6] users will need either perl-Tk or perl-Curses, depending on whether you intend to run Bastille in text-console or X Window mode. Since not all versions of all RPM-based distributions include these packages, the Bastille team maintains a chart that recommends the proper packages to use for various versions of Red Hat and Mandrake Linux, available at http://www.bastille-linux.org/perl-rpm-chart.html.

[6] Except Fedora, which as of this writing isn't yet supported, but it may be by the time you read this.

If you run Debian, you can find the deb package *bastille* in the *admin* group on your Debian installation media or your favorite Debian mirror site. As befits its age, Debian 3.0 (*stable*) uses Bastille v1.3, but the *testing* and *unstable* versions use the much newer Bastille v2.1

Chapter 4. Secure Remote Administration

Your server is bastionized, it resides in a firewall-protected DMZ network, and its services are fully patched and configured for optimal security. You've just installed it in a server room, which is monitored by surly armed guards and accessible only after peering into a retinal scanner and submitting to a body cavity search. Not that you plan to visit the system in person, though; it'll be no problem to perform your administrative duties from the comfort of your office, thanks to good old Telnet.

What's wrong with this picture?

4.1. Why It's Time to Retire Cleartext Admin Tools

TCP/IP network administration has never been simple. And yet, many of us remember a time when connecting a host to "the network" meant one's local area network (LAN), which itself was unlikely to be connected to the Internet (originally the almost exclusive domain of academia and the military) or any other external network.

Accordingly, the threat models that network and system administrators lived with were a little simpler than they are now: external threats were of much less concern then. Which is not to say that internal security is either simple or unimportant; it's just that there's generally less you can do about it.

In any event, in the old days, we used *telnet*, *rlogin*, *rsh*, *rcp*, and the X Window System to administer our systems remotely, because of the aforementioned lesser-threat model and because today's GUI-powered, user-friendly packet sniffers (which can be used to eavesdrop the passwords and data that these applications transmit unencrypted) didn't yet exist.

This is not so any more. Networks are bigger and more likely to be connected to the Internet, so packets are therefore more likely to pass through untrusted bandwidth. Furthermore, nowadays, even relatively unsophisticated users are capable of using packet sniffers and other network-monitoring tools, most of which now sport graphical user interfaces and educational help screens. "Hiding in plain sight" is no longer an option.

None of this should be mistaken for nostalgia. Although in olden times, networking may have involved fewer and less frightening security ramifications, there were far fewer interesting things you could do on those early networks. With increased flexibility and power comes complexity; with complexity comes increased opportunity for mischief.

The point is that *cleartext username/password authentication is obsolete*. (So is cleartext transmission of any but the most trivial data, and, believe me, very little in an administrative session isn't fascinating to prospective system crackers.) It's simply become too easy to intercept and view network packets.

But if *telnet*, *rlogin*, *rsh*, and *rcp* are out, what *should* one use? There *is* a convenient yet secure way to administer Unix systems from afar: it's called the Secure Shell.

4.2. Secure Shell Background and Basic Use

A few years ago, Finnish programmer Tatu Ylönen created a terrifically useful application called the Secure Shell, or SSH. SSH is a suite of tools that roughly corresponds to Sun's *rsh*, *rcp*, and *rlogin* commands, but with one very important difference: paranoia. SSH lets you do everything *rsh*, *rcp*, and *rlogin* do, using your choice of libertarian-grade encryption and authentication methods.

OpenSSH, a 100% free and open source outgrowth of the OpenBSD project, has very rapidly become the preferred version of SSH for open source Unices; as of this writing, the latest releases of Red Hat, Debian, and SUSE Linux all ship with binary packages of OpenSSH.

> *SSH v1.x* and *SSH Protocol v1* refer to SSH's software release and protocol, respectively, and are not really synonymous. But since the package and protocol major version numbers *roughly* correspond, from here on, I'll use *SSH v1x* to refer to RSA-based versions of SSH/OpenSSH and *SSH v2x* to refer to versions that support both RSA and DSA.

4.2.1. How SSH Works

Secure Shell works very similarly to Secure Sockets Layer web transactions (it's no coincidence that the cryptographical functions used by OpenSSH are provided by OpenSSL, a free version of Netscape's Secure Sockets Layer source-code libraries). Both can set up encrypted channels using generic *host keys* or with published credentials (digital certificates) that can be verified by a trusted certificate authority (such as VeriSign). Public-key cryptography is discussed in more depth later in this chapter, but here's a summary of how OpenSSH builds secure connections.

First, the client and the server exchange (public) host keys. If the client machine has never encountered a given public key before, both SSH and most web browsers ask the user whether to accept the untrusted key. Next, they use these public keys to negotiate a session key, which is used to encrypt all subsequent session data via a block cipher such as Triple-DES (3DES), blowfish, or IDEA.

> As its name implies, a session key is created specifically for a given session and is not used again after that session closes. Host and user keys, however, are static. You might wonder, why not just use host or user keys to encrypt everything? Because the algorithms used in public-key cryptography are slow and CPU-intensive. Why not use the same session key for multiple sessions? Because unique session keys require more work for an attacker who attempts to crack multiple sessions.

As with typical SSL connections, this initial round of key exchanging and session-key negotiation is completely transparent to the end user. Only after the encrypted session is successfully set up is the end user prompted for logon credentials.

By default, the server attempts to authenticate the client using RSA or DSA certificates (key pairs). If the client (user) has a certificate recognized by the server, the user is prompted by his client software for the certificate's private-key passphrase; if entered successfully, the certificate is used by the SSH client and server to complete a challenge-response authentication, which proves to the server that the client possesses the private key that corresponds to a public key registered with the server. At no point is the private key itself, its passphrase, or any other secret data sent over the network.

Also by default, if RSA/DSA authentication fails or if there is no client certificate to begin with, the remote server prompts the user for a standard Unix username/password combination that is valid for the remote system. Remember, an encrypted session has already been established between client and server, so this username/password combination, while easier to subvert or guess than certificate-based authentication, is at least encrypted prior to being

4.3. Intermediate and Advanced SSH

Although most users use *ssh* and *scp* for simple logins and file transfers, respectively, this only scratches the surface of what SSH can do. Next, we'll examine the following:

- How RSA and DSA keys can be used to make SSH transactions even more secure.
- How *null-passphrase* keys can allow SSH commands to be included in scripts.
- How to cache SSH credentials in RAM to avoid unnecessary authentication prompts.
- How to tunnel other TCP services through an encrypted SSH connection.

SSH and Perimeter Security

Secure Shell is obviously the best way to administer all your servers from a single system, especially if that system is an administrative workstation on your internal network. But is it a good idea to allow external hosts (e.g., administrators' personal/home systems) to have SSH access, passing through your firewall to hosts in the DMZ or even the internal network?

In my opinion, this is usually a bad idea. History has shown us that Secure Shell (both commercial and free versions) is prone to the same kinds of vulnerabilities as other applications: buffer-overflow exploits, misconfiguration, and plain old bugs. Ironically, the same flexibility and power that make SSH so useful also make a compromised Secure Shell daemon a terrifying thing indeed.

Therefore, if you absolutely must have the ability to administer your firewalled systems via untrusted networks, I recommend you use a dedicated VPN tool such as FreeS/WAN to connect to an *access point* in your DMZ or internal networke.g., your administrative workstation. Run SSH on *that* system to connect to the servers you need to administer. An access point adds security even if you use SSH, rather than a dedicated VPN tool, to connect to it; it's the difference between allowing inbound SSH to all your servers or to a single system.

In either case, it should go without saying that your access point must be well hardened and closely monitored.

4.3.1. Public-Key Cryptography

A complete description of public-key cryptography (or *PK crypto*) is beyond the scope of this chapter. If you're completely unfamiliar with PK crypto, I highly recommend the RSA Crypto FAQ (available at http://www.rsasecurity/rsalabs/faq/) or, even better, Bruce Schneier's excellent book, *Applied Cryptography* (Wiley).

For our purposes, it's enough to say that in a public-key scheme (illustrated in Figure 4-1), each user has a pair of keys. Your private key is used to sign things digitally and to decrypt things that have been sent to you. Your public key is used by your correspondents to verify things that have allegedly been signed by you and to encrypt data that they want only you to be able to decrypt.

Figure 4-1. Public-key cryptography

Along the bottom of Figure 4-1, we see how two users' key pairs are used to sign, encrypt, decrypt, and verify a message sent from one to the other. Note that Bob and Alice possess copies of each other's public keys, but both keep their private key secret.

As we can see, the message's journey includes four different key actions:

1. Bob signs a message using his private key.
2. Bob encrypts it using Alice's public key. (Aside from the fact that Bob has probabl

Chapter 5. OpenSSL and Stunnel

This chapter falls both technologically and literally between the behind-the-scenes and the service-intensive parts of the book: it's about OpenSSL, which provides encryption and authentication mechanisms to many of the tools covered herein. OpenSSH, Apache, OpenLDAP, BIND, Postfix, and Cyrus IMAP are just a few of the applications that depend on OpenSSL.

OpenSSL, however, is an extremely complicated technology, and to do it full justice would require a dedicated book (one such book is *Network Security With OpenSSL* (O'Reilly)). My approach with this chapter, therefore, is to show how to use OpenSSL in a particular context: wrapping otherwise unencrypted TCP services in encrypted SSL "tunnels" via the popular tool Stunnel.

As it happens, setting up Stunnel requires you to use OpenSSL for a number of tasks common to most of the other OpenSSL-dependent applications you're likely to encounter in your bastion-server activities. Therefore, even if you don't end up needing Stunnel yourself, I think you'll still find this chapter useful for figuring out how to generate server certificates, administer your own Certificate Authority, and so forth.

5.1. Stunnel and OpenSSL: Concepts

At its simplest, *tunneling* is wrapping data or packets of one protocol inside packets of a different protocol. When used in security contexts, the term usually specifies the practice of wrapping data or packets from an insecure protocol inside encrypted packets.[1] In this section, we'll see how *Stunnel*, an SSL-wrapper utility, can be used to wrap transactions from various applications with encrypted SSL tunnels.

[1] Even having said that, some network geeks may find this use of the word *tunneling* something of a stretch. An encrypted data stream is different from a network protocol, and some people insist that tunneling is about protocols, not cleartext versus ciphertext. I justify my usage based on the end result, which is that one type of transaction gets encapsulated into a different type.

Many network applications have the virtues of simplicity (with regard to their use of network resources) and usefulness but lack security features such as encryption and strong or even adequately protected authentication. Web services were previously in this category, until Netscape Communications invented the Secure Sockets Layer (SSL) in 1994.

SSL successfully grafted transparent but well-implemented encryption functionality onto the HTTP experience without adding significant complexity for end users. SSL also added the capability to authenticate clients and servers alike with X.509 digital certificates (though in the case of client authentication, this feature is underutilized). Since Netscape wanted SSL to become an Internet standard, they released enough of its details so that free SSL libraries could be created, and indeed they were: Eric A. Young's SSLeay was one of the most successful, and its direct descendant OpenSSL is still being maintained and developed today.

Note that the SSL protocol itself, while still widely used, is in fact obsolete; its successor is the Transport Layer Security protocol (TLS). Among other things, TLS allows you to initiate secure (authenticated and/or encrypted) communications over an existing application session, unlike with SSL, in which authentication and encryption must be initiated at the outset of each session. (This is why SSL-enabled services such as HTTPS traditionally use a different port than their cleartext counterpartse.g., TCP 443 for HTTPS and TCP 80 for HTTPwhile TLS-enabled applications can use the same port for all transactions regardless of whether encryption might be initiated.)

Besides its obvious relevance to web security, OpenSSL has led to the creation of Stunnel, one of the most versatile and useful security tools in the open source repertoire. Stunnel makes it possible to encrypt connections involving virtually any single-port TCP service via SSL, without any modifications to the service itself. By "single-port TCP service," I mean a service that listens for connections on a single TCP port without subsequently using additional ports for other functions.

HTTP, which listens and conducts all of its business on a single port (usually TCP 80), is such a service. *rsync*, Syslog-ng, MySQL, and, yes, even Telnet are, too: all of these can be run in encrypted Stunnel SSL wrappers.

FTP, which listens on TCP 21 for data connections but uses connections to additional random ports for data transfers, is *not* such a service. Anything that uses Remote Procedure Call (RPC) is also disqualified, because RPC uses the Portmapper service to assign random ports dynamically for RPC connections. NFS and NIS/NIS+ are common RPC services; accordingly, neither will work with Stunnel.

Sun's newer WebNFS service doesn't require the Portmapper: it can use a single TCP port (TCP 2049), making it a viable candidate for Stunnel use, though I've never done this myself. See the *nfsd(8)* and *exports(5)* manpages for more information on using WebNFS with Linux.

Microsoft's SMB (CIFS) file- and print-sharing protocol can function similarly when limited to TCP port 139, albeit to varying degrees depending on your client OS, and can thus be tunneled as well. See David Lechnyr's excellent *Samba Tutorial* at http://hr.uoregon.edu/davidrl/samba.html. Section 4 of this tutorial, "Tunneling SMB over SSH," explains how Samba behaves the same in either casealthough written with SSH in mind rather than Stunnel.

5.1.1. OpenSSL

Stunnel relies on OpenSSL for all its cryptographic functions. Therefore, to use Stunnel, you must first obtain and install OpenSSL on each host on which you intend to use Stunnel. The

Chapter 6. Securing Domain Name Services (DNS)

One of the most fundamental and necessary Internet services is the Domain Name Service (DNS). Without DNS, users and applications would need to call all Internet hosts by their Internet Protocol (IP) addresses rather than human-language names that are much easier to remember. Arguably, the Internet would have remained an academic and military curiosity rather than an integral part of mainstream society and culture without DNS. (Who besides a computer nerd would want to purchase things from 208.42.42.101 rather than from www.llbean.com?)

Yet in the SANS Institute's most recent version of their consensus document, "The Twenty Most Critical Internet Security Vulnerabilities" (Version 4.0 October 8, 2003, http://www.sans.org/top20.htm), the *number one* category of Unix vulnerabilities reported by survey participants was BIND weaknesses. The Berkeley Internet Name Domain (BIND) is the open source software package that powers the majority of Internet DNS servers. Again according to SANS, "an inordinate number" of BIND installations are vulnerable to well-known (and in many cases, old) exploits.

That there are so many hosts with vulnerabilities in an essential service is bad news indeed. The good news is that, armed with some simple concepts and techniques, you can greatly enhance BIND's security on your Linux (or other Unix) DNS server. Although I begin this chapter with some DNS background, my focus here will be security. So if you're an absolute DNS beginner, you may also wish to read the first chapter or two of Albitz and Liu's definitive book, *DNS and BIND* (O'Reilly).

If even after all this, you still mistrust or otherwise dislike BIND and wish to try an alternative, this chapter also covers djbdns, a highly regarded alternative to BIND. In addition to listing some of djbdns's pros and cons, we'll discuss rudimentary djbdns installation and security.

6.1. DNS Basics

Although I just said this chapter assumes familiarity with DNS, let's clarify some important DNS terminology and concepts with an example.

Suppose someone (*myhost.someisp.com* in Figure 6-1) is surfing the Web and wishes to view the site http://www.dogpeople.org. Suppose also that this person's machine is configured to use the nameserver *ns.someisp.com* for DNS lookups. Since the name "www.dogpeople.org" has no meaning to the routers through which the web query and its responses will pass, the user's web browser needs to learn the Internet Protocol (IP) address associated with http://www.dogpeople.org before attempting the web query.

Figure 6-1. A recursive DNS query

First, *myhost* asks *ns* whether it knows the IP address. Since *ns.someisp.com* isn't authoritative for *dogpeople.org* and hasn't recently communicated with any host that is, it begins a query on the user's behalf. In DNS parlance, making one or more queries in order to answer a previous query is called *recursion*.

ns.someisp.com begins its recursive query by asking a *root nameserver* for the IP address of a host that's authoritative for the generic Top Level Domain *.org*. (All Internet DNS servers use a static "hints" file to identify the 13 or so official root nameservers. This list is maintained at *ftp://ftp.rs.internic.net/domain* and is called *named.root*.) In our example, *ns* asks *E.ROOT-SERVERS.NET* (an actual root server whose IP address is currently 193.203.230.10), who replies that DNS for *.org* is handled by TLD1.ULTRADNS.NET, whose IP address is 204.74.112.1.

ns next asks TLD1.ULTRADNS.NET for the name and IP address of a name authority for the zone dogpeople.org. TLD1.ULTRADNS.NET replies that DNS for dogpeople.org is served by woofgange.dogpeople.org, whose IP address is 55.100.55.100.

ns then asks *woofgang* (using *woofgang's* IP address, 55.100.55.100) for the IP of *www.dogpeople.org*. *woofgang* returns the answer (55.100.55.244), which *ns* forwards back to *myhost.someisp.com*. Finally, *myhost* contacts 55.100.55.244 directly via HTTP and performs the web query.

This is the most common type of name lookup. It and other single-host type lookups are simply called *queries*; DNS queries are handled on UDP port 53.

