Building Secure Servers with Linux

By

Michael D. Bauer

Publisher : O'Reilly
Pub Date : October 2002
ISBN : 0-596-00217-3
Pages : 448

Slots : 1

This book provides a unique balance of "big picture" principles that transcend specific software packages and version numbers, and very clear procedures on securing some of those software packages. An all-inclusive resource for Linux users who wish to harden their systems, the book covers general security as well as key services such as DNS, the Apache Web server, mail, file transfer, and secure shell.

Building Secure Servers with Linux

By

Michael D. Bauer

Publisher : O'Reilly
Pub Date : October 2002
ISBN : 0-596-00217-3
Pages : 448

Slots : 1

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

Printed in the United States of America.

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

Sebastopol, CA 95472.

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

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 "Building Secure

Servers."[] 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:

[] 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, risks, and 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/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-nineties 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*. About a year and a half 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.

My folly is your gain: while "Paranoid Penguin" readers may recognize certain diagrams and even paragraphs from that material, I've spent a great deal of effort reresearching and expanding all of it, including retesting all examples and procedures. I've added entire (lengthy) chapters on topics I haven't covered at all in the magazine, and I've 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 yours.

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 between those two types of users, but the problems, risks, and threats they need to consider have more in common than not. 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 administrators whether 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 7, SuSE 7, and Debian 2.2 GNU/Linux distributions are addressed in detail.

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

- Linux distributions besides Red Hat, SuSE, and Debian, although with 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)

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*, *dselect*, 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 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, 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.

Conventions Used in This Book

I use the following font conventions in this book:

Italic

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

Boldface

Indicates names of GUI items, such as window names, buttons, menu choices, etc.

`Constant width`

Indicates command lines and options that should be typed verbatim; names and keywords in system scripts, including commands, parameter names, and variable names; and XML element tags

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

This icon indicates a warning or caution.

Request for Comments

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

O'Reilly & Associates, 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, or any additional information. You can access this page at:

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

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 the O'Reilly web site at:

http://www.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 4, and all of Chapter 6, brilliantly, in my humble opinion. Bill has added a great deal of real-world experience, skill, and humor to those two 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.

I absolutely could not have survived juggling my day job, fatherly duties, magazine column, and resulting sleep deprivation without an exceptionally patient and energetic wife. This book therefore owes its very existence to Felice Amato Bauer. I'm

grateful to her for, among many other things, encouraging me to pursue my book proposal and then for pulling a good deal of *my* parental weight in addition to her own after the proposal was accepted and I was obliged to actually *write* the thing.

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: Chapter 1 through Chapter 5, plus Chapter 8, Chapter 10, and Chapter 11 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 has been strongly influenced by two giants of the field whom I also want to thank: Bruce Schneier, to whom we all owe a great debt for his ongoing contributions not only to security technology but, even more importantly, to security *thinking*; and Dr. Martin R. Carmichael, whose irresistible passion for and unique outlook on what constitutes good security has had an immeasurable impact on my work.

It should but won't go without saying that I'm very grateful to Andy Oram and

O'Reilly & Associates for this opportunity and for their marvelous support, guidance, and patience. The impressions many people have of O'Reilly as 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, and Joshua Ball, in particular, helped immensely to improve the book's accuracy and usefulness.

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; the entire SAE Design Team (you know who you are *or do you?*); Marty J. Wolf at Bemidji State University; John B. Weaver (whom nobody initially believes can possibly be *that* cool, but they soon realize he can `cause he is); the Reverend

Gonzo at Musicscene.org; Richard Vernon and Don Marti at *Linux Journal*; Jay Gustafson of Ingenious Networks; Tim N. Shea (who, in my day job, had the thankless task of standing in for me while I finished this

book), and, of course, my dizzyingly adept pals Brian Gilbertson, 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 against 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 stupid and 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 considered 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*, *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 comprise it.

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 customers' 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's being compromised and then used to attack someone else's systems, it may be worth spending some time and effort on 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.

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 may result in financial loss. For example, an engineer who emails details about a new invention to a colleague without using encryption may be risking her ability to be first-to-market with a particular technology should those details fall into a competitor's possession.

In other cases, data disclosure might result in additional security exposures. For example, a system administrator who uses *telnet* (an unencrypted protocol) for remote administration may be risking disclosure of his logon credentials to unauthorized eavesdroppers who could subsequently use those credentials to gain illicit access to critical systems.

1.1.2.2 Data integrity

Regardless of the need to keep a given piece or body of data secret, you may need to ensure that the data isn't altered in any way. We most often think of data integrity in the context of secure data transmission, but important data should be protected from tampering even if it *doesn't* need to be transmitted (i.e., when it's stored on a system with no network connectivity).

Consider the ramifications of the files in a Linux system's */etc* directory being

altered by an unauthorized user: by adding her username to the *wheel* entry in */etc/group*,

a user could grant herself the right to issue the command *su root* -. (She'd still need

the root password, but we'd prefer that she not be able to get even this far!) This is an example of the need to preserve the integrity of local data.

Let's take another example: a software developer who makes games available for free on his public web site may not care who downloads the games, but almost certainly doesn't want those games being changed without his knowledge or permission. Somebody else could inject virus code into it (for which, of course, the developer would be held accountable).

We see then that data integrity, like data confidentiality, may be desired in any number and variety of contexts.

1.1.2.3 System integrity

System

integrity refers to whether a computer system is being used as its administrators intend (i.e., being used only by authorized users, with no greater privileges than they've been assigned). System integrity can be undermined both by remote users (e.g., connecting over a network) and by local users escalating their own level of privilege on the system.

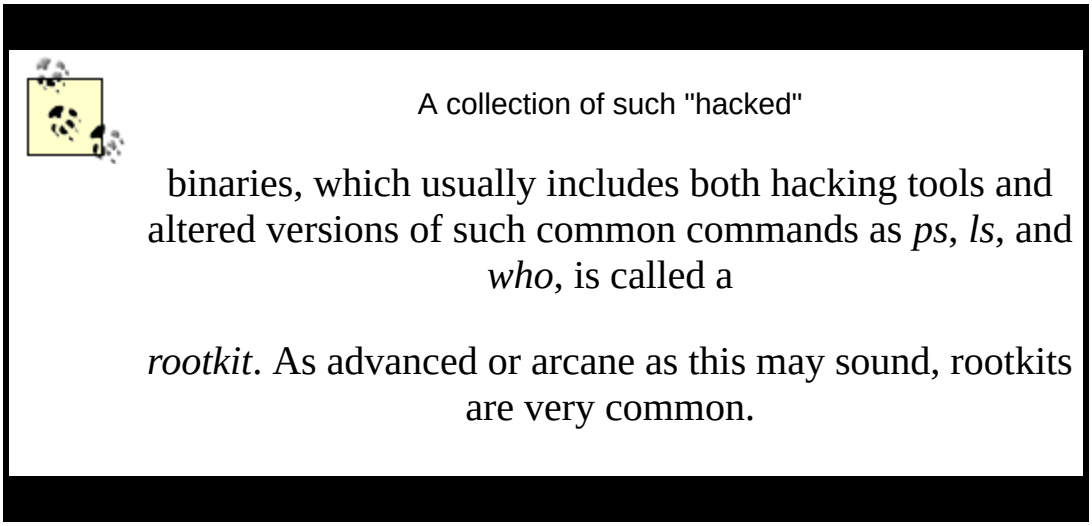
The state of "compromised system integrity" carries with it two important

assumptions:

- Data stored on the system or available to it via trust relationships (e.g., NFS shares) may have also been compromised; that is, such data can no longer be considered confidential or untampered with.
- System executables themselves may have also been compromised.

The second assumption is particularly scary: if you issue the command *ps auxw*

to view all running processes on a compromised system, are you really seeing everything, or could the *ps* binary have been replaced with one that conveniently omits the attacker's processes?



Industry best practice (not to mention common sense) dictates that a compromised system should undergo

"bare-metal recovery"; i.e., its hard drives should be erased, its operating system should be reinstalled from source media, and system data should be restored from backups dated before the date of compromise, if at all. For this reason, system integrity is one of the most important security goals.

There is seldom a quick, easy, or cheap way to recover from a system compromise.

1.1.2.4 System/network availability

The

other category of security goals we'll discuss is availability. "System availability"

is short for "the system's

availability to users." A network or system that does not respond to user requests is said to be "unavailable."

Obviously, availability is an important goal for all networks and systems. But it may be more important to some than it is to others.

An online retailer's web site used to process customers' orders, for example, requires a much greater assurance of availability than a

"brochure" web site, which provides

a store's location and hours of operation but isn't actually part of that store's

core business. In the former case, unavailability equals lost income, whereas in the latter case, it amounts mainly to inconvenience.

Availability may be related to other security goals. For example, suppose an attacker knows that a target network is protected by a firewall with two vulnerabilities: it passes all traffic without filtering it for a brief period during startup, and it can be made to reboot if bombarded by a certain type of network packet. If the attacker succeeds in triggering a firewall reboot, he will have created a brief window of opportunity for launching attacks that the firewall would ordinarily block.

This is an example of someone targeting system availability to facilitate other attacks. The reverse can happen, too: one of the most common reasons cyber-vandals compromise systems is to use them as launch-points for

"

Distributed Denial of Service" (DDoS) attacks, in which large numbers of software agents running on compromised systems are used to overwhelm a single target host.

The good news about attacks on system availability is that once the attack ends, the system or network can usually recover very quickly.

Furthermore, except when combined with other attacks, Denial of Service attacks seldom directly affect data confidentiality or data/system integrity.

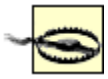
The bad news is that many types of DoS attacks are all but impossible to prevent due to the difficulty of distinguishing them from very large volumes of "legitimate"

traffic. For the most part, deterrence (by trying to identify and punish attackers) and redundancy in one's

system/network design are the only feasible defenses against DoS

attacks. But even then, redundancy doesn't make DoS

attacks impossible; it simply increases the number of systems an attacker must attack simultaneously.



When you design a

redundant system or network (never a bad idea), you should *assume that attackers will figure out the system/network topology if they really want to*. If you assume they won't and count this assumption as a major part of your security plan, you'll be guilty of "security

through obscurity." While true

secrecy is an important variable in many security equations, mere

"obscurity" is seldom very

effective on its own.

1.1.3 Threats

Who might attack your system, network, or data? Cohen et al,[2] in their scheme for classifying information security threats, provide a list of "actors" (threats), which

illustrates the variety of attackers that any networked system faces.

These attackers include the mundane (insiders, vandals, maintenance people, and nature), the sensational (drug cartels, paramilitary groups, and extortionists), and all points in between.

[2] 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://heat.ca.sandia.gov/papers/cause-and-effect.html.

As you consider potential attackers, consider two things. First, almost every type of attacker presents some level of threat to every Internet-connected computer. The concepts of distance, remoteness, and obscurity are radically different on the Internet than in the physical world, in terms of how they apply to escaping the notice of random attackers. Having an

"uninteresting" or

"low-traffic" Internet presence is

no protection at all against attacks from strangers.