Not all DNS transactions involve single-host lookups, however. Sometimes it is necessary to transfer entire name-domain (zone) databases: this is called a *zone transfer*, and it happens when you use the end-user command *host* with the `-l` flag and the command *dig* with query-type set to `axfr`. The output from such a request is a complete list of all DNS records for the requested zone.

host and *dig* are normally used for diagnostic purposes, however; zone transfers are meant to be used by nameservers that are authoritative for the same domain to stay in sync with each other (e.g., for "master to slave" updates). In fact, as we'll discuss shortly, a master server should refuse zone-transfer requests from any host that is not a known and allowed slave server. Zone transfers are handled on TCP port 53.

The last general DNS concept we'll touch on here is *caching*. Nameservers cache all local zone files (i.e., their *hints* file plus all zone information for which they are authoritative), plus the results of all recursive queries they've performed since their last startupthat is, almost all of them. Each *resource record* (RR) has its own (or inherits its zone file's default) time-to-live (TTL) setting. This value determines how long each RR can be cached before being refreshed.

This, of course, is only a fraction of what one needs to learn to fully understand and use BIND. But it's enough for the purposes of discussing BIND security.

6.2. DNS Security Principles

DNS security can be distilled into two maxims: always run the latest version of your chosen DNS software package, and never provide unnecessary information or services to strangers. Put another way, keep current and be stingy!

This translates into a number of specific techniques. The first is to limit or even disable recursion, since recursion is easily abused in DNS attacks such as cache poisoning. Limiting recursion is easy to do using configuration-file parameters; disabling recursion altogether may or may not be possible, depending on the nameserver's role.

If, for example, the server is an *external* DNS server whose sole purpose is to answer queries regarding its organization's public servers, there is no reason for it to perform lookups of nonlocal hostnames (which is the very definition of recursion). On the other hand, if a server provides DNS resolution to end users on a local area network (LAN), it definitely needs to recurse queries from local hosts but can probably be configured to refuse recursion requests, if not all requests, from nonlocal addresses.

Another way to limit DNS activity is to use *split DNS* services (Figure 6-2). Split DNS, an example of the *split services* concept I introduced in Chapter 2 in the section "Deciding What Should Reside on the DMZ," refers to the practice of maintaining both *public* and *private* databases of each local name domain (zone). The public-zone database contains as little as possible: it should have NS records for publicly accessible nameservers, MX records of external SMTP (email) gateways, A records (aliases) of public web servers, and entries pertinent to any other hosts that one wishes the outside world to know about.

Figure 6-2. Split DNS

The private-zone database may be a superset of the public one, or it may contain entirely different entries for certain categories or hosts.

The other aspect to DNS "stinginess" is the content of zone files themselves. Even public-zone databases often contain more information than they need to. Hosts may have needlessly descriptive names (e.g., you may be telling the wrong people which server does what), or too granular contact information may be given. Some organizations even list the names and versions of the hardware and software of individual systems! Such information is almost invariably more useful to prospective crackers than to its intended audience.

Maintaining current software and keeping abreast of known DNS exposures is at least as important as protecting actual DNS data. Furthermore, it's easier: the latest version of BIND can always be downloaded for free from ftp://ftp.isc.org, and djbdns from http://cr.yp.to. Information about general DNS security issues and specific BIND and djbdns vulnerabilities is disseminated via a number of mailing lists and newsgroups (some of which are listed at the end of this chapter).

There are actually third and fourth maxims for DNS security, but they're hardly unique to DNS: take the time to understand and use the security features of your software, and, similarly, know and use security services provided by your DNS-registration provider. Network Solutions and other top-level domain registrars all offer several change-request security options, including PGP. Make sure that your provider requires at least email verification of all change requests for your name domains!

6.3. Selecting a DNS Software Package

The most popular and venerable DNS software package is BIND. Originally a graduate-student project at UC Berkeley, BIND is now relied on by thousands of sites worldwide. The latest version of BIND, v9, was developed by Nominum Corporation under contract to the Internet Software Consortium (ISC), its official maintainers.

BIND has historically been and continues to be the reference implementation of the Internet Engineering Task Force's (IETF's) DNS standards. BIND Version 9, for example, provides the most complete implementation thus far of the IETF's new DNSSEC standards for DNS security. Due to BIND's importance and popularity, the better part of this chapter will be about securing BIND.

But BIND has its detractors. Like Sendmail, BIND has had a number of well-known security vulnerabilities over the years, some of which have resulted in considerable mayhem. Also like Sendmail, BIND has steadily grown in size and complexity: it is no longer as lean and mean as it once was, nor as stable. Thus, some assert that BIND is insecure and unreliable under load.

Daniel J. Bernstein is one such BIND detractor, but one who's actually done something about it: he's the creator of djbdns, a complete (depending on your viewpoint) DNS package. djbdns has some important features:

Modularity

Rather than using a single monolithic daemon like BIND's *named* to do everything, djbdns uses different processes to fill different roles. For example, djbdns not only uses different processes for resolving names and responding to queries from other resolvers; it goes so far as to require that those processes listen on different IP addresses. This modularity results in both better performance and better security.

Simplicity

djbdns's adherents claim it's easier to configure than BIND, although this is subjective. At least from a programming standpoint, though, djbdns's much smaller code base implies a much simpler design.

Security

djbdns was designed with security as a primary goal. Furthermore, its smaller code base and architectural simplicity make djbdns inherently more auditable than BIND: less code to parse means fewer overlooked bugs. To date, there have been no known security vulnerabilities in any production release of djbdns.

Performance

D. J. Bernstein claims that djbdns has much better speed and reliability, and a much smaller RAM footprint, than BIND. Several acquaintances of mine who administer extremely busy DNS servers rely on djbdns for this reason.

So, djbdns is superior to BIND in every way, and the vast majority of DNS administrators who use BIND are dupes, right? Maybe, but I doubt it. djbdns has compelling advantages, particularly its performance. If you need a caching-only nameserver but not an actual DNS authority for your domain, djbdns is clearly a leaner solution than BIND. But the IETF is moving DNS in two key directions that Mr. Bernstein apparently thinks are misguided, and therefore that he refuses to support in djbdns.

The first is DNSSEC. For secure zone transfers, djbdns must be used with rsync and OpenSSH, since djbdns does not support TSIGs or any other DNSSEC mechanism. The second is IPv6, which djbdns does not support in the manner recommended by the IETF (which is not to say that Mr. Bernstein is completely against IPv6; he objects to the way the IETF recommends it be used by DNS).

So, which software package do you choose? If performance is your primary concern, if you believe djbdns is inherently more secure than BIND (even BIND configured the way I'm about to describe), or if you want a smaller and more modular package than BIND, I think djbdns is a good choice.

6.4. Securing BIND

An installation of BIND in which you can feel confident requires quite a bit of work, regarding both how the daemon runs and how its configuration files deal with communication.

6.4.1. Making Sense out of BIND Versions

Three major versions of BIND are presently in use, despite the ISC's best efforts to retire at least one of them. BIND v9 is the newest version and its current minor-version number is, as of this writing, 9.2.3.

For a variety of practical and historical reasons, however, the BIND user community and most Unix vendors/packagers have been slow to embrace BIND v9, so BIND v8 is still in widespread use. Due to two nasty buffer-overflow vulnerabilities in BIND v8 that can lead to *root* compromise, it is essential that anyone using BIND v8 use its latest version, currently 8.4.4, or better still, upgrade to BIND v9, which shares no code with BIND v8 or earlier.

Speaking of earlier versions, although BIND v8.1 was released in May 1997, some users continue using BIND v4. In fact, a few Unix vendors and packagers still bundle BIND v4 with their operating systems. This is due mainly to stability problems and security issues with BIND v8 and mistrust of BIND v9. Accordingly, the Internet Software Consortium has continued, reluctantly, to issue occaisional security patches for Version 4, despite having ceased other development of that code version some years ago.

So, which version should you use? In my opinion, if you have a choice in the matter, version 9 is by far the most stable and secure version of BIND, and it has proven immune to most of the vulnerabilities discovered in BIND 4 and 8 since 9's debut. (That fact belies some critics' insinuations that BIND 9 still contains code from 4 and 8.) To date, there have been only two security problems in BIND v9, both of them Denial of Service opportunities (and both quickly patched); BIND 9 has had no remote-root vulnerabilities.

If for some reason you must choose between BIND v4 and BIND v8, you should use the latest version of BIND 8 (but I do not otherwise recommend BIND 8, due to its history of poor security). BIND v8's support for transaction signatures, its ability to be run chrooted, and its flags for running it as an unprivileged user and group (all of which we'll discuss shortly) far outweigh whatever stability benefits BIND 4 may seem to have over it. Because BIND 8 is still in widespread use, I'll cover both BIND 8 and BIND 9 examples in this chapter, but I repeat: if you can, *use BIND 9!*

6.4.2. Obtaining and Installing BIND

Should you use a precompiled binary distribution (e.g., RPM, tgz, etc.), or should you compile BIND from source? For most users, it's perfectly acceptable to use a binary distribution, provided it comes from a trusted source. Virtually all Unix variants include BIND with their "stock" installations; just be sure to verify that you've indeed got the latest version.

If you're not already familiar with your Linux distribution's "updates" web page, now's the time to visit it. BIND is one of the essential packages, which most distributions maintain current versions of at all times (i.e., without waiting for a major release of their entire distribution before repackaging).

The command to check the version number of your installed BIND package with Red Hat Package Manager is:

```
rpm -q -v package-name
```

if the package has already been installed, or:

```
rpm -q -v -p /path/to/package.rpm
```

if you have a package file but it hasn't been installed yet. The rpm package name for BIND is usually *bind9* or *bind*.

If you perform this query and learn that you have an old (pre-9.2.3 version), most package formats support an upgrade feature. Simply download a more current package from your Linux

6.5. djbdns

If after reading or skimming my BIND hints you're still suspicious of BIND's size, complexity, and history, you may wish to try djbdns, Daniel J. Bernstein's lightweight but robust alternative.

While this section makes particular note of djbdns's security features, the intent is to provide a general primer on djbdns use. This is (hopefully) justified for two reasons. First, the very act of choosing djbdns rather than BIND has positive security ramifications, if for no other reason than it "diversifies the DNS gene pool." Second, while widely used, djbdns hasn't yet received much treatment in the print media, so this primer is one of the first of its kind (if not *the* first).

If neither of these assumptions seems compelling to you, you needn't feel guilty for sticking with BIND (provided you run Version 9 and take the time to configure, secure, and maintain it carefully). For what it's worth, I'm a BIND v9 user myself.

6.5.1. What Is djbdns?

BIND can be considered the nuclear-powered kitchen sink, blender, and floor polisher of DNS software. It gurgles busily in the corner and occasionally springs a leak or explodes. Despite its market share, it's an old machine with spotty maintenance records.

djbdns, then, is the set of tools that you'd find at a DNS specialty store: simple, secure, fast, and safe when used as directed. Almost unnoticed, this package serves millions of domain names every day at large Internet domain-hosting companies and other busy sites, such as DirectNIC, NameZero, Interland, and TicketMaster. You may be surprised to learn that *tinydns* (the public nameserver component of djbdns) is the second most used nameserver on the Internet. A 2002 survey of 22 million *.com* domains (http://cr.yp.to/surveys/dns1.html) showed that 70% were served by BIND, and 8% by tinydns. A 2004 survey of almost 38 million domains (http://mydbs.bboy.net/survey/), which included *.com*, *.net*, *.org*, *.info*, and *.biz* domains, showed a 15.5% share for tinydns. On average, *tinydns* handled more domains per server (446) than BIND (72) or Microsoft DNS Server (21).The software is very reliable. It just keeps running without human intervention, other than to modify domain data. Memory use is limited, processes are monitored and restarted when needed, and logs are automatically rotated to avoid filling up the disk. I rarely have to worry about it, which says a lot.

Like BIND, djbdns is free software for Unix and Unix-like systems. djbdns can replace BIND or coexist as a primary or secondary nameserver.

djbdns comprises servers, clients, libraries, and helper services (see Table 6-2).

Table 6-2. djbdns's component and associated packages

djbdns package	Description
dnscache	Caching nameserver
tinydns	Authoritative nameserver
axfrdns	Zone-transfer server
axfr-get	Zone-transfer client
walldns	A reverse DNS wall: provides reverse look-ups without revealing internal network layouts
rbldns	IP-address list server, suited for blackhole lists
dnsip, dnsname, dnsmx, dnsipq	

6.6. Resources

Hopefully, we've given you a decent start on securing your BIND- or djbdns-based DNS server. You may also find the following resources helpful.

6.6.1. General DNS Security Resources

comp.protocols.tcp-ip.domains

USENET group

http://www.intac.com/~cdp/cptd-faq/

comp.protocols.tcp-ip.domains's Frequently Asked Questions about DNS

Rowland, Craig. "Securing DNS" (http://www.guides.sk/psionic/dns/)

Instructions on securing BIND on both OpenBSD and Red Hat Linux

6.6.1.1 Some DNS-related RFCs (available at http://www.rfc-editor.org)

- 1035 (general DNS specs)
- 1183 (additional Resource Record specifications)
- 2308 (Negative Caching)
- 2136 (Dynamic Updates)
- 1996 (DNS Notify)
- 2535 (DNS Security Extensions)

6.6.1.2 Some DNS/BIND security advisories (available at http://www.cert.org)

CA-2002-31

"Multiple Vulnerabilities in BIND" (Versions 4 and 8)

CA-2002-15

"Denial-of-Service Vulnerability in ISC BIND 9"

CA-2000-03

"Continuing Compromises of DNS Servers"

CA-99-14

"Multiple Vulnerabilities in BIND"

CA-98.05

"Multiple Vulnerabilities in BIND"

CA-97.22

"BIND" (cache poisoning)

6.6.2. BIND Resources

Internet Software Consortium. "BIND Operator's Guide" ("BOG")

Distributed separately from BIND 8 source code; current version downloadable from ftp://ftp.isc.org/isc/bind/src/8.3.3/bind-doc.tar.gz. The BOG is the most important and useful piece of official BIND 8 documentation.

Chapter 7. Using LDAP for Authentication

Suppose you've got an IMAP (mail) server and a bunch of users, but you don't want to give each user a shell account on the server: you'd rather use some sort of central user-authentication service that you can use for other things, too. While you're at it, you also need an online address book for your organization that could similarly be used both with email and with other groupware applications. And suppose that in addition to all that, you need to provide all your users with encryption tools that use X.509 certificates, and therefore need to manage digital certificates for your entire organization.

Would you believe that one service can address all three scenarios? LDAP, the Lightweight Directory Access Protocol, does all of this and more. And wouldn't you know it, the open source community is blessed with a free, stable, and fully functional LDAP package that is already part of most Linux distributions: OpenLDAP.

The only catch is that LDAP is a complicated beast. To make sense of it, you're going to have to add still more acronyms and some heavy-duty abstractions to your bag of Unix tricks. But armed with this chapter and a little determination, before you know it, you'll have the mighty LDAP burro pulling several very large plows simultaneously, thus making your network both more secure and easier to use. (Security and convenience seldom come hand in hand.)

This chapter is divided into three main sections: "LDAP Basics," a high-level introduction to the LDAP protocol; "Setting Up the Server," in which we'll install OpenLDAP software and get things started; and "LDAP Database Management," in which we'll create and populate an LDAP database.

7.1. LDAP Basics

In a nutshell, LDAP provides directory services: a centralized database of essential information about the people, groups, and other entities that compose an organization. Since every organization's structure and its precise definition of "essential information" may be different, a directory service must be highly flexible and customizable: it's therefore an inherently complex undertaking.

7.1.1. Directory-Services Protocols

X.500, CCIT's protocol for directory services, was designed to provide large-scale directory services for very large and complex organizations. Accordingly, X.500 is itself a large and complex protocol, so much so that a "lightweight" version of it was created: the Lightweight Directory Access Protocol (LDAP). LDAP, described in RFCs 1777 and 2251, is essentially a subset of the X.500 protocol, and it's been far more widely implemented than X.500 itself.

X.500 and LDAP are open protocols, like TCP/IP: neither is a standalone product. A protocol has to be implemented in some sort of software, such as a kernel module, a server daemon, or a client program. Also like TCP/IP, not all implementations of LDAP are alike, or even completely interoperable (without modification). The particular LDAP implementation we'll cover here is OpenLDAP, but you should be aware that other software products provide alternative implementations. These include Netscape Directory Server, Sun ONE Directory Server, and even, in a limited way, Microsoft Active Directory (in Windows 2000 Server).

Luckily, LDAP is designed to be extensible: creating an LDAP database that is compatible with different LDAP implementations is usually a simple matter of adjusting the database's record formats (or *schema*, which we'll discuss shortly). Therefore it's no problem to run an OpenLDAP server on a Linux system that can provide address-book functionality to users running Netscape Communicator or Microsoft Outlook.

7.1.2. Hierarchies and Naming Conventions

The whole point of a directory service is to provide a "roadmap" of your organization: an abstract data model that correlates closely to the "shape" and structure of that which it describes. For many organizations, it makes sense for their LDAP database to be structured like their organization chart. For others, it makes more sense for their LDAP database to correlate with the geographical locations of their organization's various offices and other buildings (especially if their org chart changes frequently). And for still others, a perfectly flat naming structure is most appropriate.

The most visible manifestation of an LDAP database's structure is in its naming convention, so much so that the terms *naming convention* and *database structure* are practically interchangeable when you're talking about LDAP. Thus, before I give some examples of LDAP database setups, let's discuss LDAP naming conventions.

You're probably already familiar with the concept of hierarchical naming conventions thanks to Internet Domain Name Service (DNS), in which each organization on the Internet belongs to some *top-level domain* such as .org, .com, .info, etc., but with its own unique *domain name* (e.g., example.com) and perhaps with *subdomains* (e.g., marketing.example.com and support.example.com). This scheme is extended to people via email addresses, each of which consists of a unique username within the organization, which is concatenated to the organization's domain name (e.g., salesweasel@marketing.example.com).

Conceptually, entity names in LDAP and X.500 are built the same way. The full name of an LDAP/X.500 entity, called its *distinguished name* (or *dn*), is similarly constructed from a unique combination of an entity name plus shared organization-name elements. For example, my own distinguished name in an LDAP database might be expressed as `cn=Mick Bauer,dc=wiremonkeys,dc=org`. (*cn* is short for *common name*, which is the name my entry is indexed by, and *dc* is short for *domain component*.)

Technically, my entity name (`cn=Mick Bauer`) need not be totally unique: if there are other people in the directory named Mick Bauer, there's no problem so long as each of us has a unique *dn*that is, so long as each one of our "full" LDAP names is unique. In actual practice, it's a lot easier to ensure unique *dn*s by enforcing unique entity names (*cn*s, *uid*s, etc.), as we'll see shortly.

7.2. Setting Up the Server

If you're like me, you're a lot less interested in LDAP theory than you are in LDAP practice, so let's go ahead and install OpenLDAPwe'll go further with LDAP database design in a minute. (And if you aren't like me, then good for you! But you'll still have to skip ahead a few pages if you want more LDAP theory right this instant.)

7.2.1. Getting and Installing OpenLDAP

Being such a useful and important thing, OpenLDAP is included in most major Linux distributions. Generally, it's split across multiple packages: server daemons in one package, client commands/programs in another, development libraries in still another, etc. You're building an LDAP server, so naturally you'll want to install your distribution's OpenLDAP server package, plus OpenLDAP runtime libraries if they aren't included in the server package.

You might be tempted to forego installing the OpenLDAP client commands on your server if there will be no local user accounts on it (i.e., if you expect all LDAP transactions to occur over the network, not locally). However, these client commands are useful for testing and troubleshooting, so I strongly recommend you install them.

The specific packages that make up OpenLDAP in Fedora and Red Hat are *openldap* (OpenLDAP libraries, configuration files, and documentation); *openldap-clients* (OpenLDAP client software/commands); *openldap-servers* (OpenLDAP server programs); and *openldap-devel* (headers and libraries for developers). Although these packages have a number of fairly mundane dependencies (e.g., *glibc*), there are two required packages in particular that you may not already have installed: *cyrus-sasl* and *cyrus-sasl-md5*, which help broker authentication transactions with OpenLDAP.

In SUSE, OpenLDAP is provided via the RPMs *openldap2-client*; *openldap2* (which includes both the OpenLDAP libraries and server daemons); and *openldap2-devel*. As with Red Hat, you'll need to be sure to also install the package *cyrus-sasl*, located in SUSE's *sec1* directory.

Note that earlier SUSE distributions (e.g., SUSE 8.0) provided packages for OpenLDAP Versions 1.2 and 2.0. If your version gives you the choice, be sure to install the newer 2.0 packages listed in the previous paragraph (e.g., *openldap2* rather than *openldap)*, unless you have a specific reason to run OpenLDAP 1.2.

For Debian 3.0 ("Woody"), the equivalent deb packages are *libldap2* (OpenLDAP libraries, in Debian's *libs* directory); *slapd* (the OpenLDAP server package, found in the *net* directory); and *ldap-utils* (OpenLDAP client commands, also found in the *net* directory). You'll also need *libsasl7*, from the Debian *libs* directory.

If your distribution of choice doesn't have binary packages for OpenLDAP, if there's a specific feature of the very latest version of OpenLDAP that is lacking in your distribution's OpenLDAP packages, or if you need to customize OpenLDAP at the binary level, you can always compile it yourself from source you've downloaded from the official OpenLDAP web site at http://www.openldap.org.

7.2.2. Configuring and Starting slapd

The main server daemon in OpenLDAP is called *slapd*, and configuring this program is the first step in getting OpenLDAP working once it's been installed. Its configuration is determined primarily by the file */etc/openldap/slapd.conf*.

The "OpenLDAP 2.0 Administrator's Guide" at http://www.openldap.org/doc/admin20/guide.html has an excellent "Quick-Start" procedure for getting *slapd* up and running: it's in Section 2, starting at Step 8. (That document also explains directory services and LDAP concepts in more depth than I do in this chapter.)

Let's step through this procedure to make sure you get off to a good start. The first thing to do is to edit *slapd.conf*, an example of which is shown in Example 7-1. As you can see, *slapd.conf* is a typical Linux configuration file: each line in it consists of a parameter name followed by a value.

Example 7-1. Customized part of /etc/openldap/slapd.conf

7.3. LDAP Database Management

Okay, we've installed OpenLDAP, configured *slapd*, gotten TLS encryption working, and created our first LDAP record. Now it's time to add some users and start using our servere.g., for authenticating IMAP sessions.

7.3.1. Database Structure

The first step in creating an LDAP user database is to decide on a directory structure i.e., whether to group users and other entities or whether to instead use a completely flat structure. If your LDAP database will be used strictly as an online address book or authentication server, a flat database may suffice; in that case, your users' Distinguished Names (DNs) will look like this: `dn=Mick Bauer,dc=wiremonkeys,dc=org`. We discussed some of the issues surrounding LDAP database structure earlier, in the section "Hierarchies and Naming Conventions."

As I mentioned then, LDAP is extremely flexible, and there are far more ways to structure an LDAP database than I can do justice to here. So to keep this discussion simple, I'm going to use a flat database for the rest of this chapter's examples; I leave it to you to determine whether and how to structure an LDAP database that best meets your particular LDAP needs. The documentation at http://www.openldap.org and included with OpenLDAP software provides ample examples.

7.3.1.1 Schema and user records

A related decision you'll need to make is which LDAP attributes to include for each record. I've described how these are grouped and interrelated in schemas; you may recall that the schemas you specify (include) in */etc/openldap/slapd.conf* determine which attributes will be available for you to use in records.

In addition to including schema in */etc/openldap/slapd.conf*, in each record you create you'll need to use `objectclass` statements to associate the appropriate schemas with each user. Again, the schema files in */etc/openldap/schema* determine which schema support which attributes, and within a given schema, which object classes those attributes apply to.

It may seem like a kluge to sort through and combine `objectclasses`, trying to cobble together the right combination of LDAP attributes to meet your particular needs: wouldn't it make more sense to somehow pull all your desired attributes into a single, custom `objectclass`? It would, and you can, by creating your own schema file. However, it turns out to be much less work, and much less of a "reinventing the wheel" exercise, to simply combine a few standard `objectclasses`.

Suppose you intend to use your LDAP server to authenticate one of the many protocols such as POP or IMAP, which request a username and a password. The essential LDAP attributes for this purpose are `uid` and `userPassword`..

One way to determine which schema and object classes provide `uid` and `userPassword` is to *grep* the contents of */etc/openldap/schema* for the strings `uid` and `userPassword`, note which files contain them, and then manually parse those files to find the object classes that contain those attributes in `MUST()` or `MAY( )` statements. If I do this for `uid` on Red Hat 7.3 system running OpenLDAP 2.0, I find that the files *core.schema*, *cosine.schema*, *inetorgperson.schema*, *nis.schema*, and *openldap.schema* contain references to the `uid` attribute.

Quick scans of these files (using *less*) tell me that:

- *core.schema*'s object `uidObject` requires `uid`
- *cosine.schema*'s only reference to the attribute `uid` is commented out and can be disregarded
- *inetorgperson.schema* contains an object class, `inetOrgPerson`, which supports `uid` as an optional attribute
- *nis.schema* contains two object classes, `posixAccount` and `shadowAccount`, both of

7.4. Conclusions

LDAP is one of the most complicated technologies I've worked with lately; to get it working the way you need to, you'll need to spend a lot of time testing, while watching logs and fine-tuning the configurations of both the LDAP server itself and the applications you wish to authenticate against it.

But having such a flexible, powerful, and widely supported authentication and directory mechanism is well worth the trouble. If it isn't already, this will become especially clear in Chapter 9, in which I'll show how to use LDAP to authenticate IMAPS email retrieval.

7.5. Resources

http://www.openldap.org

OpenLDAP software and documentation, including the important "OpenLDAP Administrator's Guide."

http://web500gw.sourceforge.net/errors.html

List of error codes used in LDAP error messages. This is essential in interpreting LDAP log messages.

http://www.ibiblio.org/oswg/oswg-nightly/oswg/en_US.ISO_8859-1/articles/exchange-replacement-howto/exchange-replacement-howto/

The Exchange Replacement HOWTO, which describes how to use LDAP as the authentication mechanism for Cyrus-IMAPD.

http://www.mandrakesecure.net/en/docs/ldap-auth.php

Vincent Danen's online article "Using OpenLDAP For Authentication," a somewhat Mandrake-centric but nonetheless useful introduction.

Carter, Gerald. *LDAP System Administration*. Sebastopol, CA: O'Reilly, 2003.

An excellent book with detailed coverage of OpenLDAP.

Team LiB ◂ PREVIOUS NEXT ▸

Chapter 8. Database Security

The "M" in LAMP, and the most popular open source database for Linux, is MySQL. It's easy to install and configure, runs light, and is quite fast. You'll commonly see it harnessed to Apacheserving up site content and authenticating users and offering a tempting target to those with more time than sense or conscience. In this chapter, we'll apply to database servers some of the methods we use to secure web servers, email servers, and nameservers. It's a little shorter than many of the other chapters because a database server is, from a security viewpoint, simpler than a web server or email server.

Working from the outside into the crunchy database center, we'll cover:

- The types of security problems. What should you worry about?
- Server placement. Where should you put your MySQL server to protect it from TCP exploits? How can you provide secure access for database clients?
- Database server installation. What version of MySQL should you use? What are the best file/directory ownerships and modes?
- Database configuration. How do you create database user accounts and grant permissions?
- Database operation. How do you protect against malicious SQL and bonehead queries? What are good practices for logging and backup?

For one reason or another, you might want to consider an alternative to MySQL. You can dip your toes in the commercial database waters (Oracle, DB2/UDB, Sybase) or stay in the open source pool. At the top of the open source list is PostgreSQL (http://www.postgresql.org/), which has more of the features of the big commercial relational databasesviews, triggers, referential integrity, subselects, stored procedures, and so on (although many of these features are coming to MySQL). Firebird (http://firebird.sourceforge.net/) is a spin-off of Borland's InterBase. Computer Associates has said it will release Ingres as open source (http://opensource.ca.com/projects/ingres/). SQLite (http://www.sqlite.org/) is an embeddable database that may become more well-known from its inclusion in recent releases of PHP.

You might also consider LDAP (Chapter 7). If your main use of a database is for user authentication and you don't need SQL, LDAP may be a faster and simpler solution.

Team LiB ◂ PREVIOUS NEXT ▸

Team LiB ◀ PREVIOUS NEXT ▶

8.1. Types of Security Problems

The problems a database server may encounter should sound familiar:

- **Server compromise**. Any software, especially code written in languages such as C or C++, has the potential for buffer overflows, format-string attacks, and other exploits that are by now all too familiar. And software written in any language has logic errors and plain old blunders.
- **Data theft**. Data can be extracted from the database even if everything seems to be configured well. It just takes one logical error or an overly permissive access control.
- **Data corruption or loss**. The person in the mirror may do as much damage inadvertently as the hooded and cloaked database vandal does by design.
- **Denial of Service**. MySQL is fast but does not always degrade gracefully under load. We'll see how far it bends before it breaks, and how to prevent the latter.

Team LiB ◀ PREVIOUS NEXT ▶

8.2. Server Location

Where should you place a database server? The main factors are:

- Who will access the database?
- How important is the data?

Exposing a database directly to the public might earn you a call from the Society for the Prevention of Cruelty to Databases. A *public database server* is normally an internal server, accessed only by other servers and clients behind the firewall. In this chapter, we'll look at examples of the most common database users: *web servers* and *database administrators*. We'll also show how to insert multiple layers of protection between the sensitive database server and the harsh weather of the public Internet.

The MySQL server listens for connections on a socketa Unix socket for connections on the same machine or a TCP socket for other machines. Its IANA-registered TCP port number is 3306, and I'll use this value in examples, but other port numbers can be used if needed.

How far from the Internet should the database be placed? Truly precious data (such as financial records) should be far back, on a dedicated database server within a second DMZ (internal to the DMZ that contains public-facing things such as web servers). The intervening firewall should pass traffic only between the database client (e.g., the web server) and database server on a specific TCP port. iptables should be configured on each machine so that the database client talks to that database port (3306) on the database server and the database server accepts a connection to port 3306 only from the host containing the web server.

For less precious data, the MySQL server may be on a dedicated machine in the outer DMZ, side by side with its clients. This is a common configuration for security, performance, and economic reasons. Configure iptables on the database server to accept connections on port 3306 only from the web server, and configure iptables on the web server to allow access to the database server on port 3306.

For local client access, MySQL can use a local Unix domain socket, avoiding TCP exploits. If a client accesses the host as *localhost*, MySQL automatically uses a Unix domain socket. By default, this socket is the special file */tmp/mysql.sock*.

8.2.1. Secure Remote Administration

Although we worry most about the security of the connection between the database server and its major clients, we also need to pay attention to the back door: administrative use.

Database administration includes creating and modifying databases and tables, changing permissions, loading and dumping data, creating reports, and monitoring performance. The main methods for administrative access are:

- VPN to the server
- *ssh* to the server
- Tunneling a local port to the server
- Using the Web

8.2.1.1 VPN to the server

If you have a VPN (virtual private network) connecting your local machine and the database server, you can access the server as though you were in the DMZ. Open source VPNs include FreeS/WAN (http://www.freeswan.org), Openswan (http://www.openswan.org/), OpenVPN (http://openvpn.sourceforge.net/), and strongSwan (http://www.strongswan.org/). All are under active development except FreeS/WAN.

Cisco and many other vendors sell commercial VPN products.

8.2.1.2 ssh to the server

8.3. Server Installation

Now that you've located your database server to protect against TCP exploits, you need to select a safe version of MySQL to guard against any code-based vulnerabilities.

8.3.1. Choosing a Version

Bug fixes, security fixes, performance enhancements, new features, and new bugs are part of each new server release. You always want the most recent stable version. At the time of writing, MySQL Server 4.1 is production, and 5.0 is the development tree. Old 3.x releases still abound, the most recent being 3.23.58. If you're running an older version of mySQL, make sure it's newer than 3.23.55 to avoid a remote MySQL *root* account (not Linux *root*) exploit. Make the move to 4.1 if you can, because there are many improvements. Here are some useful links to keep up with new problems as they're discovered:

Vulnerabilities

http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=mysql

Bugs

http://bugs.mysql.com/search.php

Change logs

http://dev.mysql.com/doc/mysql/en/News.html

8.3.2. Installing and Configuring the Server and Clients

MySQL comes standard with Red Hat and Fedora, as RPM packages *mysql-server* and *mysql* (clients and libraries). If you install from RPM, it creates the startup script */etc/init.d/mysqld* and the links to it from the runlevel directories (*/etc/rc[0-6].d*). If you want to install from source, see the latest details at http://dev.mysql.com/doc/mysql/en/Installing_source.html.

When the MySQL startup script is run by *root*, it should call another script called *safe_mysqld* (server Version 4.0 and newer) or *mysqld_safe* (pre-4.0), which is typically in */usr/bin*. This script then starts the MySQL server as user *mysql*. The database server should not run as the Unix *root* user. In fact, *mysqld* won't run as *root* unless you force it to with `--user=root`.

If you need to run MySQL as *root* for some reason, you can chroot the server to help contain a successful attack. To conserve space and avoid work here, I'll refer you to the article at http://www.securityfocus.com/infocus/1726.

8.3.3. Files

Table 8-1 shows where a Red Hat RPM installation puts things. As with any type of server, file location and ownership can affect security. A little later, I'll talk about these files and settings in the *my.cnf* configuration file(s).

Table 8-1. Common locations for MySQL files

File	Location (Red Hat 9)	Owner	Group	Mode
Server binary	*/usr/bin/mysql*	*root*	*root*	755
Global configuration file	*/etc/my.cnf*	*root*	*root*	644
Server-specific configuration file	*/var/lib/mysql/data/my.cnf*	*mysql*	*mysql*	644
Error logfile	*/var/log/mysqld.log*	*mysql*	*mysql*	644

8.4. Database Operation

Now that you've installed a reasonably secure version of the server in a reasonably secure location, let's look at how to run the thing securely.