For example, the level of threat that drug cartels present to a hobbyist's basement web server is probably minimal, but shouldn't be dismissed altogether. Suppose a system cracker in the employ of a drug cartel wishes to target FBI systems via intermediary (compromised) hosts to make his attacks harder to trace.

Arguably, this particular scenario is unlikely to be a threat to most of us. But impossible? Absolutely not. The technique of relaying attacks across multiple hosts is common and time-tested; so is the practice of scanning ranges of IP addresses registered to Internet Service Providers in order to identify vulnerable home and business users. From that viewpoint, a hobbyist's web server is likely to be scanned for vulnerabilities on a regular basis by a wide variety of potential attackers. In fact, it's arguably likely to be scanned *more heavily* than "higher-profile" targets. (This is

not an exaggeration, as we'll see in our discussion of Intrusion Detection in Chapter 11.) The second thing to consider in evaluating threats is that it's impossible to anticipate all possible or even all likely types of attackers. Nor is it possible to anticipate all possible avenues of attack (vulnerabilities). That's okay: the point in threat analysis is not to predict the future; it's to think about and analyze threats with greater depth than "someone out there might hack into this system for some reason."

You can't anticipate everything, but you can take reasonable steps to maximize your awareness of risks that are obvious, risks that are less obvious but still significant, and risks that are unlikely to be a problem but are easy to protect against.

Furthermore, in the process of analyzing these risks, you'll also identify risks that are unfeasible to protect against regardless of their significance.

That's good, too: you can at least create recovery plans for them.

1.1.4 Motives

Many of the threats are fairly obvious and easy to understand. We all know that business competitors wish to make more money and disgruntled ex-employees often want revenge for perceived or real wrongdoings. Other motives aren't so easy to pin down. Even though it's seldom addressed directly in threat analysis, there's some value in discussing the motives of people who commit computer crimes.

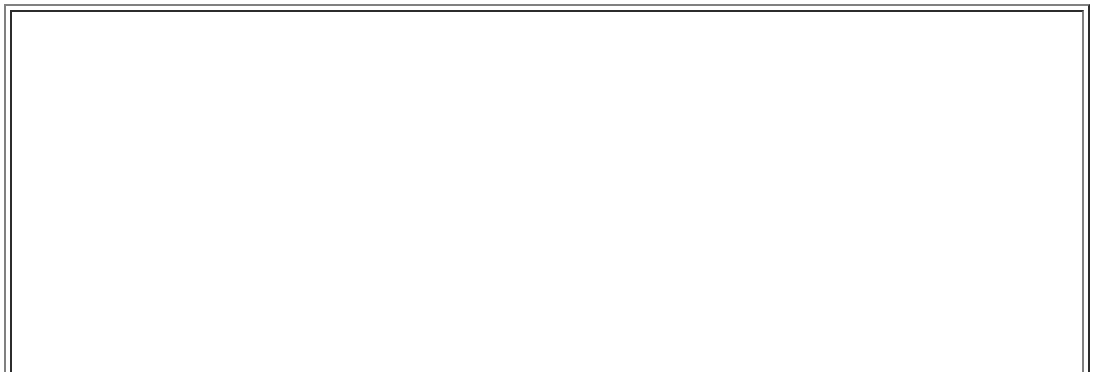
Attacks on data confidentiality, data

integrity, system integrity, and system

availability correspond pretty convincingly to the physical-world crimes of espionage, fraud, breaking and entering, and sabotage, respectively. Those crimes are committed for every imaginable motive.

As it happens, computer criminals are driven by pretty much the same motives as "real-life" criminals

(albeit in different proportions). For both physical and electronic crime, motives tend to fall into a small number of categories.



Why All the Analogies to "Physical" Crime?

No doubt you've noticed that I frequently draw analogies between electronic crimes and their conventional equivalents. This isn't just a literary device.

The more you leverage the common sense you've acquired in "real life," the more

effectively you can manage information security risk. Computers and networks are built and used by the same species that build and use buildings and cities: human beings. The venues may differ, but the behaviors (and therefore the risks) are always analogous and often identical.

1.1.4.1 Financial motives

One of the most compelling and understandable reasons for computer crime is money. Thieves use the Internet to steal and barter credit card numbers so they can bilk credit card companies (and the merchants who subscribe to their services). Employers pay industrial spies to break into their competitors' systems and steal proprietary data. And the German hacker whom Cliff Stoll helped

track down (as described in Stoll's book,

The Cuckcoo's Egg) hacked into U.S. military and defense-related systems for the KGB in return for money to support his drug habit.

Financial motives are so easy to understand that many people have trouble contemplating any *other* motive for computer crime. No security professional goes more than a month at a time without being asked by one of their clients "Why would anybody want to break into

my system? The data isn't worth anything to anyone but me!"

Actually, even these clients usually do have data over which they'd rather not lose control (as they tend to realize when you ask, "Do you mean that this data is *public*?") But financial motives do not account for all computer crimes or even for the most elaborate or destructive attacks.

1.1.4.2 Political motives

In recent years, Pakistani attackers have targeted Indian web sites (and vice versa) for defacement and Denial of Service attacks, citing resentment against India's treatment of Pakistan as the reason. A few years ago, Serbs were reported to have attacked NATO's information systems (again, mainly web sites) in reaction to NATO's air strikes during the war in Kosovo. Computer crime is very much a part of modern human conflict; it's unsurprising that this includes military and political conflict.

It should be noted, however, that attacks motivated by the less lofty goals of bragging rights and plain old mischief-making are frequently carried out with a pretense of patriotic, political, or other "altruistic" aims if

impairing the free speech or other lawful computing activities of groups with which one disagrees can be called altruism. For example, supposedly political web site defacements, which also involve self-aggrandizing boasts, greetings to other web site defacers, and insults against rival web site defacers, are far more common than those that contain only political messages.

1.1.4.3 Personal/psychological motives

Low self-esteem, a desire to impress others, revenge against society in general or a particular company or organization, misguided curiosity, romantic misconceptions of the "computer underground" (whatever that means anymore), thrill-seeking, and plain old misanthropy are all common motivators, often in combination. These are examples of personal motives motives that are intangible and sometimes inexplicable, similar to how the motives of shoplifters who can afford the things they steal are inexplicable.

Personal and psychological reasons tend to be the motives of virus writers, who are often skilled programmers with destructive tendencies. Personal motives also fuel most "script kiddies": the unskilled, usually teenaged vandals responsible for many if not most external attacks on Internet-connected systems. (As in the world of nonelectronic vandalism and other property crimes, true artistry among system crackers is fairly rare.)

Script Kiddies

Script kiddies are so named due to their reliance on "canned" exploits, often in the

form of Perl or shell scripts, rather than on their own code. In many cases, kiddies aren't even fully aware of the proper use (let alone the full ramifications) of their tools.

Contrary to what you might therefore think, script

kiddies are a major rather than a minor threat to Internet-connected systems. Their intangible motivations make them highly unpredictable; their limited skill sets make them far more likely to unintentionally cause serious damage or dysfunction to a compromised system than an expert would cause. (Damage equals evidence, which professionals prefer not to provide needlessly.)

Immaturity adds to their potential to do damage: web site defacements and Denial-of-Service attacks, like graffiti and vandalism, are mainly the domain of the young. Furthermore, script kiddies who are minors usually face minimal chances of serving jail time or even receiving a criminal record if caught.

The Honeynet Project, whose mission is "to learn the tools, tactics, and motives of the blackhat community, and share those lessons learned"

(http://www.honeynet.org), even has a Team Psychologist: Max Kilger, PhD. I mention Honeynet in the context of psychology's importance in network threat models, but I highly recommend the Honeynet Team's web site as a fascinating and useful source of real-world Internet security data.

We've discussed some of the most common motives of computer crime, since understanding probable or apparent motives helps predict the course of an attack in progress and in defending against common, well-understood threats. If a given vulnerability is well known and easy to exploit, the only practical assumption is that it *will* be exploited sooner or later. If you understand the wide range of motives that potential attackers can have, you'll be less tempted to wrongly dismiss a given vulnerability as "academic."

Keep motives in mind when deciding whether to spend time applying software patches against vulnerabilities you think unlikely to be targeted on your system. There is seldom a good reason to forego protections (e.g., security patches) that are relatively cheap and simple.

Before we leave the topic of motives, a few words about *degrees* of motivation. I mentioned in the footnote on the first page of this chapter that most attackers (particularly script kiddies) are easy to keep out, compared to the dreaded "sufficiently motivated

attacker." This isn't just a

function of the attacker's skill level and goals: to a large extent, it reflects *how badly* script kiddies and other random vandals want a given attack to succeed, as opposed to how badly a focused, determined attacker wants to get in.

Most attackers use automated tools to

scan large

ranges of IP addresses for known vulnerabilities. The systems that catch their attention and, therefore, the full focus of their efforts are "easy kills": the more systems

an attacker scans, the less reason they have to focus on any but the most vulnerable hosts identified by the scan. Keeping your system current (with

security

patches) and otherwise "hardened,"

as recommended in Chapter 3, will be sufficient protection against the majority of such attackers.

In contrast, focused attacks by strongly motivated attackers are by definition much harder to defend against. Since all-out attacks require much more time, effort, and skill than do script-driven attacks, the average home user generally needn't expect to become the target of one. Financial institutions, government agencies, and other

"high-profile" targets, however,

must plan against both indiscriminate and highly motivated attackers.

1.1.5 Vulnerabilities and Attacks Against Them

Risk isn't just about assets and attackers: if an asset has no

vulnerabilities

(which is impossible, in practice, if it resides on a networked system), there's no risk no matter how many prospective attackers there are.

Note that a vulnerability only represents a potential, and it remains so until someone figures out how to exploit that vulnerability into a successful attack. This is an important distinction, but I'll admit that in threat analysis,

it's common to lump vulnerabilities and actual attacks together.

In most cases, it's dangerous *not* to: disregarding a known vulnerability because you haven't heard of anyone attacking it yet is a little like ignoring a bomb threat because you can't hear anything ticking. This is why vendors who dismiss vulnerability reports in their products as "theoretical" are usually ridiculed

for it.

The question, then, isn't whether a vulnerability *can* be exploited, but whether foreseeable exploits are straightforward enough to be widely adopted. The worst-case scenario for any software vulnerability is that exploit code will be released on the Internet, in the form of a simple script or even a GUI-driven binary program, sooner than the software's developers can or will release a patch.

If you'd like to see an explicit enumeration of the wide range of vulnerabilities to which your systems may be subject, I again recommend the article I cited earlier by Fred Cohen and his

colleagues

(http://heat.ca.sandia.gov/papers/cause-and-effect.html). Suffice it to say here that they include physical security (which is important but often overlooked), natural phenomena, politics, cryptographic weaknesses, and, of course, plain old software bugs.