8.4.1. MySQL Table Types

Many new developers of MySQL-backed web sites have been horrified to watch their database fall over and sink into the swamp just as their site becomes popular. Although MySQL has a reputation for speed, this is primarily in cases where database reads greatly outnumber writes. Once the number of simultaneous writes crosses some threshold, performance degrades most ungracefully.

This is a self-inflicted Denial of Service by the implementation of the default *MySQL table type* : MyISAM. It locks the whole table with each write (INSERT, UPDATE, or DELETE), pushing back all other requests. It's like closing all check-in lines but one at a busy airport terminal. Waits lengthen until the administrator must kill database threads or restart the database server.

MySQL actually has multiple table types, each implementing a different storage mechanism and behavior. You'll usually deal with two: MyISAM and InnoDB. MyISAM is great for reads and counts (such as COUNT * FROM TABLE), bad for heavy writes, and lacking true *transactions*the ability to perform multiple SQL statements as a unit and roll back to the original state if there are problems.

InnoDB is more recent, with full transaction support (ACID compliance, for the database folks), foreign-key constraints, and finer-grained locking. It's preferred when there are many writes or a need for transactions. People who are used to MyISAM should be aware that COUNT(*) is much slower in InnoDB tables. InnoDB is more complex and has many specialized options.

If you're just starting with MySQL, try MyISAM first and move up to InnoDB later if you need the write performance or transaction support. Luckily, you can do this with a single SQL command:

```
alter table table_name type=innodb
```

Many public MySQL-based sites such as *slashdot.org* have migrated from MyISAM to InnoDB.

8.4.2. Loading Datafiles

If you have FILE privileges, you can bulk load data from a flat file to a MySQL table. This has obvious security implications.

The SQL LOAD DATA command reads a flat file on the database machine into a MySQL table. This could be used to load */etc/passwd* into a table, then read it with a SQL SELECT statement. Since end users should not be stuffing files into tables, it's best to restrict this to administrative accounts. For example, if you need to load a flat file into a particular table every day, create a MySQL account for that purpose and grant it load privileges:

```
GRANT FILE ON database.table TO user @host identified by "password"
```

The SQL LOAD DATA LOCAL command allows the database server to read files from the client. This permits an evil server to grab any file from the database client, or an evil client to upload a file of its choice.

Recent versions of MySQL (3.23.49+ and 4.0.2+) are compiled to include an explicit `--enable-local-infile` option for backward compatibility. To disable this ability completely, they can be compiled without this option. Local loads can also be disabled at runtime by starting *mysqld* with the `--local-infile=0` option.

8.4.3. Writing Data to Files

The SQL command SELECT ... INTO OUTFILE dumps the results of the select operation into an external file. This is another good reason not to run the server as Unix *root*. The FILE

8.5. Resources

http://www.mysql.com

Home of MySQL.

http://dev.mysql.com/doc/mysql/en/Security.html

MySQL general security issues.

http://jeremy.zawodny.com/mysql/mytop/

mytop is *top* for MySQL, an indispensable display of database traffic. Helps you to see and kill runaway queries.

Chapter 9. Securing Internet Email

Like DNS, email's importance and ubiquity make it a prime target for vandals, thieves, and pranksters. Common types of email abuse include the following:

- Eavesdropping confidential data sent via email
- "Mail-bombing" people with bogus messages that fill up their mailboxes or crash their email servers
- Sending messages with forged sender addresses to impersonate someone else
- Propagating viruses
- Starting chain letters (hoaxes)
- Hijacking the email server itself to launch other types of attacks
- Sending unsolicited commercial email (UCE), a.k.a. "spam"

The scope and severity of these threats are not helped by the complexity of running Internet email services, including both Mail Transfer Agents (MTAs) and Mail Delivery Agents (MDAs). Email administration requires a working understanding of the Simple Mail Transfer Protocol (SMTP) plus your MDA protocol of choice (typically IMAP or POP3), as well as a mastery of your MTA and MDA applications of choice. There really aren't any shortcuts around either requirement (although some MTAs and MDAs are easier to master than others).

There are a number of MTAs in common use. Sendmail is the oldest and traditionally the most popular. Postfix is a more modular, simpler, and more secure alternative by Wietse Venema. Qmail is another modular and secure alternative by Daniel J. Bernstein. Exim is the default MTA in Debian GNU/Linux. And those are just a few!

In this chapter, we'll cover some general email security concepts, and then we'll explore specific techniques for securing two different MTAs: Sendmail, because of its popularity, and Postfix, because it's my preferred MTA. But we won't stop there!

As important as MTAs are, your users don't interact directly with them; most users retrieve mail via a Mail Delivery Agent (MDA) service such as POP3 or IMAP (or a web interface that interacts with an MDA). Therefore we'll also cover MDA security basics, how to secure the popular Cyrus IMAP MDA with both SSL and LDAP, and then end with a brief discussion of email encryption.

9.1. Background: MTA and SMTP Security

MTAs move email from one host or network to another. This task contrasts with that of Mail Delivery Agents (MDAs), which move mail within a system (i.e., from an MTA to a local user's mailbox, or from a mailbox to a file or directory). In other words, MTAs are like the mail trucks (and airplanes, trains, etc.) that move mail between post offices; MDAs are like the letter carriers who distribute the mail to their destination mailboxes. Procmail is one popular MDA on Linux systems.

In addition to MTAs and MDAs, there are various kinds of email readers, including POP3 and IMAP clients, for retrieving email from remote mailboxes. These clients are also known as Mail User Agents (MUAs), of which Mutt, MS-Outlook, Pine, and Evolution are popular examples. There is no real-world analogue of these, unless your letters are handed to you each day by a servant whose sole duty is to check your mailbox now and then. But we're not concerned with MUAs or MDAs, except to mention how they relate to MTAs.

Most MTAs support multiple mail-transfer protocols, either via embedded code or separate executables. Nearly all MTAs, for example, support at least UUCP and SMTP. Nevertheless, for the remainder of this chapter, I'll assume you're interested in using your MTA for SMTP transactions, since SMTP has been the dominant mail-transfer protocol of the Internet for some time.

9.1.1. Email Architecture: SMTP Gateways and DMZ Networks

No matter what other email protocols you support internally, such as the proprietary protocols in Microsoft Exchange or Lotus Notes, you need at least one SMTP host on your network if you want to exchange mail over the Internet. Such a host, which exchanges mail between the Internet and an internal network, is called an SMTP gateway. An SMTP gateway acts as a liaison between SMTP hosts on the outside and either SMTP or non-SMTP email servers on the inside.

This liaison functionality isn't as important as it once was: the current versions of MS Exchange, Lotus Notes, and many other email-server products that used to lack SMTP support can now communicate via SMTP directly. But there are still reasons to have all inbound (and even outbound) email arrive at a single point, chief among them security.

First, it's much easier to secure a single SMTP gateway from external threats than it is to secure multiple internal email servers. Second, "breaking off" Internet mail from internal mail lets you move Internet mail transactions off the internal network and into a DMZ network. Now your gateway can be isolated from both the Internet and the internal network by a firewall (see Chapter 2).

Therefore, I recommend, even to organizations with only one email server, the addition of an SMTP gateway, even if their server already has SMTP functionality.

But what if your firewall *is* your FTP server, email server, etc.? Although the use of firewalls for any service hosting is scowled upon by the truly paranoid, this is common practice for very small networks (e.g., home users with broadband Internet connections). In this particular paranoiac's opinion, DNS and SMTP can, if properly configured, offer less exposure for a firewall than services such as HTTP.

For starters, DNS and SMTP potentially involve only indirect contact between untrusted users and the server's filesystem. (I say "potentially" because it's certainly possible, with badly written or poorly configured software, to run extremely insecure DNS and SMTP services.) In addition, many DNS and SMTP servers (e.g., BIND and Postfix) have chroot options and run as unprivileged users. These two features reduce the risk of either service being used to gain *root* access to the rest of the system if they're compromised in some way.

9.1.2. SMTP Security

There are several categories of attacks on SMTP email. The scenario we tend to worry about most is exploitation of bugs in the SMTP server application itself, which may result in a disruption of service or even in the hostile takeover of the underlying operating system. Buffer-overflow attacks are a typical example, such as the one described in CERT® Advisory CA-1997-05 (*MIME Conversion Buffer Overflow in Sendmail Versions 8.8.3 and 8.8.4*; see

Team LiB PREVIOUS NEXT

9.2. Using SMTP Commands to Troubleshootand Test SMTP Servers

Before diving into specific software-configuration tips, here's a technique that can be used to troubleshoot or test any SMTP server: manual mail delivery. Normally, end users don't use SMTP commands because end users generally don't transfer their email manually. That's the job of MUAs, MDAs, and MTAs.

But it so happens that SMTP is a simple ASCII-based protocol built on TCP, and it's therefore possible to use SMTP commands to interact directly with an email server by *telnet*ing to TCP port 25 on that server. This is a useful technique for checking and troubleshooting MTA configurations. All you need is a *telnet* client and a working knowledge of a few of the commands in RFC 2821.

Here's a sample session:

```
$ telnet buford.hackenbush.com 25
Trying 10.16.17.123...
Connected to buford.hackenbush.com.
Escape character is '^]'.
220 buford.hackenbush.com ESMTP Postfix
helo woofgang.dogpeople.org
250 buford.hackenbush.org
mail from:<mick@dogpeople.org>
250 Ok
rcpt to:<groucho@hackenbush.com>
250 Ok
data
354 End data with <CR><LF>.<CR><LF>
Subject: Test email from Mick
Testing, testing, 1-2-3...
.
250 Ok: queued as F28B08603
quit
221 Bye
Connection closed by foreign host.
```

Let's dissect the example, one command at a time:

`helo woofgang.dogpeople.org`

The *HELO* command (SMTP commands are case insensitive) provides the remote server with your hostname or domain name. This is usually *not* verified by the server (e.g., via reverse-DNS).

`mail from:<mick@dogpeople.org>`

The *MAIL* command is used to specify your email's "from:" address. Again, this is usually taken at face value.

`rcpt to:<groucho@hackenbush.com>`

Use the *RCPT* command to specify your email's "to:" address. This address may or may not be validated: a well-configured SMTP host will reject nonlocal destination addresses for incoming mail to prevent unauthorized mail relaying.

`data`

DATA means "and now, here's the message." To specify an optional *Subject* line, make the first word of the first line of your message `Subject:`, which is followed immediately by your subject string. You can specify other SMTP headers, too, each on its own line; if you want, you can even make up your own headers (e.g., `X-Slartibartfast: Whee!`)

9.3. Securing Your MTA

Now we come to the specifics: how to configure SMTP server software securely. But which software should you use?

My own favorite MTA is Postfix. Wietse Venema, its creator, has outstanding credentials as an expert and pioneer in TCP/IP application security, making security one of the primary design goals. What's more, Postfix has a very low learning curve: simplicity is another design goal. Finally, Postfix is extremely fast and reliable. I've never had a bad experience with Postfix in any context (except the self-inflicted kind).

Qmail also has an enthusiastic user base. Even though it's only slightly less difficult to configure than Sendmail, it's worth considering for its excellent security and performance. D. J. Bernstein's official Qmail web site is at http://cr.yp.to/qmail.html.

Exim, another highly regarded mailer, is the default MTA in Debian GNU/Linux. The official Exim home page is http://www.exim.org, and its creator, Philip Hazel, has written a book on it, *Exim: The Mail Transfer Agent* (O'Reilly).

I mention Qmail and Exim because they each have their proponents, including some people I respect a great deal. But as I mentioned at the beginning of the chapter, Sendmail and Postfix are the MTAs we're going to cover in depth here. So if you're interested in Qmail or Exim, you'll need to refer to the URLs I just pointed out.

After you've decided *which* MTA to run, you need to consider *how* you'll run it. An SMTP gateway that handles all email entering an organization from the Internet and vice versa but doesn't actually host any user accounts will need to be configured differently from an SMTP server with local user accounts and local mailboxes.

The next two sections are selective tutorials on Sendmail and Postfix. I'll cover some basic aspects (but by no means all) of what you need to know to get started on each application, and then I'll cover as much as possible on how to secure it. Where applicable, we'll consider configuration differences between two of the most common roles for SMTP servers: gateways and what I'll call "shell servers" (SMTP servers with local user accounts).

Both Sendmail and Postfix are capable of serving in a wide variety of roles and therefore support many more features and options than I can cover in a book on security. Sources of additional information are listed at the end of this chapter.

9.4. Sendmail

Sendmail is one of the most venerable Internet software packages still in widespread use: it first appeared in 4.1c BSD Unix (April 1983), and to this day, it has remained the most relied-upon application of its kind. But Sendmail has both advantages and disadvantages.

9.4.1. Sendmail Pros and Cons

On the plus side, Sendmail has a huge user community; as a result, it's easy to find both free and commercial support for it, not to mention a wealth of electronic and print publications. It's also stable and predictable, one of the most mature network applications of all time.

On the downside, Sendmail has acquired a certain amount of "cruft" (layers of old code) over its long history, resulting in a reputation of it being insecure and bloated. Both charges are open to debate, however.

While it's true that Sendmail has had a number of significant vulnerabilities over the years, these have been brought to light and fixed very rapidly. An argument can therefore be made that Sendmail security is a glass half-empty/half-full situation. Depending on your viewpoint, Sendmail's various vulnerability reports and subsequent patches may prove that Sendmail is inherently insecure; or perhaps the fact that they come to light and are fixed quickly proves that Sendmail's development team and user community are pretty much on top of things, or maybe you think the truth is somewhere in between. (I'm in this last camp.)

A more useful criticism is that Sendmail is monolithic: a vulnerability in one portion of its functionality results in the compromise of the entire application. Since Sendmail must run as *root* when performing some of its duties, *any* Sendmail vulnerability has the potential to be used to gain *root* privileges.

As for the "bloatware" charge, it's true that Sendmail has a much larger code base than other MTAs such as Qmail and Postfix, as well as a larger RAM footprint. This probably has at least as much to do with Sendmail's monolithic architecture (one executable provides the great majority of Sendmail's functionality) as it does with cruft. However, Sendmail's code has been scrutinized so closely by so many programmers over the years that it's a little hard to believe that too much blatantly unnecessary or inefficient code has survived intact over the past 20 years.

Sendmail is also criticized for its complexity. The syntax of its configuration file, *sendmail.cf*, is nonintuitive, to say the least. In my opinion, its difficulty ranks somewhere between C and regular expressions. Like them, this is due to Sendmail's power. Regardless, this point is now largely moot: modern versions of Sendmail can be configured via *m4* macros, which provide a much less user-hostile experience than editing *sendmail.cf* directly.

A Disclaimer

I'm a Postfix fan myself. I run Postfix as my domain's public SMTP gateway (though I do use Sendmail on my private network for local mail delivery). Therefore, nothing in this section, including its very existence, should be construed to mean that I think Sendmail is the best choice for everyone's MTA needs. You'll need to decide for yourself whether Sendmail is the best tool for your environment.

However, I will say that I've spent a good deal of time over the past few years using and helping others to use Sendmail, and I think it's a lot better than many people give it credit for. In my experience, Sendmail is *not* the lumbering, slobbering, fragile beast some of its critics make it out to be.

In fact, I've found Sendmail to be stable and powerful, if a bit scary in its complexity. Furthermore, since the last CERT® advisory involving a remote-exploit vulnerability in Sendmail was in 1997 (number CA-1997-05), I'm simply not convinced that Sendmail is inherently unsecurable, as D. J. Bernstein and others insist. If it were, the CERT® advisories would continue to roll right out: Sendmail has been under *more* scrutiny in the past seven years than it was beforehand!

So while other MTAs (notably Postfix and Qmail) may have clear advantages over

9.5. Postfix

Wietse Venema's program, Postfix, provides an alternative to Sendmail that is simpler in design, more modular, and easier to configure and administer. Equally important, it's designed with scalability, reliability, and security as fundamental requirements.

This part of the chapter brings you up to speed quickly on how to use Postfix as a secure means of exchanging your network's email with Internet hosts. In particular, I'll focus on deploying Postfix on firewalls, in DMZs, and in other settings in which your SMTP server will have contact with untrusted systems.

I won't go into nearly as much depth with Postfix as I just did with Sendmail. The whole point of Postfix is ease of use: you'll have no problem figuring out how to use Postfix given little more than the documentation and example configurations included with Postfix itself.

9.5.1. Postfix Architecture

On the one hand, since Postfix can do most of what Sendmail can, its architecture is arguably as complex or even a little more so than Sendmail's. Postfix consists of a suite of daemons and helper applications, whereas Sendmail is essentially monolithic.

On the other hand, Postfix's modularity actually makes it much simpler in practice. For Mr. Venema and the others who maintain Postfix's code, it's easier to fix a bug in the SMTP daemon if that daemon's code is self-contained and not part of a much larger whole. As for end users, Postfix is administered mainly with the *postfix* command and a few others (most users only need *postqueue* and *postalias*).