As long as Cohen's list is, it's a necessarily incomplete list. And as with attackers, while many of these vulnerabilities are unlikely to be applicable for a given system, few are

impossible.

I haven't reproduced the list here, however, because my point isn't to address all possible

vulnerabilities in every system's security planning.

Rather, of the myriad possible attacks against a given system, you need to identify and address the following:

1. Vulnerabilities that are clearly applicable to your system and must be mitigated immediately
2. Vulnerabilities that are likely to apply in the future and must be planned against
3. Vulnerabilities that seem unlikely to be a problem later but are easy to mitigate

For example, suppose you've installed the imaginary Linux distribution Bo-Weevil Linux from CD-ROM. A quick way to identify and mitigate known, applicable vulnerabilities (item #1 from the previous list) is to download and install the latest security patches from the Bo-Weevil web site. Most (real) Linux distributions can do this via automated software tools, some of which are described in Chapter 3.

Suppose further that this host is an SMTP gateway (these are described in detail in Chapter 7).

You've installed the latest release of Cottonmail 8.9, your preferred (imaginary) Mail Transport Agent (MTA), which has no known security bugs. You're therefore tempted to skip configuring some of its advanced security features, such as running in a restricted subset of the filesystem (i.e., in a "chroot jail," explained in Chapter 6).

But you're aware that MTA applications have historically been popular entry points for attackers, and it's certainly possible that a buffer overflow or similar vulnerability may be discovered in Cottonmail 8.9 one that the bad guys discover before the Cottonmail team does. In other words, this falls into category #2 listed earlier: vulnerabilities that don't currently apply but may later. So you spend an extra hour reading manpages and configuring your MTA to operate in a chroot jail, in case it's compromised at some point due to an as-yet-unpatched security bug.

Finally, to keep up with emerging threats, you subscribe to the official Bo-Weevil Linux Security Notices email list. One day you receive email from this list describing an Apache vulnerability that can lead to unauthorized root access. Even though you don't plan on using this host as a web server, Apache is installed, albeit not configured or active: the Bo-Weevil installer included it in the default installation you chose, and you disabled it when you hardened the system.

Therefore, the vulnerability doesn't apply now and probably won't in the future. The patch, however, is trivially acquired and applied, thus it falls into category #3 from our list. There's no reason for you not to fire up your autoupdate tool and apply the patch. Better still, you can uninstall Apache altogether, which mitigates the Apache vulnerability completely.

Single Loss x expected Annual = Annualized Loss

Expectency (cost) Rate of Occurrences Expectancy (cost/year)

950 $/incident * 0.5 incidents/yr = 475 $/yr

The ALE for Denial of Service attacks on the example business' SMTP gateway is thus $475 per year.

Now, suppose your friends are trying to talk you into replacing your homegrown Linux firewall with a commercial firewall: this product has a built-in SMTP proxy that will help minimize but not eliminate the SMTP gateway's exposure to DoS attacks. If that commercial product costs $5,000, even if its cost can be spread out over three years (at 10% annual interest, this would total $6,374), such a firewall upgrade would not appear to be justified by this single risk.

Figure 1-1 shows a more complete threat analysis for our hypothetical business' SMTP gateway, including not only the ALE we just calculated, but also a number of others that address related assets, plus a variety of security goals.

Figure 1-1. Sample ALE-based threat model

Asset	Security Goal	Vulnerability	SLE ($/incident)	ARO (incdts/yr)	ALE ($/yr)
SMTP Gateway	System Integrity	sendmail bugs	$2,400	0.5	$1,200
		misc. system bugs	$2,400	0.5	$1,200
	System Availability	DOS Attacks	$950	0.5	$475
Confidential email (customer account info)	Data Confidentiality	Eavesdropping on Internet or ISP	$50,000	2	$100,000
		Compromise of SMTP Gateway	$50,000	0.5	$25,000
		Malicious insider	$150,000	0.33	$49,500
	Data Integrity	Forged email to/from customer	$10,000	1	$10,000
		In-transit alteration on Internet or ISP	$10,000	0.25	$2,500
		Compromise of SMTP Gateway	$10,000	0.5	$5,000
Non-confidential email (operations info)	Data Integrity Data Integrity	In-transit alteration on Internet or ISP	$3,000	0.25	$750
		Compromise of SMTP Gateway	$3,000	0.5	$1,500

In this sample analysis, customer data in the form of confidential email is the most valuable asset at risk; if this is eavesdropped or tampered with, customers could be lost, resulting in lost revenue. Different perceived loss potentials are reflected in the Single Loss Expectancy figures for different vulnerabilities; similarly, the different estimated Annual Rates of Occurrence reflect the relative likelihood of each vulnerability actually being exploited.

Since the sample analysis in Figure 1-1 is in the form of a spreadsheet, it's easy to sort the rows arbitrarily. Figure 1-2 shows the same analysis sorted by vulnerability.

Figure 1-2. Same analysis sorted by vulnerability

Asset	Security Goal	Vulnerability	SLE ($/incident)	ARO (incdts/yr)	ALE ($/yr)
SMTP Gateway	System Integrity	sendmail bugs	$2,400	0.5	$1,200
SMTP Gateway	System Integrity	misc. system bugs	$2,400	0.5	$1,200
Confidential email (customer account info)	Data Confidentiality	Malicious insider	$150,000	0.33	$49,500
Confidential email (customer account info)	Data Integrity	In-transit alteration on Internet or ISP	$10,000	0.25	$2,500
Non-confidential email (operations info)	Data Integrity	In-transit alteration on Internet or ISP	$3,000	0.25	$750
Confidential email (customer account info)	Data Integrity	Forged email to/from customer	$10,000	1	$10,000
Confidential email (customer account info)	Data Confidentiality	Eavesdropping on Internet or ISP	$50,000	2	$100,000
SMTP Gateway	System Availability	DOS Attacks	$950	0.5	$475
Confidential email (customer account info)	Data Confidentiality	Compromise of SMTP Gateway	$50,000	0.5	$25,000
Confidential email (customer account info)	Data Integrity	Compromise of SMTP Gateway	$10,000	0.5	$5,000
Non-confidential email (operations info)	Data Integrity	Compromise of SMTP Gateway	$3,000	0.5	$1,500

This is useful for adding up ALEs associated with the same vulnerability. For example, there are two ALEs associated with in-transit alteration of email while it traverses the Internet or ISPs, at $2,500 and $750, for a combined ALE of $3,250. If a training consultant will, for $2,400, deliver three half-day seminars for the company's workers on how to use free GnuPG software to sign and encrypt documents, the trainer's fee will be justified by this vulnerability alone.

We also see some relationships between ALEs for different vulnerabilities. In Figure 1-2 we see that the bottom three ALEs all involve losses caused by compromising the SMTP gateway. In other words, not only will a SMTP gateway compromise result in lost productivity and expensive recovery time from consultants ($1,200 in either ALE at the top of Figure 1-2); it will expose the business to an additional $31,500 risk of email data compromises for a total ALE of $32,700.

Clearly, the Annualized Loss Expectancy for email eavesdropping or tampering caused by system compromise is high. ABC Corp. would be well advised to call that $2,400 trainer immediately!

There are a few problems with relying on the ALE as an analytical tool. Mainly, these relate to its subjectivity; note how often in the example I used

words like "unlikely" and "reasonable." Any ALE's significance, therefore, depends much less on empirical data than it does on the experience and knowledge of whoever's calculating it. Also, this method doesn't lend itself too well to correlating ALEs with one another (except in short lists like Figures 1-1 and 1-2).

The ALE method's strengths, though, are its simplicity and flexibility. Anyone sufficiently familiar with their own system architecture, operating costs, and current trends in IS security (e.g., from reading CERT advisories and incident reports now and then) can create lengthy lists of itemized ALEs for their environment with very little effort. If such a list takes the form of a spreadsheet, ongoing tweaking of its various cost and frequency estimates is especially easy.

Even given this method's inherent subjectivity (which isn't completely avoidable in practical threat analysis techniques), it's extremely useful as a tool for enumerating, quantifying, and weighing risks. It's especially useful for expressing risks in terms that managers can understand. A well-constructed list of Annualized Loss Expectancies can help you not only to focus your IS security expenditures on the threats likeliest to affect you in ways that matter; it can also help you to get and keep the budget you need to pay for those expenditures.

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:

1. Obtain backup tapes from ABC's file server.
2. Intercept email between ABC Corp. and their customers.
3. 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).

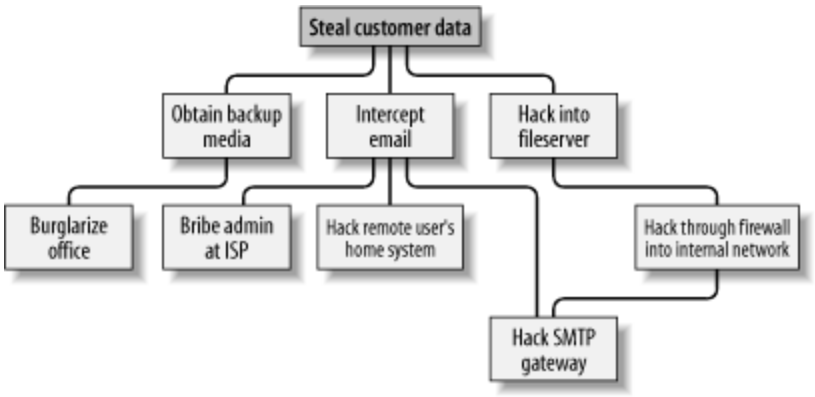
Figure 1-3. Root node with three leaf nodes

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.

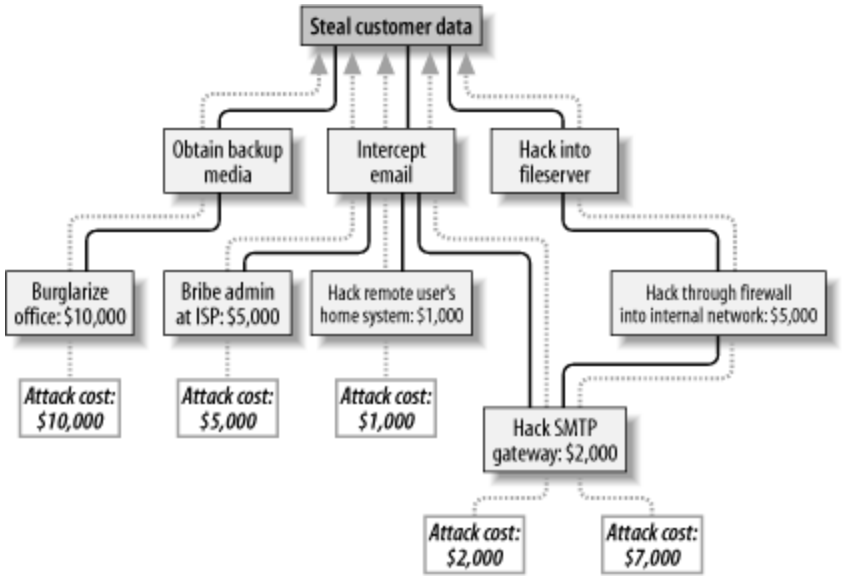
After an attack tree has been mapped to the desired level of detail, you can start quantifying the leaf nodes. For example, you could attach a "

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 employee will be concerned about losing her job and getting a criminal record.