Separating functions across different processes is a big factor in Postfix's speed and stability. Another factor is the intelligence with which Postfix handles mail. Rather than processing mail out of one big queue as Sendmail does, Postfix uses four different queues:

Maildrop queue

Mail that is submitted locally on the system is accepted in the maildrop queue. Here the mail is checked for proper formatting (and fixed if necessary) before being handed to the incoming queue.

Incoming queue

Mail initially received both from local processes via the maildrop queue and from external hosts via Postfix's *smtpd* process is preformatted if necessary and then sent to the incoming queue. Here it will stay until there's room in the active queue.

Active queue

Since the active queue contains messages that Postfix is actively trying to deliver, it has the greatest risk of something going wrong. Accordingly, the active queue is intentionally kept small, and it accepts messages only if there is space for them.

Deferred queue

Email that cannot be delivered is placed in the deferred queue. This prevents the system from continuously trying to deliver email and keeps the active queue as short as possible to give newer messages priority. This also enhances stability. If your MTA cannot reach a given domain, all the email for that domain is assigned a wait time and placed in the deferred queue so that those messages will not needlessly monopolize system resources.

When a deferred message's wait time has expired, the message is placed in the active queue again for delivery (as soon as there's room in the active queue). Each time delivery is attempted and failed, the message's wait time is increased, and it is returned to the deferred queue.

9.5.2. Getting and Installing Postfix

9.6. Mail Delivery Agents

As important as it is to run secure Mail Transfer Agent services, it's only part of your email picture, and it isn't even the part your end users will interact with directly. A Mail Delivery Agent (MDA) allows users to read (or download) email from their mailbox on a server. IMAP and POP3 are two popular MDA protocols used for Internet email; *webmail* interfaces, in fact, usually act as frontends to IMAP and POP3 servers. Our focus in the remainder of this chapter will be on the IMAP protocol, which is both newer and more powerful than POP3. (Much of what follows, however, should to some extent apply to POP3.)

An IMAP-based MDA system has two parts: an IMAP server, which houses user mailboxes and receives mail from some MTA; and a group of users running IMAP client software. The three most popular open source IMAP servers are University of Washington IMAP (UW IMAP), Cyrus IMAPD from Carnegie Mellon University, and Courier IMAP from Inter7 Internet Techologies. Popular IMAP client applications include Netscape/Mozilla Communicator, Microsoft Outlook, Mutt, Pine, and Apple Mac OS X Mail.

IMAP clients are out of the scope of our purposes here, but they're relatively easy to configure and use. Furthermore, most IMAP clients easily interoperate with most IMAP servers, so there isn't much to explain.

9.6.1. Principles of MDA Security

In practice, good MDA security requires two things: meaningful authentication, to keep strangers out, and encryption, to protect both the integrity of authentication transactions and the confidentiality of your users' email sessions. In addition, your MDA software needs to be configured in a way that takes full advantage of whatever other security features it supports, including running as a nonpriviliged user, running in a chroot jail, etc. (By now, I hope these principles are utterly familiar to you!)

MDA authentication is usually handled one of several ways:

- By authenticating users via the MDA server's underlying operating system, e.g., requiring each email user to have a user account on the MDA server.
- By authentiating users via a dedicated database of email user accounts.
- By using some sort of centralized authentication service such as LDAP (see Chapter 7).

MDA encryption can also be implemented a couple of different ways. Most modern MDA server applications, such as Cyrus IMAP, natively support encrypted email sessions via the SSL and TLS protocols (see Chapter 5). Alternatively, since MDA protocols such as POP3 and IMAP are *single TCP port* protocols, an encryption "wrapper" such as Stunnel (Chapter 5) may be used to transparently add encryption at the network level, if your MDA server software doesn't have its own encryption capabilities.

In the remainder of this part of the chapter, I'll show how to:

- Configure Cyrus IMAP to use LDAP to authenticate email users.
- Configure Cyrus IMAP to accept only SSL/TLS-encrypted email-retrieval sessions.
- Make the most of Cyrus IMAP's other security features.

While the mechanics of these three tasks are specific to Cyrus IMAP, the principles and goals behind them are the same whether you run Cyrus, Courier IMAP, or an entirely different MDA service.

Note that in these procedures and examples, I'll assume that you've already got a working LDAP server and already know how to generate X.509 certificates. For more information on LDAP and digital certificates, see Chapter 5 and Chapter 7.

9.6.2. Which IMAP Server?

The first choice an email administrator must make in building an IMAP system is which server to use. What are the major differences between UW IMAP, Courier IMAP, and Cyrus IMAP? In

9.7. A Brief Introduction to Email Encryption

Encrypting your email from end to end is the very best defense against eavesdropping attacks; encrypting it and signing it is also a powerful defense against identity theft. However, because this book is about bastion-server security, and since email encryption is in most respects much more of a client/local application than a "back-office" application, I'm not going to go very far in depth on this topic. (The extent to which it *does* involve backend services, e.g., in Public Key Infrastructures, is outside the scope of this book.)

There are two predominant email encryption technologies in use nowadays, PGP and S/MIME. Both are end-to-end solutions (end users do all the encrypting and decrypting, with servers involved only in key distribution) And both are based on open standards. However, neither PGP nor S/MIME has achieved much popularity with less technical or nontechnical users. The ugly reality is that email encryption as we know it places a much higher burden of skill and knowledge on end users than, say, SSL does with web encryption.

That's because most SSL sessions on the Internet are, in real terms, "anonymously" encrypted. If I buy something from an online retailer, I may or may not care whether the retailer's secure web server presents me with an SSL certificate with a valid signature; the retailer absolutely does *not* care about whether my web browser even *has* a certificate. My browser and the server will happily build an encrypted session between each other without being terribly certain that the other party is who they say they are.[3] So in most real-world SSL transactions, there's no authentication.

[3] Yes, the server always presents the client with a certificate, but unfortunately, most users don't hesitate to accept any certificate presented by an authentic-looking web sitethat's what I mean by "anonymous, in real terms." Also, I may have a "customer account" with the retailer and be asked to type in a username and password before I can, e.g., view my account information. But the underlying encryption mechanism itself, SSL, has even more powerful authentication features that, for a variety of reasons, are seldom implemented. That's what I mean by "anonymous encryption." See Chapter 5 for more information about client-certificate authentication.

And that's fine, in most of those cases. But email encryption is another matter altogether: if you encrypt something for "your friend's eyes only," you care very much whether the key you're using to encrypt the message truly is your friend's: you don't want anybody else to be able to read the message. Your friend probably cares equally strongly whether it was actually you who sent the message and not some imposter.

Thus, email encryption isn't just about encryption; it's about *identity management*. (In fact, I'll go so far as to say that the encryption itself is the easy part.) Modern email encryption systems have yet to present users with simple and intuitive mechanisms for keeping track of the encryption credentials (keys) of everyone they need to communicate with, managing their own credentials, etc. It's an inherently complex and still somewhat immature technology.

Still, this stuff *does work*, and it's worth the effort it takes to deploy and use it.

PGP, short for "Pretty Good Privacy," is the older and more popular of the two technologies. The other, S/MIME, is rapidly gaining ground, thanks at least in part to the fact that support for it is built into Microsoft Exchange and Outlook.

9.7.1. PGP and GnuPG

The brainchild of hacker saint Phil Zimmerman, PGP was the first email encryption tool to gain anything resembling widespread popularity, and to this day, it is used all over the world. PGP exists in both free and commercial versions, but over its long history it has been, at various times, illegal for export from the U.S.; free for noncommercial use only; closed source; and in limbo (neither being sold as a commercial product or available for use in a free version).

Happily, PGP is now back to being actively maintained both as a commercial product and in a free-for-noncommercial-use version (see http://www.pgp.com/products/freeware.html for more information about PGP Freeware). However, for all of the reasons I just listed, even the ones that no longer apply, many people have switched from PGP to a 100% free and open source alternative: the GNU Privacy Guard, a.k.a. GnuPG (http://gnupg.org).

GnuPG is completely compliant with the OpenPGP protocol that PGP uses, but unlike PGP, GnuPG has always been a purely noncommercial project. It also intentionally lacks support for the patented IDEA algorithm, which makes GnuPG less "encumbered" (legally speaking) than

9.8. Resources

The following sources of information address not only security but also many other important aspects of SMTP and MTA configuration.

9.8.1. SMTP Information

RFC 2821, "Simple Mail Transfer Protocol." (*ftp://ftp.isi.edu/in-notes/rfc2821.txt*)

Useful for making sense of mail logs, SMTP headers, etc.

Shapiro, Gregory Neil. "Very brief introduction to create a CA and a CERT." (http://www.sendmail.org/~ca/email/other/cagreg.html)

A bare-bones procedure for generating a Certificate Authority certificate, generating server/client certificates, and using the CA certificate to sign server and client certificates. Handy for people who want to use X.509 mechanisms such as *STARTTLS* without becoming X.509 gurus.

9.8.2. Sendmail Information

Costales, Bryan, with Eric Allman. *sendmail*, Sebastopol, CA: O'Reilly, 1997.

The definitive guide to Sendmail. Chapters 19 and 34 are of particular interest, as they concern use of the *m4* macros. Most of the rest of this weighty tome covers the ugly insides of *sendmail.cf*.

Fennelly, Carole. "Setting up Sendmail on a Firewall, Part III." Unix Insider 06/01/1999 (http://www.itworld.com/Net/3314/swol-0699-security/)

Excellent article on running Sendmail 8.9 and later in a chroot environment.

Allman, Eric and Greg Shapiro. "Securing Sendmail." (http://www.sendmail.net/000705securitygeneral.shtml)

Describes many built-in security features in Sendmail and offers security tips applicable to most Sendmail installations.

Durham, Mark. "Securing Sendmail on Four Types of Systems." (http://www.sendmail.net/000710securitytaxonomy.shtml)

Durham, Mark. "Using SMTP AUTH in Sendmail 8.10." (http://www.sendmail.net/usingsmtpauth.shtml)

"Using New AntiSpam Features in Sendmail 8.10." (http://www.sendmail.net/810usingantispam.shtml)

"SMTP STARTTLS in sendmail/Secure Switch." (http://www.sendmail.org/~ca/email/starttls.html)

http://mail-abuse.com/services/mds-rbl.html

Home of the Realtime Blackhole List, which is a list of known sources of UCE.

9.8.3. Postfix Information

http://www.postfix.org

The definitive source for Postfix and its documentation.

http://msgs.securepoint.com/postfix/

Archive site for the Postfix mailing list.

Chapter 10. Securing Web Servers

You've hardened your server from the bottom up, with an external firewall protecting your DMZ, a local firewall blocking ports, and all the latest patches applied to your operating system. Your fortress is impregnable. But then you blast a hole straight through all these walls to a port on your server. Then you let anyone in the world wander in and run programs on your server, *using their own input*. You've lost touch with realityand/or you're a web administrator.

The Web continues to grow, and security problems follow. As firewalls and security tools improve, attacks move up the food chain, particularly toward web applications. In this chapter, I assume that you are hosting web servers and are responsible for their security. Although the examples discuss servers exposed to the Internet, most of the discussion applies to intranets and extranets as well. The platform is still *LAMP*: Linux, Apache, MySQL, PHP (and Perl). I'll talk about *A*, *M*, and *P* here. MySQL database server security is covered in Chapter 8, but database access from Perl and PHP is discussed here. We'll see how to protect your whole web environmentserver, content, applicationsand keep the weasels out of your web house.

10.1. Web Security

Bad things happen to good servers. Malice or mistake, local or remote, can foil the security goals mentioned in the first chapter. Table 10-1 lists some security problems you may encounter, as well as the desired security goals.

Table 10-1. Web-security problems and goals

Problems	Goals
Theft of service Warez or pornography uploads Pirate servers and applications Password sniffing Rootkit and Trojan program installation Distributed Denial of Service participation	System integrity
Vandalism, data tampering, or site defacement Inadvertent file deletion or modification	Data integrity
Theft of personal information Leakage of personal data into URLs and logs	Data confidentiality
Unauthorized use of resources Denial of Service Crash/freeze from resource exhaustion (e.g., memory, disk, process space, file descriptors, or database connections)	System and network availability

10.1.1. What, When, and Where to Secure

First secure your network and the operating system on your server, or all else will be for naught. Then work your way through the topics covered in this chapter:

Web server
Build time: obtaining and installing Apache
Setup time: configuring Apache
Web content
Static
Dynamic: SSI
Dynamic: CGI
Web applications
Authentication
Authorization
Sessions
Database access
Site management
Web services
Layers of defense

10.1.2. Some Principles

Before we begin, let's draw a deep breath and meditate on the basic security mantras that underlie what we do in this chapter:

10.2. The Web Server

A secure web service starts with a secure web server, which in turn starts with good codeno buffer overflows or other problems that could be exploited to gain *root* privileges. Apache has had a handful of critical vulnerabilities over the past few years, and has generally released fixed versions promptly. Apache powers about two-thirds of the 55 million hosts in the monthly Netcraft survey (http://news.netcraft.com/archives/web_server_survey.html).

Microsoft's Internet Information Server (IIS), with less than a third of Apache's market share, has had many critical and ongoing security problems. A Microsoft Security Bulletin issued in April 2002 described 10 critical problems in IIS 4 and 5. These include vulnerabilities to buffer overruns, Denial of Service, and cross-site scripting; a number of these provide full-system privileges to the attacker. IIS 6 is reportedly better.

In practice, most Apache security problems are caused by configuration errors, and I'll talk about how to avoid these shortly. Still, there are always bug fixes, new features, and performance enhancements, along with the occasional security fix, so it's best to start from the most recent stable release.

Although Apache 2.0 was released a few years ago, security and bug fixes continue for the 1.3 branch. Apache 2.0 has some interesting additions, such as *filters* (pipelined input modules) and *MPMs* (multiprocessing modules). The default MPM, *prefork*, works like 1.3 by starting a bunch of processes and assigning requests among them. The *worker* MPM handles requests in threads. But 2.0 uptake has been slow. One reason is that the threaded MPM requires all linked Apache modules *and all of their supporting libraries* to be threadsafe. Although Apache 2 and PHP (Version 4 and up) are threadsafe, some of the libraries used by PHP extensions may not be. This can cause errors that are extremely difficult to track. For this reason, Rasmus Lerdorf and the other PHP developers recommend using Apache 1.3 with PHP, or Apache 2 with the prefork MPM. Another method is to use FastCGI (http://www.fastcgi.com/), which runs as a separate process from Apache.

I still use Apache 1.3 with PHP. Since most users are still working with 1.3, that's what will be used in the examples in this chapter, with some 2.0 notes where needed. The book *Apache Security* (O'Reilly) has more details on security for 2.0.

10.2.1. Build Time: Installing Apache

Attacks are so frequent on today's Internet that you don't want to leave a window for attack, even for the few minutes it takes to set up a secure server. This section covers setting up your environment and obtaining the right version of Apache.

10.2.1.1 Setting up your firewall

A public web server is commonly located with email and nameservers in a DMZ, between outer and inner firewalls. You want to configure access for two classes of visitor:

- The public, visiting your site from the Internet
- Web administrators, who may be coming from the outside, inside, or another server in the DMZ

Web servers normally listen on TCP ports 80 (*http*:) and 443 (secure HTTP, *https*:). While you're installing Apache and the pieces are lying all around, block external access to these ports at your firewall (with iptables or other open source or commercial tools). If you're installing remotely, open only port 22 and use *ssh*. After you've configured Apache, tightened your CGI scripts (as described in this chapter), and tested the server locally, you can then reopen ports 80 and 443 to the world.

How you handle administrators depends on where they are and how they want to get to the web server. If administrators use command-line tools such as those described in this chapter, *ssh* is sufficient. If they use some web GUI, permissions and passwords need to be set for the corresponding scripts. Administrators might also tunnel to some port with *ssh* or *stunnel*, or use use other tools over a VPN.

10.2.1.2 Checking your Apache version

10.3. Web Content

After you've thoroughly configured Apache's configuration, you can finally deal with web content.

10.3.1. Static Content

Static content includes HTML, JavaScript, Flash, images, and other files that are served directly by the web server without interpretation. The files and their directories need to be readable by the user ID running Apache (*apache*, in our examples).

Static files don't pose much of a security threat on the server side. The web server just reads them and sends them to the requesting browser. Although there are many security issues with web browsers, client security is outside the scope of this chapter. Watch your browser vendor's web site for security news, patches, and new versions.

10.3.2. Dynamic Content: Server-Side Includes (SSI)

A step up from purely static pages, *server-side includes* allow inclusion of other static content, special dynamic content such as file-modification times, and even the output from the execution of external programs. Unlike CGI scripts, there is no way to pass input arguments to an SSI page.

10.3.2.1 SSI configuration

Apache needs to be told that an SSI file is not a lump of inert HTML, but should be parsed for SSI directives. First, check that includes are permitted for at least some files in this directory. Add this to *httpd.conf* or *access.conf*:

```
<Location /ssi_dir>
Options IncludesNoExec
</Location>
```

One way to differentiate HTML from SSI files is to use a special suffix such as *.shtml* and associate it with Apache's built-in MIME type for parsable content:

```
AddType application/x-server-parsed .shtml
```

or just assign the Apache handler directly:

```
AddHandler server-parsed .shtml
```

Using this tells the world that your pages use server-side includes. If you'd like to conceal this fact, use another suffix. One trick I've seen is to use *.html* for static text and *.htm* for SSI text:

```
AddHandler server-parsed .htm
```

A little-known feature of Apache is its ability to use the execute bit of a file to indicate that it should be parsed. I've used this to mix static and parsed HTML files in the same directory with the same suffix. The directive is as follows:

```
<Location /ssi_dir>
Options +IncludesNoExec
XBitHack full
</Location>
```

The extra attribute `full` tells Apache to check the modification time of the included file rather than the including file. To change an HTML file into an SSI file, make it executable:

```
chmod +x changeling.html
```

A visitor to the web site can't tell if the file is plain HTML or SSI.

10.3.2.2 Including files

10.4. Web Applications

The Web Application Security Consortium has classified web threats and tried to standardize their descriptions (http://www.webappsec.org/threat.html). The Open Web Application Secuity Project (OWASP) describes the top 10 vulnerabilities (http://www.owasp.org/documentation/topten.html) and how to secure web applications (http://www.owasp.org/documentation/guide/guide_about.html). All are well worth reading.

10.4.1. Processing Forms

The top risk in the OWASP list is currently *unvalidated input*. This is most evident in the workhorse of web applications, form processing.

In the previous section, I showed how to get and echo the value of the form element named *string*. I'll now show how to circumvent this simple code, and how to protect against the circumvention.

Client-side form checking with JavaScript is a convenience for the user, and it avoids a round-trip to the server to load a new page with error messages. However, it does not protect you from a handcrafted form submission with bad data. Here's a simple form that lets the web user enter a text string:

```
<form name="user_form" method="post" action="/cgi-bin/echo">
<input type="text" name="string">
<input type="submit" value="submit">
</form>
```

When submitted, we want to echo the string. Let's look again at a naive stab at `echo` in PHP:

```
<? echo "string = ", $_REQUEST["string"], "\n"; ?>
```

And the same in Perl:

```
#!/usr/bin/perl -w
use strict;
use CGI qw(:standard);
print header;
print "string = ", param("string"), "\n";
```

This looks just ducky. In fact, if you type `quack` into the *string* field, you see the output:

```
string = quack
```

But someone with an evil mind might enter this text into the *string* field:

```
<script language=javascript>history.go(-1);</script>
```

Submit this, and watch the JavaScript code bounce you right back to your input form. If this form did something more serious than echo its input (such as entering the contents of a literal tag into a database), the results could be more serious.

> Never trust user input. Validate everything on the server. Check for commands within data.

This is an example of someone uploading code to your server without your knowledge and then getting it to download and execute on any browser. This *cross-site scripting bug* was fixed within JavaScript itself some time ago, but that doesn't help in this case, because JavaScript is being injected into the data of a server-side script. HTML tags that invoke active content are shown in Table 10-7.

Table 10-7. HTML active content tags

10.5. Layers of Defense

Test your setup with a vulnerability scanner. The best open source tool is *nessus* (http://www.nessus.org), which includes tests for buffer overflows, bad Apache configurations, buggy CGI scripts, and many other problems. It includes tests from *nikto* (http://www.cirt.net/code/nikto.shtml) and *libwhisker* (http://www.wiretrip.net/rfp/p/doc.asp/i2/d21.htm), which can also be run on their own.

When you're ready for production, use multiple levels of protection:

- Firewall (Chapter 2)
- Intrusion detection and logging, such as S*nort*/*ACID* (Chapter 13)
- Log monitoring (Chapter 12)

10.6. Resources

Ristic, Ivan. Apache Security. O'Reilly, 2005.

Web Application Security Consortium: Threat Classification

http://www.webappsec.org/threat.html

The Ten Most Critical Web Application Security Vulnerabilities

http://www.owasp.org/documentation/topten.html

A Guide to Building Secure Web Applications

http://www.owasp.org/documentation/guide/guide_about.html

The World Wide Web Security FAQ

http://www.w3.org/Security/faq/www-security-faq.html

An oldie and goodie.

Improving Web Application Security: Threats and Countermeasures

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp

Big document on web threats and Microsoft solutions.

Chapter 11. Securing File Services

File transfers are among the most important Internet transactions. All Internet applications support file transfer in one form or another. In email, MIME attachments can take virtually any form, including executables and archives. HTTP supports file transfers with aplomb: "loading a web page" actually entails the downloading and displaying of a multitude of text, graphic, and even executable code files by your browser. Even Internet Relay Chat can be used to transfer files between chatters.

When all is said and done, however, email, HTTP, and IRC are all designed to handle relatively small chunks of data. This chapter covers tools and protocols specifically designed for transferring large files and large quantities of files.

The File Transfer Protocol (FTP) in particular is one of the oldest and (still) most useful methods for TCP/IP file transfers. Accordingly, this chapter covers both general FTP security and specific techniques for securing the ProFTPD FTP server. But FTP isn't the best tool for every bulk-data-transfer job, so we'll also cover *scp* and *rsync*. These, unlike FTP, can be encrypted with the help of Secure Shell or Stunnel, covered in Chapters Chapter 4 and Chapter 5, respectively. (Chapter 4 also covers SFTP, an FTP-like frontend for the Secure Shell.)

11.1. FTP Security

What would we do without FTP? You can use FTP to install Linux, download software from public archives, and share files with friends and colleagues. It's both venerable and ubiquitous. Most major sites on the Internet offer some level of public FTP access.

But like many other Internet applications, FTP is showing its age. Designed for a simpler era, FTP is gradually going the way of Telnet: it's still useful for "anonymous" (public) access, but its cleartext login makes it too dangerous for use with important user accounts.

Anonymous FTP, though, will probably remain with us for some time, so let's discuss FTP security, both in general and with specific regard to my preferred FTP servers, ProFTPD and vsftpd.

11.1.1. Principles of FTP Security

With FTP, we have several major threat models. The first concerns anonymous access: anonymous users shouldn't be able to do anything but list and download public files and maybe upload files to a single "incoming" directory. Needless to say, we don't want them to "escalate" their privileges to those of a more trusted user.

Another important FTP threat model involves local user accounts. If a local user logs in via FTP to upload or download something to or from his home directory, we don't want that session hijacked or eavesdropped on by anybody else, or the user's credentials may be stolen and used with other services such as *telnet*, SSH, etc.

The third threat model worth considering involves confidentiality. At the very least, login credentials must be protected from disclosure, as should any other sensitive data that is transmitted.

Unfortunately, by its very design FTP fails miserably in addressing any but the first of these threat models: a good FTP server package that is carefully configured can protect against privilege escalation, but like *telnet*, the FTP protocol as described in RFC 959 (ftp://ftp.isi.edu/in-notes/rfc959.txt) is designed to transmit both authentication credentials and session data in cleartext.

Accordingly, FTP is the wrong tool for almost anything but the anonymous exchange of public files. Using real user accounts for FTP exposes those users' credentials to eavesdropping attacks; all subsequent session data is similarly exposed. For this reason, most people's FTP security efforts tend to focus on properly configuring anonymous FTP services and on keeping their FTP server software up to date. Protecting FTP transactions themselves is all but futile.

If your users need to move data onto or off of the system, require them to use *scp*, *sftp*, or *rsync* in combination with *stunnel*. I describe all of these later in the chapter.

11.1.1.1 Active mode versus passive mode FTP

To make matters worse, FTP's use of TCP ports is, to put it charitably, inopportune. You may have already learned that FTP servers listen on TCP port 21. However, when an FTP client connects to an FTP server on TCP port 21, only part of the transaction uses this initial "control" connection.

FTP Server Packages Compared

For some time, WU-FTPD has been the most popular FTP server for Unix and Unix-like platforms. This is probably because, compared to the traditional BSD *ftpd* from which it evolved, WU-FTPD is very rich in features, very stable, and theoretically, more securable. I say "theoretically" with a bit of irony because in recent years, WU-FTPD itself has been vulnerable to a series of buffer overflows that, since WU-FTPD runs as *root*, have led to many servers being compromised. While its developers have been quick to provide patches, I personally avoid WU-FTPD since these bugs crop up with more regularity than I'm comfortable with.

ProFTPD, a "written-from-scratch" package with Apache-like configuration syntax and modularity, claims security as one of its fundamental design goals. Despite

11.2. Other File-Sharing Methods

Despite the amount of ink I've devoted here to FTP, I've also said repeatedly that despite its ubiquity, FTP is one of the least secure and least securable file-transfer techniques. The remainder of this chapter therefore concerns file-transfer mechanisms more appropriate for the exchange of nonpublic data between authenticated hosts and users.

11.2.1. SFTP and scp

The first FTP alternative I'll cover here is the most FTP-like: Secure FTP (SFTP), part of the Secure Shell (SSH) suite of tools. SSH was designed as a secure replacement for the "r" commands (*rlogin*, *rsh*, and *rcp*), which, like FTP, transmit all session data in cleartext, including authentication credentials. In contrast, SSH transparently encrypts all its transactions from start to finish, including authentication credentials: local logon credentials are never exposed to network eavesdroppers. SSH offers a remarkable combination of security and flexibility and is the primary topic of Chapter 4.

SSH has always supported *scp*, its encryption-enabled replacement for the *rcp* command, so it may seem redundant for SSH to also support *sftp*. But usability and familiarity notwithstanding, *sftp* provides a key feature lacking in *scp*: interactivity. By being interactive, *sftp* allows the client to browse files both on the remote host and locally (via the FTP commands *dir* and *ldir*, respectively) prior to downloading or uploading anything.

To use *scp*, however, you need prior knowledge of the remote system's filesystem layout and contents. While in many situations this isn't a big deal, particularly when using *scp* in scripts, it's an annoying limitation in many others. Thus, *sftp* deserves a place in the toolkits of SSH beginners and experts alike.

Note, however, that SSH doesn't explicitly support anonymous/public file sharing via either *sftp* or *scp*. It's certainly possible, given hefty amounts of caution and testing, to set up a nonprivileged account with an empty password and a closely watched home directory for this purpose. (*sshd* has a configuration option called `PermitEmptyPasswords` that is disabled by default but may be set to `yes`.) I consider this to be playing with fire, however: SSH was designed for and excels at providing secure, *restricted* access. Anonymous file services are not only the best use of conventional FTP daemons such as vsftpd; such access is best provided by them.

Configuration and use of the OpenSSH version of the Secure Shell, including *scp* and *sftp*, is covered in depth in Chapter 4.

11.2.2. rsync

Andrew Tridgell's *rsync* is another useful file-transfer tool, one that has no encryption support of its own but is easily "wrapped" (tunneled) by encryption tools such as SSH and Stunnel. What differentiates *rsync* (which, like *scp*, is based on *rcp*) is that it has the ability to perform *differential* downloads and uploads of files.

What About NFS and Samba?

NFS and Samba provide two ways to mount volumes on remote systems as though they were local. This is extremely useful, particularly if you use "thin clients" with limited local storage space or if you want to relieve users of backing up their personal data. NFS, developed and touted mainly by Sun Microsystems, is widely used in both Sun and Linux environments; in fact, the Linux version interoperates very well with the Sun version. Similarly, Samba is a Linux port of the Microsoft (actually IBM) SMB protocol and its related file- and printer-sharing functions, allowing Linux systems to act as clients and even servers to Windows hosts.

As nifty as both NFS and Samba are, however, I'm not covering them in any depth here, for the simple fact that neither is very secure, especially for Internet use. Both rely heavily on UDP, a connectionless and therefore easily spoofed protocol, and both have authentication mechanisms that have been successfully attacked in various ways over the years, in some cases trivially.

11.3. Resources

Bernstein, D. J. "PASV Security and PORT Security."

Online article at http://cr.yp.to/ftp/security.html (17 April 2004).

http://cr.yp.to/publicfile.html. (17 April 2004)

The home of publicfile, D. J. Bernstein's secure FTP/HTTP server. Like djbdns, it uses Bernstein's daemontools and ucspi-tcp packages.

Carnegie Mellon University (CERT Coordination Center). "Anonymous FTP Abuses." (http://www.cert.org/tech_tips/anonymous_ftp_abuses.html) 17 April 2004.

Carnegie Mellon University (CERT Coordination Center). "Anonymous FTP Configuration Guidelines." (http://www.cert.org/tech_tips/anonymous_ftp_config.html) 17 April 2004.

Carnegie Mellon University (CERT Coordination Center). "Problems with the FTP PORT Command or Why You Don't Want Just Any PORT in a Storm." (http://www.cert.org/tech_tips/ftp_port_attacks.html) 17 April 2004.

Garfinkel, Simson and Gene Spafford. *Practical Unix and Internet Security*. Sebastopol, CA: O'Reilly, 1996.

Klaus, Christopher. "How to Set up a Secure Anonymous FTP Site."

Online article; no longer maintained (Last update: 28 April 1994), but available at http://www.eecs.umich.edu/~don/sun/SettingUpSecureFTP.faq.

http://www.proftpd.org.

The official ProFTPD home page.

http://vsftpd.beasts.org.

The official vsftpd home page.

http://rsync.samba.org.

The official rsync home page.

Team LiB ◀ PREVIOUS NEXT ▶

Chapter 12. System Log Management and Monitoring

Whatever else you do to secure a Linux system, it must have comprehensive, accurate, and carefully watched logs. Logs serve several purposes. First, they help to troubleshoot all kinds of system and application problems. Second, they provide valuable early warning signs of system abuse. Third, after all else fails (whether that means a system crash or a system compromise), logs can provide us with crucial forensic data.

This chapter is about making sure your system processes and critical applications log the events and states you're interested in and dealing with this data once it's been logged. The two logging tools we'll cover are syslog and the more powerful Syslog-ng ("syslog new generation"). In the monitoring arena, we'll discuss Swatch (the Simple Watcher), a powerful Perl script that monitors logs in real time and takes action on specified events, plus a few "offline" log-reporting tools.

Team LiB ◀ PREVIOUS NEXT ▶

12.1. syslog

syslog is the tried-and-true workhorse of Unix logging utilities. It accepts log data from the kernel (by way of *klogd*), from any and all local process, and even from processes on remote systems. It's flexible as well, allowing you to determine what gets logged and where it gets logged to.

A preconfigured syslog installation is part of the base operating system in virtually all variants of Unix and Linux. However, relatively few system administrators customize it to log the things that are important for their environment and disregard the things that aren't. Since, as few would dispute, information overload is one of the major challenges of system administration, this is unfortunate. Therefore, we begin this chapter with a comprehensive discussion of how to customize and use syslog.

What About klogd?

One daemon you probably won't need to reconfigure but should still be aware of is *klogd*, Linux's kernel log daemon. This daemon is started automatically at boot time by the same script that starts the general system logger (probably */etc/init.d/syslogd* or */etc/init.d/sysklogd*, depending on which Linux distribution you use).

By default, *klogd* directs log messages from the kernel to the system logger, which is why most people don't need to worry about *klogd*: you can control the handling of kernel messages by editing the configuration file for *syslogd*.

This is also true if you use Syslog-ng instead of syslog, but since Syslog-ng accepts messages from a much wider variety of sources, including */proc/kmsg* (which is where *klogd* receives its messages), some Syslog-ng users prefer to disable *klogd*. Don't do so yourself unless you first configure Syslog-ng to use */proc/kmsg* as a source.

klogd can be invoked as a standalone logger; that is, it can send kernel messages directly to consoles or a logfile. In addition, if it isn't already running as a daemon, *klogd* can be used to dump the contents of the kernel log buffers (i.e., the most recent kernel messages) to a file or to the screen. These applications of *klogd* are especially useful to kernel developers.

For most of us, it's enough to know that for normal system operations, *klogd* can be safely left alone (that is, left with default settings and startup options*not* disabled). Just remember that when you use syslog in Linux, all kernel messages are handled by *klogd* first.

12.1.1. Configuring syslog

Whenever *syslogd*, the syslog daemon, receives a log message, it acts based on the message's type (or "facility") and its priority. syslog's mapping of actions to facilities and priorities is specified in */etc/syslog.conf*. Each line in this file specifies one or more facility/priority selectors followed by an action; a selector consists of a facility or facilities and a (single) priority.

In the following *syslog.conf* line in Example 12-1, `mail.notice` is the selector and `/var/log/mail` is the action (i.e., "write messages to */var/log/mail*").

Example 12-1. Sample syslog.conf line

```
mail.notice                /var/log/mail
```

Within the selector, `mail` is the facility (message category) and `notice` is the level of priority.

12.1.1.1 Facilities

Facilities are simply categories. Supported facilities in Linux are *auth*, *auth-priv*, *cron*, *daemon*

12.2. Syslog-ng

As useful and ubiquitous as syslog is, it's beginning to show its age. Modern Unix and Unix-like systems are considerably more complex than they were when syslog was invented, and they have outgrown both syslog's limited facilities and its primitive network-forwarding functionality.