Hacking is a bit different, however. Hacking through a firewall takes more skill than the average script kiddie has, and it will take some time and effort. Therefore, this is an expensive goal. But hacking an SMTP gateway should be easier, and if one or more remote users can be identified, the chances are good that the user's home computer will be easy to compromise. These two goals are therefore much cheaper.

Based on the cost of hiring the right kind of criminals to perform these attacks, the most promising attacks in this example are hacking the SMTP gateway and hacking remote users. ABC Corp., it seems, had better take a close look at their perimeter network

architecture, their SMTP server's system security, and their remote-access policies and practices.

Cost, by the way, is not the only type of value you can attach to leaf nodes. Boolean values such as

"feasible" and

"not feasible" can be used: a

"not feasible" at any point on an

attack path indicates that you can dismiss the chance of an attack on that path with some safety. Alternatively, you can assign effort indices, measured in minutes or hours. In short, you can analyze the same attack tree in any number of ways, creating as detailed a picture of your vulnerabilities as you need to.

Before we leave the subject of attack tree threat modeling, I should mention the importance of considering different types of attackers.

The cost estimates in Figure 1-5 are all based on the assumption that the attacker will need to hire others to carry out the various tasks. These costs might be computed very differently if the attacker is himself a skilled system cracker; in such a case, time estimates for each node might be more useful.

So, which type of attacker should you model against? As many different types as you realistically think you need to. One of the great strengths of this method is how rapidly and easily attack trees can be created; there's no reason to quit after doing only one.

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 of 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 dye bombs: 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, and 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 to add a layer of authentication to a web application with inadequate access controls).

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.

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://heat.ca.sandia.gov/papers/cause-and-effect.html.

Chapter 2. Designing Perimeter Networks

A well-designed perimeter network (the part or parts of your internal network that has direct contact with the outside world e.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 that's where firewalls go, 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, then 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.

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 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 of 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 (although these tend to be prepackaged routers or computers), 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. 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 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 some combination of their Source IP Address, Destination IP Address, Source Port, and their Destination Port (Service). Application data is not considered; i.e., intentionally malformed packets are not necessarily noticed, assuming their IP

headers can be read. Packet-filtering is a necessary part of nearly all firewalls' functionality, but is not considered, by itself, to be sufficient protection against any but the most straightforward attacks. Most routers (and many low-end firewalls) are limited to packet-filtering.

Perimeter Network

The portion or portions of an organization's network that are directly connected to the Internet, plus any "DMZ" networks (see earlier in this

list). This isn't a precise term, but if you have much trouble articulating where your network's perimeter ends and your protected/trusted network begins, you may need to re-examine your network architecture.

Proxying

An intermediary in all interactions of a given service type (ftp, http, etc.) between internal hosts and untrusted/external hosts. In the case of SOCKS, which uses

Generic Service Proxies, the proxy may authenticate each connection it proxies. In the case of Application Gateways, the proxy intelligently parses Application-Layer data for anomalies.

Stateful packet-filtering

At its simplest, the tracking of TCP sessions; i.e., using packets' TCP header information to determine which packets belong to which transactions, and thus filtering more effectively. At its most sophisticated, stateful packet-filtering refers to the tracking of not only TCP headers, but also some amount of Application-Layer information (e.g., end-user commands) for each session being inspected. Linux's iptables include modules that can statefully track most kinds of TCP transactions and even some UDP transactions.

TCP/IP Stack Attack

A network attack that exploits vulnerabilities in its target's TCP/IP stack (kernel-code or drivers).

These are, by definition, OS specific: Windows systems, for example, tend to be vulnerable to different stack attacks than Linux systems.

That's a lot of jargon, but it's useful jargon (useful enough, in fact, to make sense of the majority of firewall vendors' propaganda!). Now

we're ready to dig into DMZ architecture.

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.

Such 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

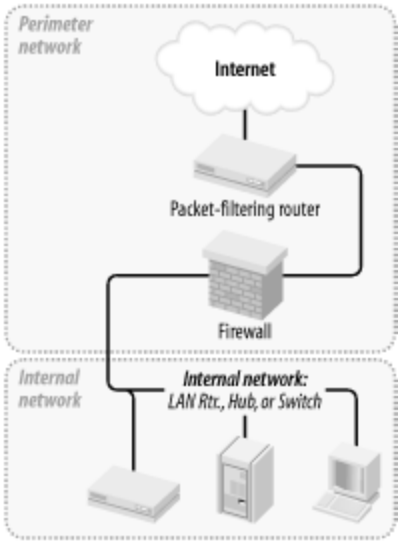
The most common firewall architecture one tends to see nowadays is the one illustrated in Figure 2-1. In this diagram, we have a

packet-filtering router that acts as the initial, but not sole, line of defense. Directly behind this router is a "proper"

firewall in this case a Sun SparcStation running, say, Red Hat 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 doesn't directly expose other internal hosts to attack, but it does magnify the consequences of an 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? In a DMZ

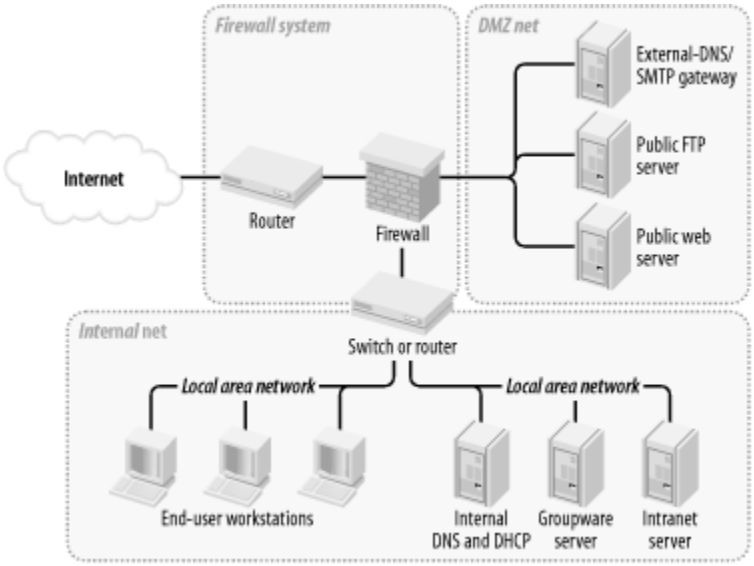
(DeMilitarized Zone) network!

2.2.2 The "Three-Homed Firewall" DMZ Architecture

At its simplest, a DMZ is any network reachable by the public but isolated from one's internal network. Ideally, however, a DMZ is also protected by the firewall. Figure 2-2 shows my preferred Firewall/DMZ

architecture.

Figure 2-2. Single-firewall DM2 architecture



In Figure 2-2, we have a

three-homed host as our firewall. Hosts providing publicly accessible services are in their own network with a dedicated connection to the firewall, and the rest of the corporate network face a different firewall interface. If configured properly, the firewall uses different rules in evaluating traffic:

- From the Internet to the DMZ
- From the DMZ to the Internet
- From the Internet to the Internal Network
- From the Internal Network to the Internet
- From the DMZ to the Internal Network
- From the Internal Network to the DMZ

This may sound like more administrative overhead than that associated with internally hosted or firewall-hosted services, but it's potentially much simpler since the DMZ can be

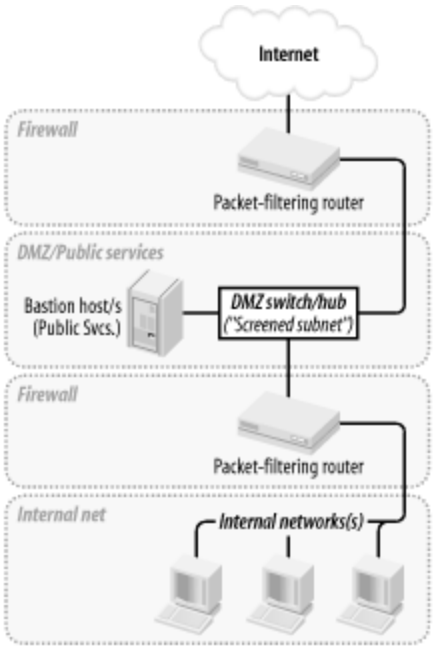
treated as a single logical entity. In the case of internally hosted services, each host must be considered individually (unless all the services are located on a single IP network whose address is distinguishable from other parts of the internal network).

2.2.3 A Weak Screened-Subnet Architecture

Other architectures are sometimes used, and Figure 2-3 illustrates one of them. This version of the *screened-subnet*

architecture made a lot of sense back when routers were better at coping with high-bandwidth data streams than multihomed hosts were. However, current best practice is *not* to rely exclusively on routers in one's firewall architecture.

Figure 2-3. "Screened subnet" DM2 architecture



2.2.4 A Strong Screened-Subnet Architecture

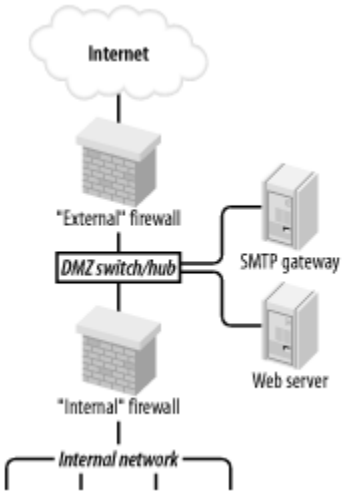
The architecture in Figure 2-4 is therefore better: both the DMZ and the internal networks are protected by full-featured firewalls that are almost certainly more sophisticated than routers.

The weaker screened-subnet design in Figure 2-3 is still used by some sites, but in my opinion, it places too much trust in routers. This is problematic for several reasons.

First, routers are often under the control of a different person than the firewall is, and this person many insist that the router have a weak administrative password, weak access-control lists, or even an attached modem so that the router's vendor can maintain it! Second, routers are considerably more hackable than well-configured computers (for example, by default, they nearly always support remote administration via Telnet, a highly insecure service).

Finally, packet-filtering alone is a crude and incomplete means of regulating network traffic. Simple packet-filtering seldom suffices when the stakes are high, unless performed by a well-configured firewall with additional features and comprehensive logging.

Figure 2-4. Better screened subnet architecture (fully firewalled variant)



This architecture is useful in scenarios in which very high volumes of traffic must be supported, as it addresses a significant drawback of the three-homed firewall architecture in Figure 2-2: if one firewall handles all traffic between three networks, then a large volume of traffic between any two of those networks will negatively impact the third network's ability to reach either. A screened-subnet architecture distributes network load better.