Syslog-ng ("syslog new generation") is an attempt to increase syslog's flexibility by adding better message filtering, better forwarding, and eventually (though not quite yet), message integrity and encryption. In addition, Syslog-ng supports remote logging over both the TCP and UDP protocols. Syslog-ng is the brainchild of and is primarily developed and maintained by Balazs ("Bazsi") Scheidler.

Although its' much newer than syslogd, Syslog-ng is both stable and mature and has already been incorporated into major Linux distributions, including SUSE and Debian. A couple of its advanced security features are still works in progress, but Syslog-ng can be used in conjunction with TCP "tunneling" tools such as *stunnel* and *ssh* to authenticate or encrypt log messages sent to remote hosts.

12.2.1. Installing Syslog-ng from Binary Packages

As I just mentioned, Syslog-ng is already a standard package in the Debian and SUSE distributions as a drop-in replacement for syslogd. Debian's deb package is called *syslog-ng*, as is SUSE's RPM package. If you run Red Hat or Fedora, a simple Google search for "syslog-ng rpm" will turn up at least a couple of different sources of Syslog-ng RPMs for your distribution.

One of these will probably be Seth Vidal's page at http://www.dulug.duke.edu/~skvidal/RPMS/ . The subdirectories *fc1/* and *fc2/* contain binary RPMs for Fedora. You'll need both the *syslog-ng* and *libol* packages.

Of these three distributions (Debian, SUSE, and Fedora), only in Debian does Syslog-ng seamlessly replace *syslogd*. For SUSE and Fedora, you'll have a little bit of setup to do before you can go much further.

12.2.1.1 Replacing syslogd with Syslog-ng on SUSE

Once you've installed the RPM *syslog-ng*, you need to follow these steps (as *root*, naturally):

1. Enter the command `SuSEconfig --module syslog-ng`.
2. Stop *syslogd* with the command `rcsyslog stop`.
3. Open */etc/sysconfig/syslog* with the text editor of your choice, and change the value of the `SYSLOG_DAEMON` variable to `syslog-ng`.
4. Start Syslog-ng with the command `rcsyslog start`.
5. As you can see, both *syslogd* and Syslog-ng are started by the same init script. Therefore, do *not* make the change to */etc/sysconfig/syslog* (in Step three) before stopping the syslog service, otherwise you may end up with both *syslogd* and Syslog-ng running, with unpredictable results.

12.2.1.2 Replacing syslogd with Syslog-ng on Fedora (Vidal's RPMs)

Unlike with SUSE, in Fedora *syslogd* and Syslog-ng (as packaged by Seth Vidal) each have their own startup script. When you install the *libol* and *syslog-ng* RPMs, the post-installation script will automatically start Syslog-ng and enable its startup script, but will leave *syslogd* both running and enabled.

Follow these steps to gracefully replace *syslogd* with Syslog-ng:

1. Stop *syslogd* with the command `/etc/init.d/syslog stop`.
2. Restart Syslog-ng with the command `/etc/init.d/syslog-ng restart`.

12.3. Testing System Logging with logger

Before we leave the topic of system-logger configuration and use, we should cover a tool you can use to test your new configurations, regardless of whether you use syslog or Syslog-ng: *logger*. *logger* is a command-line application that sends messages to the system logger. In addition to being a good diagnostic tool, *logger* is especially useful for adding logging functionality to shell scripts.

The usage we're interested in here, of course, is diagnostics. It's easiest to explain how to use *logger* with an example.

Suppose you've just reconfigured syslog to send all daemon messages with priority *warn* to */var/log/warnings*. To test the new *syslog.conf* file, you'd first restart *syslogd* and *klogd* and then you'd enter a command like the one in Example 12-22.

Example 12-22. Sending a test message with logger

```
mylinuxbox:~# logger -p daemon.warn "This is only a test."
```

As you can see, *logger*'s syntax is simple. The `-p` parameter allows you to specify a *facility.priority* selector. Everything after this selector (and any other parameters or flags) is taken to be the message.

Because I'm a fast typist, I often use *while...do...done* statements in interactive *bash* sessions to run impromptu scripts (actually, just complex command lines). Example 12-23s sequence of commands works interactively or as a script.

Example 12-23. Generating test messages from a bash prompt

```
mylinuxbox:~# for i in {debug,info,notice,warning,err,crit,alert,emerg}
> do
> logger -p daemon.$i "Test daemon message, level $i"
> done
```

This sends tests messages to the daemon facility for each of all eight priorities.

Example 12-24, presented in the form of an actual script, generates messages for *all* facilities at each priority level.

Example 12-24. Generating even more test messages with a bash script

```
#!/bin/bash
for i in {auth,auth-priv,cron,daemon,kern,lpr,mail,mark,news,syslog,user,
uucp,
local0, local1,local2,local3,local4,local5,local6,local7}
  # (this is all one line!)

do
for k in {debug,info,notice,warning,err,crit,alert,emerg}
do
logger -p $i.$k "Test daemon message, facility $i priority $k"
done
done
```

Logger works with both syslog and Syslog-ng.

12.4. Managing System Logfiles with logrotate

Configuring and fine-tuning your system-logging facilities is extremely important for system security and general diagnostics. But if your logs grow too large and fill up their filesystem, all that work will be counterproductive.

Just What Do We Mean By "Rotate?"

All log-management mechanisms involve periodically moving/renaming a logfile to an archive copy and creating a new (empty) logfile. Rotation is necessary when multiple archive copies are maintained.

In the most common log-rotation scheme, a set of static filenames is maintained. For example, *messages, messages.1, messages.2, messages.3* is a typical three-archive filename set*messages* being the current logfile and *messages.3* being the oldest archive.

In this scheme, rotation is achieved by copying the second-to-oldest file over the oldest file (e.g., `mv messages.2 messages.3`). The third-oldest file's name is then changed to that of the second-oldest file's, and so forth, until the current file is renamed and a new (empty) "current" logfile is created (e.g., `mv messages messages.1; touch messages`). This is how *logrotate* behaves when its *rotate* parameter is set to a nonzero value.

As with syslog itself, most Linux distributions come with a preconfigured log-rotation scheme; on most of these distributions, this scheme is built on the utility *logrotate*. As with syslog, while this default scheme tends to work adequately for many users, it's too important a mechanism to take for granted. It behooves you to understand, periodically evaluate, and if necessary, customize your log-management setup.

12.4.1. Running logrotate

Red Hat, Fedora, SUSE, and Debian use *logrotate* to handle system-log growth. Global options and low-level (system) logfiles are addressed in */etc/logrotate.conf*, and application-specific configuration scripts are kept in */etc/logrotate.d/*.

When *logrotate* is run, all scripts in */etc/logrotate.d* are included into *logrotate.conf* and parsed as one big script. This makes *logrotate*'s configuration very modular: when you install an RPM or DEB package (of software that creates logs), your package manager automatically installs a script in */etc/logrotate.d*, which will be removed later if you uninstall the package.

Actually, the `include` directive in *logrotate.conf* may be used to specify additional or different directories and files to include. In no event, however, should you remove the statement that includes */etc/logrotate.d* if you use Red Hat or Debian, both of whose package managers depend on this directory for package-specific log-rotation scripts.

12.4.1.1 Syntax of logrotate.conf and its included scripts

There are really only two types of elements in *logrotate.conf* and its included scripts: directives (i.e., options) and logfile specifications. A *directive* is simply a parameter or a variable declaration; a *logfile specification* is a group of directives that apply to a specific logfile or group of logfiles.

In Example 12-25, we see a simple */etc/logrotate.conf* file.

Example 12-25. Simple logrotate.conf file

```
# Very simple logrotate.conf file

# Global options: rotate logs monthly, saving four old copies and sending
```

12.5. Using Swatch for Automated Log Monitoring

Okay, you've painstakingly configured, tested, and fine-tuned your system logger to sort system messages by type and importance and then log them both to their respective files and to a central log server. You've also configured a log-rotation scheme that keeps as much old log data around as you think you'll need.

But who's got the time to actually *read* all those log messages?

Swatch (the "Simple WATCHer") does. Swatch, a free log-monitoring utility written 100% in Perl, monitors logs as they're being written and takes action when it finds something you've told it to look out for. Swatch does for logs what Tripwire does for system-file integrity.

12.5.1. Installing Swatch

There are two ways to install Swatch. First, of course, is via whatever binary package of Swatch your Linux distribution of choice provides. (I use the term loosely here; "executable package" is more precise.) The current version of Mandrake has an RPM package of *swatch*, as does Debian, but none of the other most popular distributions (i.e., Red Hat, Fedora, and SUSE) do, though you can download Gavin Henry's Swatch RPMs for Fedora and Red Hat at http://fedoranews.org/ghenry/swatch/.

This is just as well, though, since the second way to install Swatch is quite interesting. Swatch's source distribution, available from http://swatch.sourceforge.net, includes a script called *Makefile.PL* that automatically checks for all necessary Perl modules (see "Should We Let Perl Download and Install Its Own Modules?" later in this chapter). If it finds them, it then generates a *Makefile* that can be used to build Swatch.

The required Perl modules are *Time::HiRes*, *File::Tail*, *Date::Calc*, and *Date::Format*. In earlier versions of Swatch, *Makefile.PL* would automatically download and install these from CPAN as needed. Nowadays, however, most distributions have their own binary packages for them, so if *Makefile.PL* complains that one or more of them isn't present, you should check your distribution's installation media or web site before going to CPAN (see sidebar).

After you've installed the required modules, either automatically from Swatch's *Makefile.PL* script or manually (and then running `perl Makefile.PL`), *Makefile.PL* should return the contents of Example 12-26.

Example 12-26. Successful Makefile.PL run

```
[root@barrelofun swatch-3.0.1]# perl Makefile.PL


Checking if your kit is complete...
Looks good
Writing Makefile for swatch
[root@barrelofun swatch-3.0.1]#
```

Once *Makefile.PL* has successfully created a *Makefile* for Swatch, you can execute the following commands to build and install it:

```
    make
    make test
    make install
    make realclean
```

The `make test` command is optional but useful: it ensures that Swatch can properly use the Perl modules we just went to the trouble of installing. If these tests fail, check out the "Help" forum at the Swatch site; when I built Swatch 3.1.1 on my SUSE 9.0 system, it initially failed, but thanks to the Help forum, I realized I was simply missing the *File::Tail* Perl module.

12.5.2. Swatch Configuration in Brief

Since the whole point of Swatch is to simplify our lives, configuring Swatch itself is, well, simple. Swatch is controlled by a single file, *$HOME/.swatchrc* by default. This file contains

12.6. Some Simple Log-Reporting Tools

Before we leave the topic of logging and log reporting, I should say just a few words about a less glamorous category of log tools: *offline* or *non-real-time* log reporters. The idea behind these is that periodically reviewing automatically-excerpted parts of your logfiles, while not as good as monitoring things in real time, is better than nothing.

Log reporters run as cron jobs. At the appointed time, the reporter searches the designated logfiles for particular words or strings (specified in a configuration file or word list), gleans some simple system statistics by running commands such as *df* and *free*, and emails a handy report to *root* (or some other designated user).

Over the years, I've found these sorts of utilities to be a nice sanity check against other mechanisms. However, be forewarned: you won't learn about anything important in such a log report *until well after the fact*! Therefore I recommend using log reporters *in addition to*, not instead of, real-time log-checkers such as Syslog-ng `match( )` rules and Swatch.

SUSE's log reporting package is called *logdigest*; Debian's is called *logcheck*; Red Hat and Fedora use *logwatch*. See these tools' respective manpages for configuration and usage information.

12.7. Resources

http://www.balabit.com

Official home of Syslog-ng.

Campin, Nate. "Central Loghost Mini-HOWTO." http://www.campin.net/newlogcheck.html)

Nate's site is an all-around excellent source of Syslog-ng information.

http://swatch.sourceforge.net

Swatch home page. (Has links to the latest version, online manpages, etc.)

http://www.cert.dfn.de/eng/logsurf/

Logsurfer home page. (An alternative to Swatch, provided by CERT-DFN.)

Friedl, Jeffrey E. F. *Mastering Regular Expressions*. Sebastopol, CA: O'Reilly, 1998.

http://defconX.wiremonkeys.org

The slideshow from my Defcon X talk "Stealthful Sniffing, Logging, and Intrusion Detection: Useful and Fun Things You Can Do Without An IP Address."

Team LiB ◂ PREVIOUS NEXT ▸

Chapter 13. Simple Intrusion Detection Techniques

Last night someone came into my house and replaced everything with an exact duplicate.

Steven Wright

Comprehensive logging, preferably with automated monitoring and notification, can help keep you abreast of system security status (besides being invaluable in picking up the pieces after a crash or a security incident). But as a security tool, logging only goes so far: it's no more sophisticated than the operating-system processes and applications that write those log messages. Events not anticipated by those processes and applications may be logged with a generic message or, worse still, not at all. And what if the processes, applications, or their respective logs are tampered with?

That's where Intrusion Detection Systems (IDS) come in. A simple *host-based IDS* can alert you to unexpected changes in important system files based on stored checksums. A *network IDS* (NIDS) can alert you to a potential attack in progress, based on a database of known attack signatures or even on differences between your network's current state and what the IDS considers its normal state. Some of these attacks (especially those at the application level, such as web exploits) might breeze through your firewalls. Multiple layers of defense are better than one. In the 2004 *CSI/FBI Computer Crime and Security Survey* (http://www.gocsi.com/), 98% of the organizations surveyed used a firewall, and 68% used an IDS.

Between simple host-based IDSes and advanced statistical NIDSes, there is a lot of information I can't do justice to in one chapter: I highly recommend Northcutt's and Amoroso's books (listed in the "Resources" section at the end of this chapter) if you're interested in learning about this topic in depth. But as it happens, you can achieve a high degree of intrusion detection potential without a lot of effort, using free, well-documented tools such as Tripwire Open Source and Snort.

This chapter describes some basic intrusion detection concepts and how to put them to work without doing a lot of work yourself.

Team LiB ◂ PREVIOUS NEXT ▸

13.1. Principles of Intrusion Detection Systems

In practical terms, there are two main categories of IDS: host-based and network- based. A host-based IDS, obviously enough, resides on and protects a single host. In contrast, a network-based IDS resides on one or more hosts (any of which may be a dedicated "network probe") and protects all the hosts connected to its network.

13.1.1. Host-Based IDSes: Integrity Checkers

Dedicated host-based IDSes tend overwhelmingly to rely on integrity checking. In theory, host-based IDSes should use a much broader category of tools. Commercial IDS products, such as ISS RealSecure and Marcus Ranum's Network Flight Recorder, both of which I categorize as Network IDSes, can use sophisticated methods (such as traffic analysis) on a single host, if desired.

Integrity checking involves the creation and maintenance of a protected database of checksums, cryptographic hashes, and other attributes of a host's critical system files (and anything else you don't expect to change on that system). The integrity checker periodically checks those files against the database: if a file has changed, an error or alert is logged. Ideally this database should be stored on a read-only volume, or off the system altogether, to prevent its being tampered with.

The assumption here is that unexpected changes may be the result of some sort of attack. For example, after "rooting" a system, a system cracker will often replace common system utilities such as *ls*, *ps*, and *netstat* with "rootkit" versions, which appear to work normally but conveniently neglect to list files, processes, and network connections (respectively) that might betray the cracker's presence. (See http://www.chkrootkit.org/ for a script that can be used to detect installed rootkits and for links to many other related sites and articles.)

By regularly checking system utilities and other important files against the integrity checker's database, we can minimize the chances of our system being compromised without our ever knowing it. The less time between a system's compromise and its administrators' learning that it's been compromised, the greater the chance its administrators can catch or at least evict the intruders before too much damage is done.

Integrity checking has a beautiful simplicity: we don't necessarily care *how* a monitored file has been changed; we mainly care that it *has*. To be effective, an integrity checker doesn't need to be smart enough to know that */bin/ls* no longer shows files belonging to the user *evild00d*; it only needs to know that */bin/ls* has been altered since the last legitimate system update. Having said that, a good integrity checker *will* also tell us which external characteristics of */bin/ls* have changed: its size, modification date, physical location (inode), etc.

> Any integrity checker with an untrustworthy database is worthless. It's imperative to create this database as soon as possible after installing the host's operating system from trusted media. I repeat: installing, configuring, and maintaining an integrity checker is not worth the effort unless its database is initialized on a clean system.

Also keep in mind with integrity checkers is that they are *not proactive*. (Unless one or more of your perimeter systems is a honeypota "sacrificial lamb" that sets off alerts when compromised so you can prevent other systems from being compromised, too. However, I wouldn't count on attackers obliging you by attacking the honeypot system first!) In most cases, by the time your integrity checker registers an alert, you only have a small chance of intervening before a serious compromise occurs. Furthermore, the attacker may tamper with or altogether suppress the alert before it reaches you.

This does *not* mean that integrity checking is futile! On the contrary, the first step in incident response is learning that something has occurred in the first place, and if you install an integrity checker properly, you *do* have a better chance of learning about attacks soon enough to take meaningful action. If the worst happens, data from your integrity checker can be invaluable in figuring out what happened and in rebuilding your system if need be.