It also lends itself well to

heterogeneous firewall environments. For example, a packet-filtering firewall with high network throughput might be used as the

"external" firewall; an Application

Gateway (proxying) firewall, arguably more secure but probably slower, might then be used as the

"internal" firewall. In this way,

public web servers in the DMZ would be optimally available to the outside world, and private systems on the inside would be most effectively isolated.

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 one (and a sensitive 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 may be different than the one who manages the firewall and DMZ servers, and he may 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 4 and Chapter 5 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, all 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, without exception.

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.

(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 DMZ'ed host from being compromised, but also what the consequences might be if it is, and its being used to sniff other traffic on the DMZ is one possible consequence. 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 100Mbps hub can't really process 1000 Mbps if it has an 800Mbps backplane. Nonetheless, even low-end switches disproportionately outperform comparable hubs.

The other two questions concerning how to distribute DMZ services can usually be determined by nonsecurity-driven factors (cost, expected load, efficiency, etc.), provided

that all DMZ hosts are thoroughly hardened and monitored and that firewall rules (packet-filters, proxy configurations, etc.) governing traffic to and from the DMZ are as restrictive as possible.

```
iptables -I INPUT 1 -i eth0 -s 192.168.0.0/16 -j DROP

iptables -I INPUT 2 -i eth0 -s 10.0.0.0/8 -j DROP

iptables -I INPUT 3 -i eth1 -s ! 192.168.111.0/24 -j DROP

iptables -I INPUT 4 -i eth2 -s ! 10.0.0.0/8 -j DROP

iptables -I FORWARD 1 -i eth0 -s 192.168.0.0/16 -j DROP

iptables -I FORWARD 2 -i eth0 -s 10.0.0.0/8 -j DROP

iptables -I FORWARD 3 -i eth1 -s ! 192.168.111.0/24 -j DROP

iptables -I FORWARD 4 -i eth2 -s ! 10.0.0.0/8 -j DROP
iptables -P INPUT DROP

iptables -P FORWARD DROP

iptables -P OUTPUT DROP
```

Note that most firewalls, including Linux 2.4's iptables, can be configured to reject packets two different ways. The first method, usually called Dropping, is to discard denied packets "silently" i.e., with no notification to the packet's sender. The second method, usually called Rejecting, involves returning a TCP RST (reset) packet if the denied request was via the TCP protocol, or an ICMP "Port Unreachable" message if the request was via UDP.

In most cases, you'll probably prefer to use the Drop method, since this adds significant delay to port scans. Note, however, that it runs contrary to relevant RFCs, which instead specify the TCP-RST and ICMP-Port-Unreachable behavior used in the Reject method.. The Drop method is therefore used only by firewalls, which means that while a port-scanning attacker will experience delay, he'll know precisely why.

Most firewalls that support the Drop method can be configured to log the dropped packet if desired.

2.5.3.4 Strictly limit incoming traffic

The most obvious job of a firewall is to block incoming attacks from external hosts. Therefore, allow incoming connections only to specific (hopefully DMZed) servers. Furthermore, limit those connections to the absolute minimum services/ports necessary e.g., to TCP 80 on your public web server, TCP 25 on your SMTP gateway, etc.

2.5.3.5 Strictly limit all traffic out of the DMZ

A central assumption with DMZs is that its hosts are at significant risk of being compromised. So to contain this risk, you should restrict traffic out of the DMZ to known-necessary services/ports. A DMZed web server, for example, needs to receive HTTP sessions on TCP 80, but does not need to initiate sessions on TCP 80, so it should not be allowed to. If that web server is somehow infected with, say, the Code Red virus, Code Red's attempts to identify and infect other systems from your server will be blocked.

Give particular consideration to traffic from the DMZ to your internal network, and design your environments to minimize the need for such traffic. For example, if a DMZed host needs to make DNS queries, configure it to use the DNS server in the DMZ (if you have one) rather than your internal DNS server. A compromised DMZ server with poorly controlled access to the Internet is a legal liability due to the threat it poses to other networks; one with poorly controlled access into your internal network is an egregious threat to your own network's security.

2.5.3.6 Don't give internal systems unrestricted outbound access

It's common practice to configure firewalls with the philosophy that "inbound transactions are mostly forbidden, but all outbound transactions are permitted." This is usually the result not only of politics ("surely we trust our own users!"), but also of expedience, since a large set of outbound services may legitimately be required, resulting in a long list of firewall rules.

However, many "necessary" outbound services are, on closer examination, actually "desirable" services (e.g., stock-ticker applets, Internet radio, etc.). Furthermore, once the large list of allowed services is in place, it's in place: requests for additional services can be reviewed as needed.

There are two reasons to restrict outbound access from the internal network. First, it helps conserve bandwidth on your Internet connection. Certainly, it's often possible for users to pull audio streams in over TCP 80 to get around firewall restrictions, but the ramifications of doing so will be different than if outbound access is uncontrolled.

Second, as with the DMZ, restricting outbound access from the inside helps mitigate the risk of compromised internal systems being used to attack hosts on other networks, especially where viruses and other hostile code is the culprit.

2.5.3.7 If you have the means, use an application-Gateway firewall

By now, there should be no mistaking my stance on proxying firewalls: if you have the technical wherewithal and can devote sufficient hardware

resources, Application-Gateway firewalls provide superior protection over even stateful packet-filtering firewalls. If you must, use application proxies for some services and packet-filtering only part of the time. (Proxying firewalls nearly always let you use some amount of filtering, if you so choose.)

Linux 2.4's *netfilter* code, while a marked improvement over 2.2's *ipchains*, will be even better if/when Balazs Scheidler adds Linux 2.4 support to his open source Zorp proxy suite. (It's at least partly supported now.)

2.5.3.8 Don't be complacent about host security

My final piece of firewall advice is that you must avoid the trap of ever considering your firewall to be a provider of absolute security. The only absolute protection from network attacks is a cut network cable. Do configure your firewall as carefully and granularly as you possibly can; don't skip hardening your DMZ servers, for example, on the assumption that the firewall provides all the protection they need.

In particular, you should harden publicly accessible servers such as those you might place in a DMZ, as though you have *no firewall at all*. "Security in depth" is extremely important: the more layers of protection you can construct around your important data and systems, the more time-consuming and therefore unattractive a target they'll represent to prospective attackers.

Chapter 3. Hardening Linux

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, disable unnecessary services in 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 firewall one that's at the mercy of the firewall and of 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.

[mick@woofgang]$ man libglade No manual entry for libglade

[mick@woofgang]$ apropos libglade libglade: nothing appropriate

[mick@woofgang]$ rpm -q --whatrequires libglade memprof-0.3.0-8

rep-gtk-gnome-0.13-3

[mick@woofgang mick]# chkconfig --help chkconfig version 1.2.16 - Copyright (C) 1997-2000 Red Hat, Inc.

This may be freely redistributed under the terms of the GNU Public License.

usage: chkconfig --list [name]

chkconfig --add <name> chkconfig --del <name> chkconfig [--level <levels>] <name> <on|off|reset>)

[root@woofgang root]# chkconfig --list anacron 0:off 1:off 2:on 3:on 4:on 5:on 6:off httpd 0:off 1:off 2:off 3:off 4:off 5:off 6:off syslog 0:off 1:off 2:on 3:on 4:on 5:on 6:off crond 0:off 1:off 2:on 3:on 4:on 5:on 6:off network 0:off 1:off 2:on 3:on 4:on 5:on 6:off linuxconf 0:off 1:off 2:on 3:off 4:off 5:off 6:off (etc.)

[root@woofgang root]# chkconfig --level 2 linuxconf off [root@woofgang root]# chkconfig --list linuxconf linuxconf 0:off 1:off 2:off 3:off 4:off 5:off 6:off

/etc/init.d/lpd

#

BEGIN INIT INFO

Provides: lpd

Required-Start: network route syslog named # Required-Stop: network route syslog # Default-Start: 2 3 5

Default-Stop:

Description: print spooling service

END INIT INFO

[root@woofgang root]# mv /etc/rc.d/rc2.d/S30named /etc/rc.d/rc2.d/disabled_S30named [root@woofgang root]# mv /etc/rc.d/rc3.d/S30named /etc/rc.d/rc3.d/disabled_S30named [root@woofgang root]# mv /etc/rc.d/rc5.d/S30named /etc/rc.d/rc5.d/disabled_S30named

wget -nd --passive-ftp ftp://updates.redhat.com/7.0/en/os/i386/rhs-printfilters-1.81-4.rh7.0.i386.rpm

rpm --checksig

./rhs-printfilters-1.81-4.rh7.0.i386.rpm

rpm -Uhv

./rhs-printfilters-1.81-4.rh7.0.i386.rpm

rpm --checksig ./perl-*

rpm -Uhv ./perl-*

rpm -qa

rpm -qa |grep squid

squid23-2.3.STABLE4-75

rpm -qa > packages_07092002.txt

bash-$ su

bash-# export DISPLAY="" bash-# yast2

apt-get update

apt-get -u upgrade

apt-get install blozzo

yard:x:29:29:YARD Database Admin:/usr/lib/YARD:/bin/bash

root@woofgang:~ # find / -user yard -print /usr/lib/YARD

root@woofgang:~ # ls -lu /usr/lib/YARD/ total 20

drwxr-xr-x 2 yard yard 35 Jan 17 2001 .

drwxr-xr-x 59 root root 13878 Dec 13 18:31 ..

-rwsr-xr-x 1 root root 22560 Jan 19 2001 crontab

find / -perm +4000 -user root -type f -print find / -perm +2000 -group root -type f -print

chmod u-s /full/path/to/filename

chmod g-s /full/path/to/filename

Routing/forwarding:

none

Inbound services, public:

FTP, HTTP

Inbound services, private:

SSH

Outbound services

ping, DNS queries

 iptables --list

iptables --list INPUT

iptables --line-numbers --list INPUT

iptables --flush

iptables -I [nsert] chain_name rule_# rule_specification   -D [elete]   -R [eplace]   -A [ppend]

iptables -D OUTPUT 3

iptables -A INPUT -p tcp --dport 80 -j ACCEPT -m state --state NEW

-s <tt>sourceIP</tt>

-d <tt>destinationIP</tt>

-i <tt>ingressInterface</tt>

-o <tt>egressInterface</tt>

-p <tt>tcp</tt> | <tt>udp</tt> | <tt>icmp</tt> | <tt>all</tt>

--dport <tt>destinationPort</tt>

--sport <tt>sourcePort</tt>

--tcp-flags <tt>mask match</tt>

--icmp-type <tt>type</tt>

-m state --state <tt>statespec</tt>

-j <tt>accept</tt> | <tt>drop</tt> | <tt>log</tt> | <tt>reject</tt> |

<tt>[chain_name]</tt>

modprobe ip_tables

modprobe ip_conntrack_ftp

Flush old rules, old custom tables $IPTABLES --flush

$IPTABLES --delete-chain

Set default-deny policies for all three default chains

$IPTABLES -P INPUT DROP

$IPTABLES -P FORWARD DROP

$IPTABLES -P OUTPUT DROP

Give free reign to loopback interfaces $IPTABLES -A INPUT -i lo -j ACCEPT

$IPTABLES -A OUTPUT -o lo -j ACCEPT

Do some rudimentary anti-IP-spoofing drops $IPTABLES -A INPUT -s 255.0.0.0/8 -j LOG --log-prefix "Spoofed source IP!"

$IPTABLES -A INPUT -s 255.0.0.0/8 -j DROP

$IPTABLES -A INPUT -s 0.0.0.0/8 -j LOG --log-prefix "Spoofed source IP!"

$IPTABLES -A INPUT -s 0.0.0.0/8 -j DROP

$IPTABLES -A INPUT -s 127.0.0.0/8 -j LOG --log-prefix "Spoofed source IP!"

$IPTABLES -A INPUT -s 127.0.0.0/8 -j DROP

$IPTABLES -A INPUT -s 192.168.0.0/16 -j LOG --log-prefix "Spoofed source IP!"

$IPTABLES -A INPUT -s 192.168.0.0/16 -j DROP

$IPTABLES -A INPUT -s 172.16.0.0/12 -j LOG --log-prefix " Spoofed source IP!"

$IPTABLES -A INPUT -s 172.16.0.0/12 -j DROP

$IPTABLES -A INPUT -s 10.0.0.0/8 -j LOG --log-prefix " Spoofed source IP!"

$IPTABLES -A INPUT -s 10.0.0.0/8 -j DROP

```
$IPTABLES -A INPUT -s 208.13.201.2 -j LOG --log-prefix "Spoofed Woofgang!"

$IPTABLES -A INPUT -s 208.13.201.2 -j DROP

$IPTABLES -A INPUT -s 208.13.201.2 -j DROP

# Tell netfilter that all TCP sessions do indeed begin with SYN

$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j LOG --log-prefix "Stealth scan attempt?"

$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j DROP

# Accept inbound packets that are part of previously-OK'ed sessions
$IPTABLES -A INPUT -j ACCEPT -m state --state ESTABLISHED,RELATED

# Accept inbound packets which initiate SSH sessions $IPTABLES -A INPUT -p tcp -j ACCEPT --dport 22 -m state --state NEW

# Accept inbound packets which initiate FTP sessions $IPTABLES -A INPUT -p tcp -j ACCEPT --dport 21 -m state --state NEW

# Accept inbound packets which initiate HTTP sessions $IPTABLES -A INPUT -p tcp -j ACCEPT --dport 80 -m state --state NEW

# Log anything not accepted above

$IPTABLES -A INPUT -j LOG --log-prefix "Dropped by default:"

$IPTABLES -A INPUT -p tcp -j ACCEPT -s 208.13.201.1 --dport 22 -m state --state NEW

# If it's part of an approved connection, let it out $IPTABLES -I OUTPUT 1 -m state --state RELATED,ESTABLISHED -j ACCEPT

# Allow outbound ping (comment-out when not needed!) $IPTABLES -A OUTPUT -p icmp -j ACCEPT --icmp-type echo-request # Allow outbound
```

DNS queries, e.g. to resolve IPs in logs $IPTABLES -A OUTPUT -p udp --dport 53 -m state --state NEW -j ACCEPT

Log anything not accepted above - if nothing else, for t-shooting
$IPTABLES -A OUTPUT -j LOG --log-prefix "Dropped by default:"

#! /bin/sh

init.d/localfw

#

System startup script for Woofgang's local packet filters

#

last modified 30 Dec 2001 mdb

```
#

IPTABLES=/usr/sbin/iptables

test -x $IPTABLES || exit 5

case "$1" in

start)

echo -n "Loading Woofgang's Packet Filters"

# SETUP -- stuff necessary for any host # Load kernel modules first

modprobe ip_tables

modprobe ip_conntrack_ftp

# Flush old rules, old custom tables $IPTABLES --flush

$IPTABLES --delete-chain

# Set default-deny policies for all three default chains
```

$IPTABLES -P INPUT DROP

$IPTABLES -P FORWARD DROP

$IPTABLES -P OUTPUT DROP

Give free reign to loopback interfaces $IPTABLES -A INPUT -i lo -j ACCEPT

$IPTABLES -A OUTPUT -o lo -j ACCEPT

Do some rudimentary anti-IP-spoofing drops $IPTABLES -A INPUT -s 255.0.0.0/8 -j LOG --log-prefix "Spoofed source IP!"

$IPTABLES -A INPUT -s 255.0.0.0/8 -j DROP

$IPTABLES -A INPUT -s 0.0.0.0/8 -j LOG --log-prefix "Spoofed source IP!"

$IPTABLES -A INPUT -s 0.0.0.0/8 -j DROP

$IPTABLES -A INPUT -s 127.0.0.0/8 -j LOG --log-prefix "Spoofed source IP!"

$IPTABLES -A INPUT -s 127.0.0.0/8 -j DROP

$IPTABLES -A INPUT -s 192.168.0.0/16 -j LOG --log-prefix "Spoofed source IP!"

$IPTABLES -A INPUT -s 192.168.0.0/16 -j DROP

$IPTABLES -A INPUT -s 172.16.0.0/12 -j LOG --log-prefix " Spoofed source IP!"

$IPTABLES -A INPUT -s 172.16.0.0/12 -j DROP

$IPTABLES -A INPUT -s 10.0.0.0/8 -j LOG --log-prefix " Spoofed source IP!"

$IPTABLES -A INPUT -s 10.0.0.0/8 -j DROP

```
$IPTABLES -A INPUT -s 208.13.201.2 -j LOG --log-prefix "Spoofed Woofgang!"

$IPTABLES -A INPUT -s 208.13.201.2 -j DROP

# Tell netfilter that all TCP sessions do indeed begin with SYN

$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j LOG --log-prefix "Stealth scan attempt?"

$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j DROP

# Finally, the meat of our packet-filtering policy:
```

INBOUND POLICY

Accept inbound packets that are part of previously-OK'ed sessions
$IPTABLES -A INPUT -j ACCEPT -m state --state
ESTABLISHED,RELATED

Accept inbound packets which initiate SSH sessions $IPTABLES -A
INPUT -p tcp -j ACCEPT --dport 22 -m state --state NEW

Accept inbound packets which initiate FTP sessions $IPTABLES -A
INPUT -p tcp -j ACCEPT --dport 21 -m state --state NEW

Accept inbound packets which initiate HTTP sessions $IPTABLES -A
INPUT -p tcp -j ACCEPT --dport 80 -m state --state NEW

Log anything not accepted above

$IPTABLES -A INPUT -j LOG --log-prefix "Dropped by default:"

OUTBOUND POLICY

If it's part of an approved connection, let it out $IPTABLES -I OUTPUT 1 -m state --state RELATED,ESTABLISHED -j ACCEPT

Allow outbound ping (comment-out when not needed!) $IPTABLES -A OUTPUT -p icmp -j ACCEPT --icmp-type echo-request # Allow outbound DNS queries, e.g. to resolve IPs in logs $IPTABLES -A OUTPUT -p udp --dport 53 -m state --state NEW -j ACCEPT

Log anything not accepted above - if nothing else, for t-shooting $IPTABLES -A OUTPUT -j LOG --log-prefix "Dropped by default:"

;;

wide_open)

echo -n "DANGER!! Unloading Woofgang's Packet Filters!!"

Unload filters and reset default policies to ACCEPT.

FOR EMERGENCY USE ONLY -- else use `stop'!!

$IPTABLES --flush

$IPTABLES -P INPUT ACCEPT

$IPTABLES -P FORWARD ACCEPT

$IPTABLES -P OUTPUT ACCEPT

;;

stop)

echo -n "Portcullis rope CUT..."

Unload all fw rules, leaving default-drop policies $IPTABLES --flush

;;

status)

echo "Querying iptables status (via iptables --list)..."

$IPTABLES --line-numbers -v --list

;;

*)

echo "Usage: $0 {start|stop|wide_open|status}"

exit 1

;;

esac

root@woofgang: # cd nmap-2.54BETA30 root@woofgang: # ./configure root@woofgang: # make root@woofgang: # make install

nmap <tt>[</tt>-s <tt>(scan-type)]</tt> <tt>[</tt>-p<tt> (port-range)]|-</tt>F<tt> (options) target</tt>

[root@mcgruff]# nmap -sT -F -P0 -O woofgang.dogpeople.org Starting nmap V. 2.54BETA30 (www.insecure.org/nmap/) Warning: OS detection will be MUCH less reliable because we did not find at least 1 open and 1 closed TCP port

Interesting ports on woofgang.dogpeople.org (208.13.201.2): (The 1091 ports scanned but not shown below are in state: filtered) Port State Service

21/tcp open ftp

22/tcp open ssh

80/tcp open http

Remote operating system guess: Linux Kernel 2.4.0 - 2.4.9 (X86) Uptime 1.163 days (since Mon Dec 31 12:24:18 2001) Nmap run completed 1 IP address (1 host up) scanned in 127 seconds

[root@mcgruff]# nmap -sTUR -F -P0 -O woofgang.dogpeople.org Starting nmap V. 2.54BETA30 (www.insecure.org/nmap/) Warning: OS detection will be MUCH less reliable because we did not find at least 1 open and 1 closed TCP port

Interesting ports on woofgang.dogpeople.org (208.13.201.2): (The 2070 ports scanned but not shown below are in state: filtered) Port State Service (RPC)

21/tcp open ftp

22/tcp open ssh

80/tcp open http

Remote operating system guess: Linux Kernel 2.4.0 - 2.4.9 (X86) Uptime 1.180 days (since Mon Dec 31 12:24:18 2001) Nmap run completed 1 IP address (1 host up) scanned in 718 seconds

sh ./nessus-installer.sh

Nessus installation : Finished ----------

Congratulations ! Nessus is now installed on this host . Create a nessusd certificate using $prefix/sbin/nessus-mkcert . Add a nessusd user use $prefix/sbin/nessus-adduser . Start the Nessus daemon (nessusd) use $prefix/sbin/nessusd -D

. Start the Nessus client (nessus) use $prefix/bin/nessus . To uninstall Nessus, use $prefix/sbin/uninstall-nessus . A step by step demo of Nessus is available at : http://www.nessus.org/demo/

Press ENTER to quit

nessusd -P bobo,scuz00DL

The password ("scuz00DL" in the previous example) is called a "one-time" password because by default, after *bobo* first logs in and gives this password, his public key will be registered with the Nessus server. Subsequent logins will not require him to enter this password again (they'll be authenticated transparently using an SSL-like challenge-response transaction).

The second and more powerful way to create new user accounts on the server is to use the `nessus-adduser` command. This script actually does most of its magic by invoking *nessusd*, but presents you with a convenient interface for managing users with more granularity than a simple `nessusd -P`. You are prompted not only for a username and one-time password, but also IP addresses from which the user may connect and rules that restrict which hosts the user may scan with Nessus.