13.2. Using Tripwire

Among the most celebrated and useful things to come out of Purdue's COAST project (http://www.cerias.purdue.edu/coast/) was the Unix integrity checker Tripwire, created by Dr. Eugene Spafford and Gene Kim. Tripwire was originally both open source and free, but in 1997, Tripwire went commercial, and fee-free use was restricted to academic and other noncommercial settings.

Happily, a couple of years ago, Tripwire, Inc. released "Tripwire Open Source, Linux Edition." Until Tripwire Open Source was released, the older Academic Source Release (ASR) lacked features long available in commercial versions of Tripwire. The current release of Tripwire Open Source is based on Version 2.2 of the commercial product, which is now up to Version 4.5. Although it still lacks a few "enterprise" features such as centralized management of multiple systems (Tripwire, Inc. understandably still wishes to differentiate its commercial product line), it is functionally very similar to the commercial Tripwire for Servers.

Note that Tripwire Open Source is free for use only on noncommercial Unices (i.e., Linux and Free/Net/OpenBSD). In fact, it's officially supported only on Red Hat Linux and FreeBSD, although there's no obvious reason why it shouldn't compile and run equally well on other Linux and BSD distributions. (I run it not only on Red Hat but also on SUSE and Debian Linux, with no problems to report). For commercial Unices such as Sun Solaris and HP-UX, commercial Tripwire is still the only legal option in commercial settings.

13.2.1. Obtaining, Compiling, and Installing Tripwire

A format-string vulnerability affects versions of Tripwire OpenSource through Version 2.3.1. As of this writing, the most current version of Tripwire Open Source is 2.3.1-2. If your Linux distribution of choice doesn't provide a reasonably current Tripwire package (Debian 2.2 and SUSE 7.3, for example, both ship with Tripwire 1.2, the 1994 Academic Source Release!), then I strongly recommend that you obtain, compile, and install the latest version. Needlessly running old security software is seldom a good idea; furthermore, as Linux users, we're eligible to use Tripwire Open Source. Tripwire Open Source can be downloaded as a source-code tarball at http://sourceforge.net/projects/tripwire/.

If you have *gcc* Version 3.0 or higher (Red Hat 9 and other recent Linux distributions; use `gcc --version` to find out what you have), you may have problems compiling some of Tripwire's C++ source. There are two solutions: patch and build the official source, or build from an alternative version.

13.2.1.1 Building from official source

Download the Tripwire Open Source tarball (http://prdownloads.sourceforge.net/tripwire/tripwire-2.3.1-2.tar.gz), then apply a patch that fixes the *gcc* problems:

```
# tar xvzf tripwire-2.3.1-2.tar.gz
# cd tripwire-2.3.1-2
# wget http://www.linuxfromscratch.org/patches/blfs/5.1/
tripwire-2.3.1-2-gcc3-build-fixes.patch
# patch -Np1 -i tripwire-2.3.1-2-gcc3-build-fixes.patch
```

Change to the source tree's *src* directory and make any necessary changes to the variable definitions in *src/Makefile*. Be sure to verify that the appropriate `SYSPRE` definition is uncommented (`SYSPRE = i686-pc-linux`, or `SYSPRE = sparc-linux`, etc.).

The Makefile relies on *gmake*, so check whether you have a copy of *gmake*, or a symbolic link from *gmake* to *make* somewhere in your *$PATH*. (Non-Linux Unices don't all come with GNU *make*, so Tripwire explicitly looks for *gmake*but on most Linux systems, this is simply called *make*). If you don't have such a link, create one.

Another thing to check for is a full set of subdirectories in */usr/share/man;* Tripwire will need to place manpages in *man4*, *man5*, and *man8*. On my Debian system, */usr/ man/man4* was missing; as a result, the installer created a file called */usr/man/man4*, which of course was actually a manpage that was incorrectly copied to that name rather than within it.

13.3. Other Integrity Checkers

As powerful and useful as Tripwire Open Source is, it's also complex and CPU-intensive. Furthermore, if you run "commercial" operating systems such as Windows or Solaris, no free version is available. Therefore, two 100% free and open source alternatives to Tripwire are worth mentioning.

The Advanced Intrusion Detection Environment (AIDE) is designed to meet and exceed Tripwire's functionality and is available from http://www.cs.tut.fi/~rammer/aide.html or http://aide.sourceforge.net. As of this writing its version number is 0.10, which reflects its youth: this may or may not have performance and stability implications. (For what it's worth, based on recent postings to the AIDE mailing list, AIDE seems to have more compile-time than runtime issues.) AIDE is 100% free to run on any of its supported platforms, whether in commercial or noncommercial settings.

IDS, Forensic Tool, or Both?

The premise behind this part of the chapter is that Tripwire and other integrity checkers can act as burglar alarms when run automatically at set intervals. Many people run integrity checkers in this way, as do I (admittedly, on a limited scale). But is this a reliable IDS methodology?

Not everyone thinks so. In his book *Network Intrusion Detection: An Analyst's Handbook* (Sams), Stephen Northcutt says:

To run a program such as Tripwire once at system build to get a file-integrity baseline is cheap, easy, and smart. To run Tripwire every day is costly because someone has to examine the results of the scan.

In other words, in Northcutt's opinion, you shouldn't run Tripwire checks routinely: only after you determine, through other means, that a breach has occurred. This approach limits Tripwire's role to assisting your forensics efforts (i.e., figuring out what happened and which files were affected).Then you're using it more like a security camera's backup tape.

I personally think using Tripwire only for forensics makes sense if you have reason to fear attackers skilled enough to trick Tripwire or you have too many servers from which to monitor frequent lengthy Tripwire reports. If either condition applies to you, do further research on the subject and consider a more sophisticated host-based IDS package such as the free Linux Intrusion Detection System (LIDS) (http://www.lids.org). Information on LIDS and many other IDS tools can be found in the "Tools" section at http://online.securityfocus.com.

A less Unix-centric alternative is *Fcheck*, which is available at http://www.geocities.com/fcheck2000/fcheck.html. *Fcheck* is a Perl script, which makes it both highly portable and very easy to customize. It's also extremely easy to configure: the configuration file is primarily a list of directories and files to scan and files and subdirectories to exclude. Command-line flags determine which attributes are checked for all of these: *Fcheck* has an "all or nothing" approach. (For you, that may or may not be a plus.)

On the downside, *Fcheck* has no built-in cryptographic functionality: unless you configure it to use an external program like *md5sum* (part of the GNU *textutils* package), it relies on simple CRC hashes, which are much easier to subvert than cryptographic hashes such as MD5 or Haval. Nor does it encrypt its database as Tripwire does. *Fcheck* was originally designed with change-control in mind, not security per se.

For this reason, *Fcheck*'s performance is very fast. While running any integrity checker without cryptographic hash checks is probably a bad idea on high-risk systems, it may be justifiable on systems on which you want a nominal check in place that uses minimal system resources. (Note that Tripwire can be configured this way, too.)

Another mitigating factor is frequency of checks: if your integrity checker runs every half hour, an attacker has only 30 minutes to disable or otherwise subvert it before their activity is caught by the checker. Thus, if using noncryptographic hashes makes it feasible for you to

13.4. Snort

Integrity checkers are more like security camera tapes than burglar alarms. They aren't nearly as useful during an attack as they are afterward; usually by the time the bad guys start changing files on a system, the attack has succeeded. This is because integrity checking is limited to the local system: it involves local files, not network packets. For more proactive intrusion detection ("intrusion in progress" or "attempted intrusion" detection), we need to monitor attempted and pending attacks while they're still on the wire*before* they make landfall on our systems.

The undisputed champion open source NIDS is Snort. Snort is a marvelous, versatile thing. First, as a packet sniffer (or, if you prefer the more formal term, "protocol analyzer"), Snort is to *tcpdump* what Homo sapiens is to Homo habilis: same basic genetic material, better brain. As a packet sniffer, Snort is extraordinarily fast, thorough, and user friendly (or at least geek friendly).

Second, Snort is a packet logger. Snort can preserve complete audit trails of network traffic, trails that name names and encase evidence in (figurative) acrylic blocks.

Third, Snort is a 100% customizable Network Intrusion Detection System with both a library of contributed attack signatures (*rules*) and a user-configurable rule engine. Snort not only holds its own with expensive commercial IDSes, but in some cases is better and faster than them. In this regard, Snort is the GIMP, Apache, and Nessus of IDSes.

Unlike some commercial IDSes, it's possible to write your own Snort rules and even your own inspection engines ("Snort plug-ins"). In this way, you're not dependent on anyone else to provide you with rules when a new exploit comes to your attention: you can write your own rules quickly and easily (provided you know something about TCP/IP networking, but that's a prerequisite of running any NIDS). This is an important feature, since new attacks are invented and reported all the time.

Snort can stand alone, but there are many useful enhancement packages with names such as Barnyard, ACID, and Sguil. I'll discuss these after we get down and dirty with Snort.

13.4.1. Obtaining, Compiling, and Installing Snort

Red Hat, Debian, and SUSE all provide binary packages of Snort in the current versions of their respective distributions. Of the three distributions, however, only SUSE ships a Snort package recent enough to support Snort v1.8's new rule format.

Since each new version of Snort is more sophisticated and therefore more effective at detecting suspicious network activity, I strongly recommend that you either obtain and compile the latest Snort source code or use the latest binary packages provided by the Snort team rather than those that come with your Linux distribution (even if you run SUSE).

13.4.1.1 Getting Snort source code and binaries

The official home and source of Snort code, binaries, rules, documentation, etc. is http://www.snort.org. Being an actively developed application, Snort has both stable and development code branches; as of this writing, the latest stable version is 2.2.0., but 2.3.0 should be out by the time you read this. Naturally, you should stick to the stable versions if you intend to run Snort on production (or otherwise important) systems.

If you navigate to the Snort web site's "downloads" page, you'll see links to the latest source tarballs. If you continue on to the site's "binaries" page, you'll find Snort binaries for Linux and Windows. (That's right, Snort runs on Windows!) Navigate to the "RPMs" page for current RPM packages for Red Hat and its derivatives (Mandrake, etc.). (To the best of my knowledge, these RPMs do *not* work on SUSE systems.)

13.4.1.2 Installing Snort RPMs

If you choose to install RPMs, you'll need at least one *snort*, which is a package of Snort's documentation, configuration files, and a bare-bones version of the *snort* binary itself. If you want a *snort* binary with support for MySQL databases, SNMP traps, or other advanced features, you'll also need one of the other RPMs on this page (*snort-snmp*, *snort-mysql*,

13.5. Resources

Amoroso, Ed. *Intrusion Detection*. Sparta, NJ: Intrustion.Net Books, 1999.

Excellent introduction to the subject.

Baker, Andrew, Brian Caswell, and Mike Poor. *Snort 2.1 Intrusion Detection, Second edition*. Syngress, 2004.

Up-to-date details on Snort, ACID, Barnyard, and Sguil.

Card, Rémy, Theodore Ts'o, and Stephen Tweedie. "Design and Implementation of the Second Extended Filesystem." (http://web.mit.edu/tytso/www/linux/ext2intro.html)

Excellent paper on the LinuxEXT2 filesystem; the section entitled "Basic File System Concepts" is of particular interest to Tripwire users.

Northcutt, Stephen and Judy Novak. *Network Intrusion Detection: An Analyst's Handbook*. Indianapolis: New Riders Publishing, 2001.

A very practical book with many examples showing system log excerpts and configurations of popular IDS tools.

http://www.chkrootkit.org/

Home of the c*hkrootkit* shell script and an excellent source of information about how to detect and defend against rootkits.

http://sourceforge.net/projects/tripwire

Project pages for Tripwire Open Source. The place to obtain the latest Tripwire Open Source code and documentation.

http://prdownloads.sourceforge.net/tripwire/tripwire-2.3.0-docs-pdf.tar.gz

Tripwire Open Source Manual and the Tripwire Open Source Reference Card in PDF format. Required reading! (If this link doesn't work, try http://sourceforge.net/project/showfiles.php?group_id=3130)

http://www.tripwire.org

Home page for Tripwire Open Source. Binaries for Linux available here.

http://www.tripwire.com/downloads/tripwire_asr/

Tripwire Academic Source Release download site.

http://securityportal.com/topnews/tripwire20000711.html

Article on using Tripwire Academic Source Release, by Jay Beale (principal developer of Bastille Linux).

http://sourceforge.net/projects/aide

Official web site for the Advanced Intrusion Detection Environment (AIDE).

http://www.geocities.com/fcheck2000/

Official web site for *Fcheck*, an extremely portable integrity checker written entirely in Perl.

Ranum, Marcus J. "Intrusion Detection & Network Forensics."

Presentation E1/E2 at the Computer Security Institute's 26th Annual Computer Security Conference and Exhibition, Washington, D.C., 17-19 Nov 1999.

Appendix A. Two Complete iptables Startup Scripts

These two scripts use *iptables* to configure *netfilter* on a DMZed server and on the firewall that protects it, assuming a simple inside-DMZ-outside architecture as described in Chapters Chapter 2 and Chapter 3. For the full example scenario to which these scripts apply, refer to Section 3.1.9 in Chapter 3.

Both of the examples in this appendix are available online at http://examples.oreilly.com/linuxss2/. Please remember that they are just models to use for developing your own firewall rules; they should never be dropped blindly onto a system.

The first script is for the bastion host *Woofgang*, a public FTP/HTTP server, shown in Example A-1.

Example A-1. iptables script for a bastion host running FTP and HTTP services

```
#! /bin/sh
# init.d/localfw
#
# System startup script for local packet filters on a bastion server
# in a DMZ (NOT for an actual firewall)
#
# Functionally the same as Example 3-10, but with SuSE-isms restored and
# with many more comments.
#
# Structurally based on SuSE 7.1's /etc/init.d/skeleton, by Kurt Garloff
#
# The following 9 lines are SuSE-specific
#
### BEGIN INIT INFO
# Provides: localfw
# Required-Start: $network $syslog
# Required-Stop:  $network $syslog
# Default-Start:  2 3 5
# Default-Stop:   0 1 2 6
# Description:    Start localfw to protect local heinie
### END INIT INFO
# /End SuSE-specific stuff (for now)

# Let's save typing & confusion with a couple of variables.
# These are NOT SuSE-specific in any way.

IP_LOCAL=208.13.201.2
IPTABLES=/usr/sbin/iptables
test -x $IPTABLES || exit 5

# The following 42 lines are SuSE-specific

# Source SuSE config
#  (file containing system configuration variables, though in SuSE 8.0 this
#     has been split into a number of files in /etc/rc.config.d)
. /etc/rc.config

# Determine the base and follow a runlevel link name.
base=${0##*/}
link=${base#*[SK][0-9][0-9]}

# Force execution if not called by a runlevel directory.
test $link = $base && START_LOCALFW=yes
test "$START_LOCALFW" = yes || exit 0
```

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 image on the cover of *Linux Server Security, Second Edition* is a caravan. An essential mode of transport for 19th-century Americans making the epic migration westward along the Oregon Trail, the typical family caravan was a covered wagon approximately 10 feet long and 4 feet wide. It was essential for one's caravan to accommodate a large supply of food, clothing, and household necessities; however, settlers were wise to keep luxury goods to a minimum to economize space and avoid taxing their oxen and horses. Living conditions in the caravan were usually quite cramped. The boxes and trunks that lined the floor of the wagon doubled as beds for the weary travelers. Completing the Oregon Trail was an arduous and hazardous endeavor, as casualties caused by perils ranging from cholera to firearm mishaps took the lives of many intrepid pioneers. Those that survived the harrowing 2,000-mile journey settled in the Willamette Valley of northwest Oregon, as well as in Washington State and California. Today, motorists can travel much of the length of this historic route on U.S. Highway 26.

Sanders Kleinfeld was the production editor and copyeditor for Linux Server Security, Second Edition. Linley Dolby was the proofreader. Matt Hutchinson and Claire Cloutier provided quality control. Julie Hawks wrote the index.

Emma Colby designed the cover of this book, based on a series design by Hanna Dyer and Edie Freedman. The cover image is a 19th-century engraving from The American West in the 19th Century (Dover). Emma Colby produced the cover layout with Adobe InDesign CS using Adobe's ITC Garamond font.

Melanie Wang designed the interior layout. The chapter opening images are from the Dover Pictorial Archive, Marvels of the New West: A Vivid Portrayal of the Stupendous Marvels in the Vast Wonderland West of the Missouri River, by William Thayer (The Henry Bill Publishing Co., 1888) and The Pioneer History of America: A Popular Account of the Heroes and Adventures, by Augustus Lynch Mason, A.M. (The Jones Brothers Publishing Company, 1884).

This book was converted to FrameMaker 5.5.6 by Julie Hawks 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 MX and Adobe Photoshop CS. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Sanders Kleinfeld.

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

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Team LiB ◀ PREVIOUS NEXT ▶

Index

Team LiB ◀ PREVIOUS NEXT ▶

Index

Index

Index

Team LiB PREVIOUS NEXT

Index

Index

Index

Index

Index

Index