I leave it to you to read the *nessus-adduser* manpage if you're interested in this level of user-account management. Our remaining space here is better spent discussing how to build, run, and interpret Nessus scans.

Before we leave the topic of authentication, though, I should mention the other kind of authentication Nessus uses, this one local to each client session. When you start *nessus* for the first time (the client, not the daemon), you are prompted for a passphrase.

This passphrase protects a private key that's stored in the home directory of the Unix account you're logged into when you start *nessus*, and you'll be prompted for it whenever you start *nessus*. Then, when you connect to a Nessus server, your private key will be used in the transparent challenge-response transaction described earlier that actually authenticates you to the remote *nessusd* process.

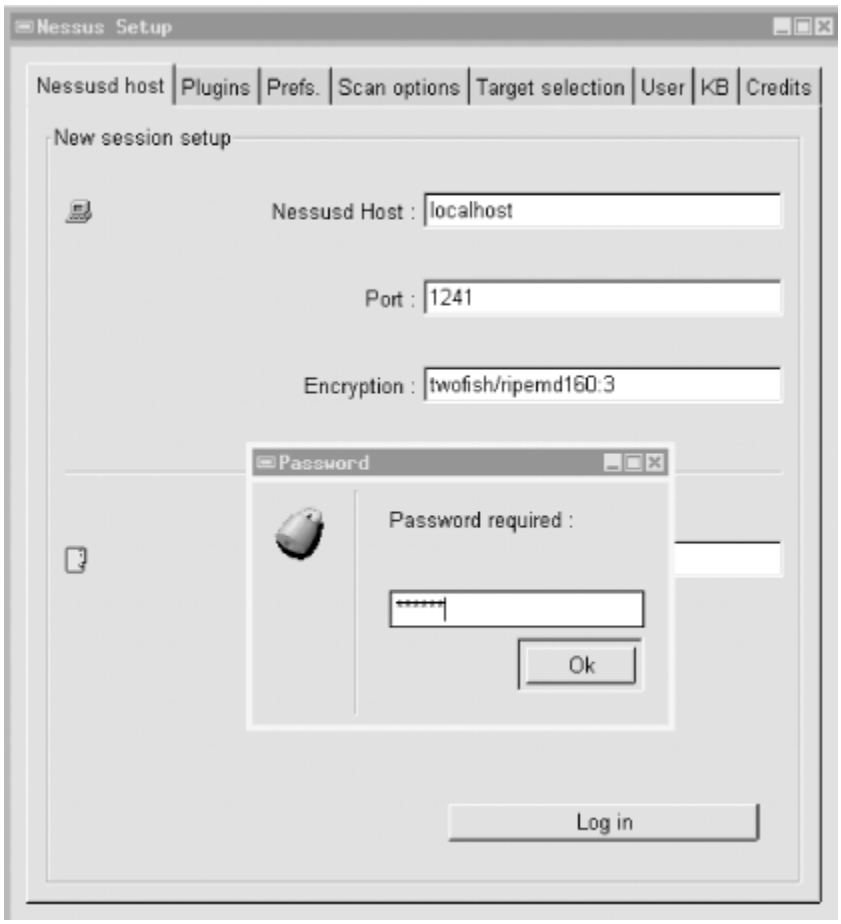
If all this seems confusing, don't worry: just remember that the password you're prompted for each time you start *nessus* has nothing to do with the password you use the first time you connect to a Nessus server.

3.1.9.13 Performing security scans with Nessus

And now the real fun begins! After Nessus has been installed and at least one user account set up, you're ready to scan. First, start a client session, and enter your client-private-key's passphrase when prompted (by the way, you can change or delete this passphrase with the command `nessus -C`, which will prompt you for your current passphrase and what you'd like to change it to).

Next, enter the name or IP address of the "Nessusd host" (server) to which you wish to connect, the port on which it's listening, your preferred encryption method (the default should be fine), and your Nessus login/username (Figure 3-9). The defaults for Port and Encryption are usually fine.

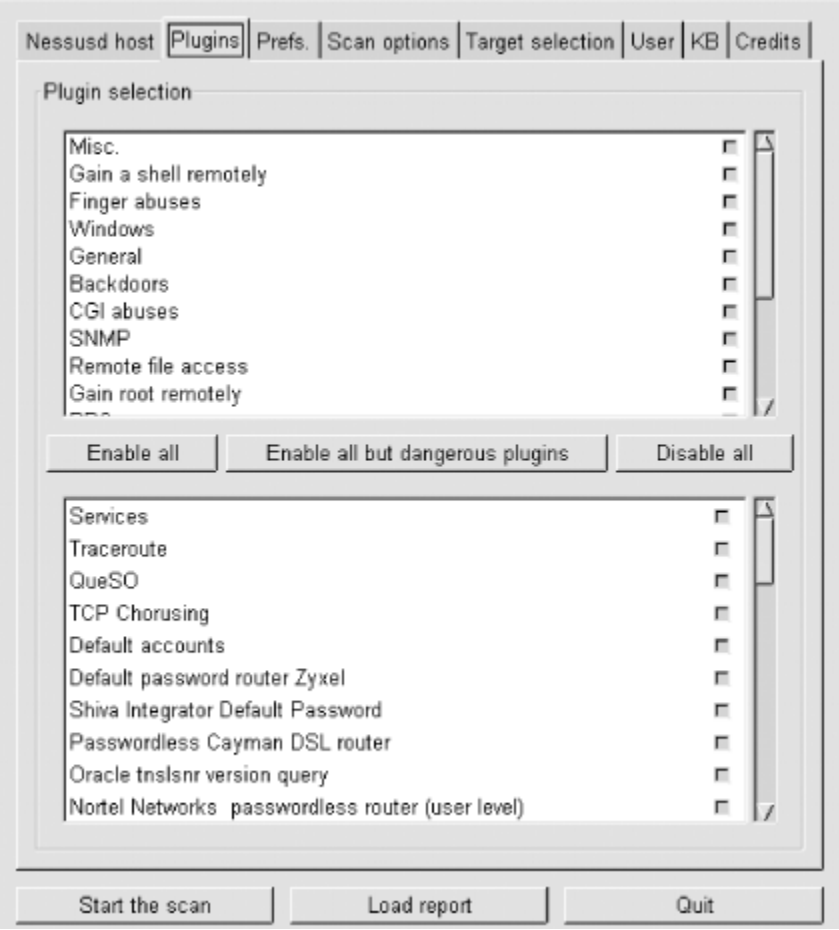
Figure 3-9. User "Bobo's" first login to a Nessus server



When you're ready to connect, click the "Log in" button. If this is the first time you've connected to the server using the specified login, you'll be prompted for your "one-time" password (next time, you won't be). With that, you should be connected and ready to build a scan!

If you click the "Plugins" tab, you're presented with a list of all vulnerability tests available on the Nessus server, grouped by "family" (Figure 3-10). Click on a family's name (these are listed in the upper half of the window) to see a list of that family's plug-ins below. Click on a family's checkbox to enable or disable all its plug-ins.

Figure 3-10. Plugins screen



If you don't know what a given plug-in does, click on its name: an information window will pop up. If you "hover" the mouse pointer over a plug-in's name, a summary caption will pop up that states very briefly what the plug-in does. Plug-ins with yellow triangles next to their checkboxes are dangerous: the particular tests they perform have the potential to interrupt or even crash services on the target (victim) host.

By the way, don't be too worried about selecting all or a large number of plug-ins: Nessus is intelligent enough to skip, for example, Windows tests on non-Windows hosts. In general, Nessus is efficient in deciding which tests to run and in which circumstances.

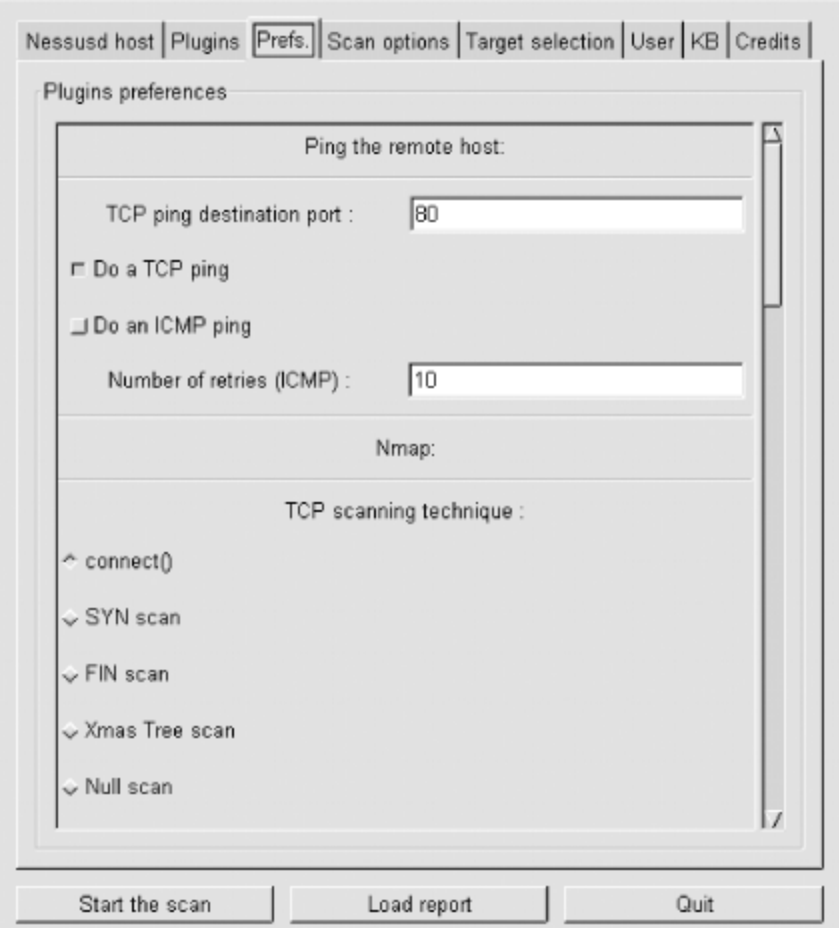
The next screen to configure is "Prefs" (Figure 3-11). Contrary to what you might think, this screen contains not general, but plug-in-specific preferences, some of which are mandatory for their corresponding plug-in

to work properly. Be sure to scroll down the entire list and provide as much information as you can.

Take particular care with the Ping section (at the very top): more often than not, selecting either ping method (TCP or ICMP) can cause Nessus to decide mistakenly that hosts are down when in fact they are up. Nessus will not perform any tests on a host that doesn't reply to pings; so when in doubt, don't ping.

In the "Nmap" section, Linux users should select only "tcp connect( )" and should deselect all other scan types due to a bug in *libpcap* that affects the way Nessus performs port scans.

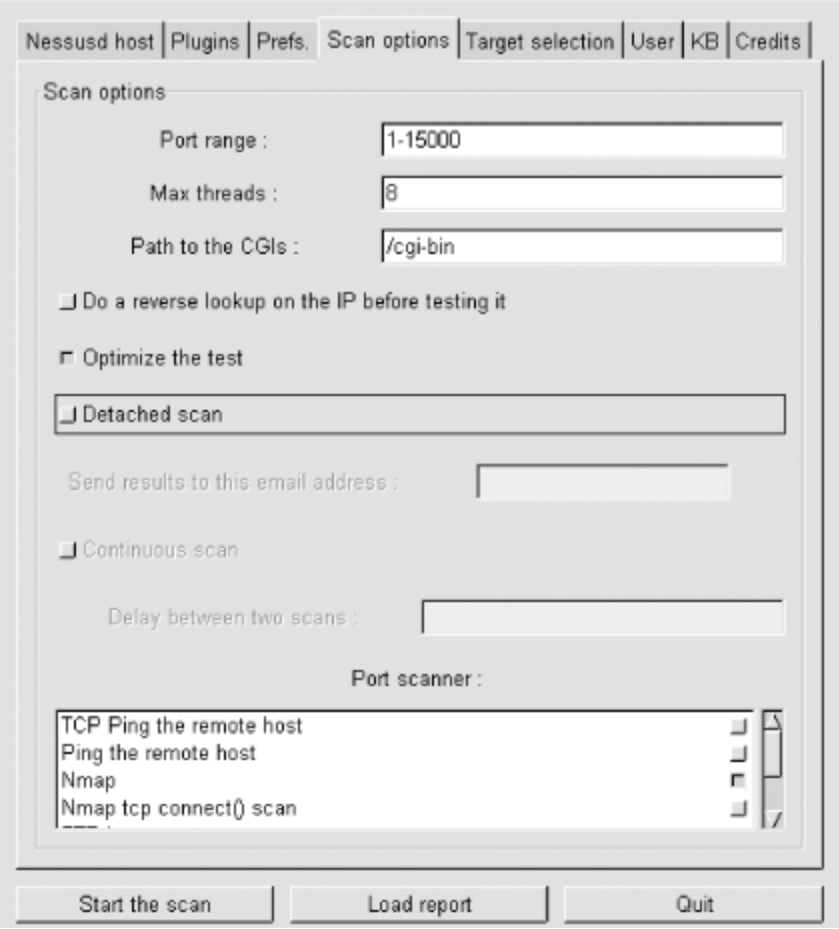
Figure 3-11. Plugins Prefs screen



After Prefs comes "Scan Options" (Figure 3-12). Note that the Nessus installation in Figure 3-12 was compiled with the "Save Session" feature, as evidenced by the "Detached Scan" and "Continuous Scan" options, which would otherwise be absent.

As in the Prefs screen, you should deselect everything under "Port scanner" except "Nmap tcp connect( ) scan" due to the bug mentioned earlier.

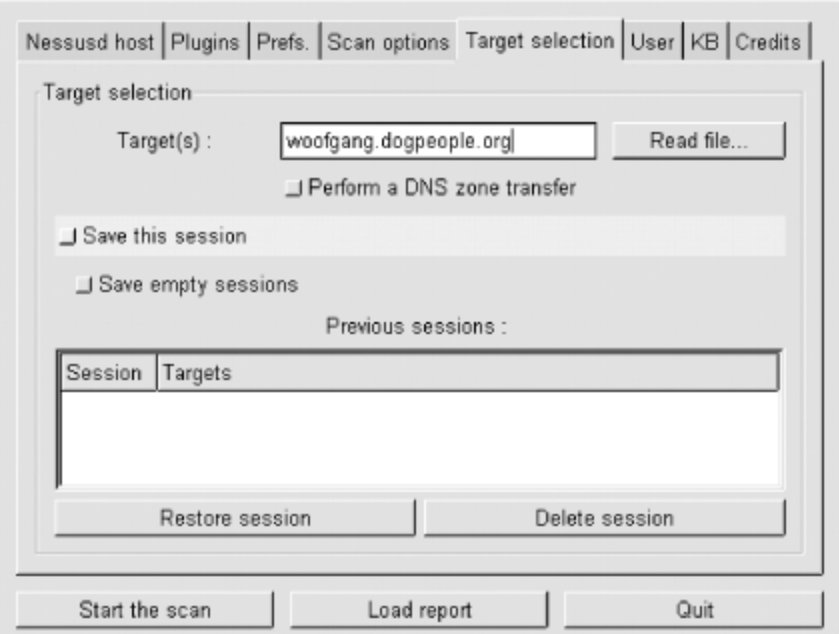
Figure 3-12. Scan options screen



The "Optimize the test" option tells Nessus to avoid all apparently inapplicable tests, but this can at least theoretically result in "false negatives." Balance the risk of false negatives against the advantage of a complete scan as quickly as possible. Speaking of speed, if you care about it, you probably want to avoid using the "Do a reverse (DNS) lookup..." feature, which attempts to determine the hostnames for all scanned IP addresses.

Now we specify our targets. We specify these in the "Target(s):" field of the "Target Selection" screen (Figure 3-13). This can contain hostnames, IP addresses, network addresses in the format x.x.x.x/y (where x.x.x.x is the network number and y is the number of bits in the subnet mask e.g., 192.168.1.0/24), in a comma-separated list.

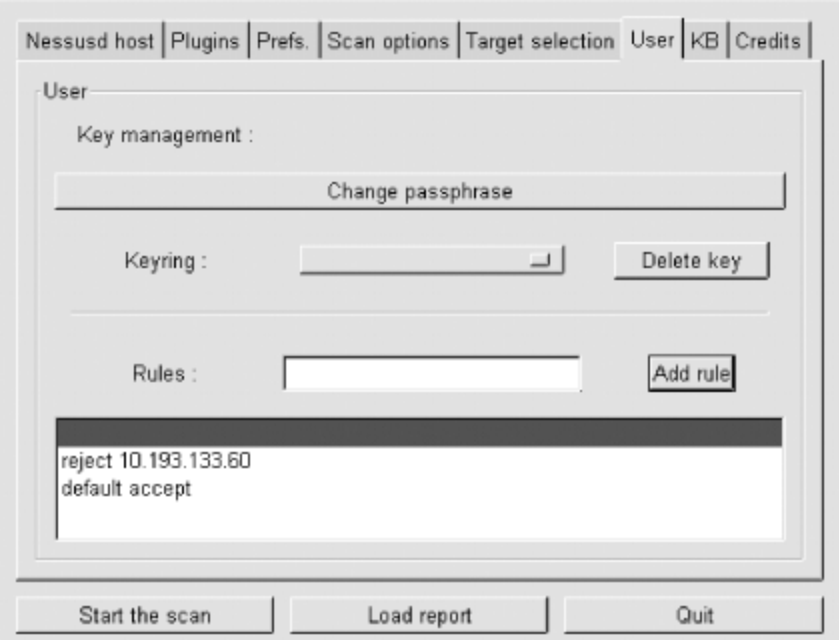
Figure 3-13. Target selection screen



The "Perform a DNS zone transfer option" instructs Nessus to obtain all available DNS information on any domain names or subdomain names referred to in the "Target(s):" box. Note that most Internet DNS servers are configured to deny zone-transfer requests to TCP port 53 by unknown hosts. The other options in this screen have to do with the experimental Save Session feature I mentioned earlier see http://www.nessus.org/documentation.html for more information on what the experimental features do and how to use them.

Finally, one last screen before we begin our scan (we're skipping "KB," which applies only if you've compiled and wish to use the Knowledge Base features): "User" (Figure 3-14). In this screen, we can change our client passphrase (this has the same effect as `nessus -c`), and we can fine-tune the targets we specified in the "Target selection" screen.

Figure 3-14. User screen



The specifications you type in this text box are called "rules," and they follow a simple format: `accept` *`address`*, `deny` *`address`*, or `default` *`[`*`accept` | `reject`*`]`*. The rules listed in Figure 3-14 mean "don't scan 10.193.133.60, but scan everything else specified in the Target screen."

Finally, the payoff for all our careful scan setup: click the "Start the scan" button at the bottom of the screen. The scan's length will vary, depending mainly on how many hosts you're scanning and how many tests you've enabled. The end result? A report such as that shown earlier in Figure 3-8.

From the Report window, you can save the report to a file, besides viewing the report and drilling down into its various details. Supported report file formats include HTML, ASCII, $LaTeX$, and of course a proprietary Nessus Report format, "NSR" (which you should use for reports from which you wish to view again within Nessus).

Read this report carefully, be sure to expand all "+" boxes, and fix the things Nessus turns up. Nessus can find problems and can even suggest solutions, but it won't fix things for you. Also, Nessus won't necessarily find everything wrong with your system.

Returning to our woofgang example (see Figure 3-8), Nessus has determined that woofgang is running a vulnerable version of OpenSSH! Even after all the things we've done so far to harden this host, there's still a major vulnerability to take care of. We'll have to upgrade woofgang's OpenSSH packages before putting this system into production.

Interestingly, I had run *yast2*'s "Online Update" utility on the host I used in these examples, but evidently not recently enough to catch the new OpenSSH packages. This is an excellent illustration of how judicious use of security scanning can augment your other security practices.

3.1.10 Understanding and Using Available Security Features

This corollary to the principle of least privilege is probably one of the most obvious but least observed. Since many applications' security features aren't enabled by default (running as an unprivileged user, running in a chroot jail, etc.), those features tend not to be enabled, period. Call it laziness or call it a logical aversion to fixing what doesn't seem to be broken, but many people tinker with an application only enough to get it working, indefinitely postponing that crucial next step of securing it too.

This is especially easy to justify with a server that's supposedly protected by a firewall and maybe even by local packet filters: it's covered, right? Maybe, but maybe not. Firewalls and packet filters protect against certain types of network attacks (hopefully, most of them), but they can't protect you against vulnerabilities in the applications that firewalls/filters still allow.

As we saw with woofgang, the server we hardened with iptables and then scanned with nmap and Nessus, it only takes one vulnerable application (OpenSSH, in this case) to endanger a system. It's therefore imperative that a variety of security strategies and tools are employed. This is called "Defense in Depth," and it's one of the most important concepts in information security.

3.1.11 Documenting Bastion Hosts' Configurations

Finally, document the steps you take in configuring and hardening your bastion hosts. Maintaining external documentation of this kind serves three important functions. First, it saves time when building subsequent, similar systems. Second, it helps you to rebuild the system quickly in the event of a hard-drive crash, system compromise, or any other event requiring a "bare-metal recovery."

Third, good documentation can also be used to disseminate important information beyond one key person's head. (Even if you work alone, it can keep key information from being lost altogether should it get misplaced somewhere in that head!)

InteractiveBastille -c

Now read Bastille's explanations (Figure 3-15), answer its questions, and when you reach the end, reboot to implement Bastille's changes. That's really all there is to running Bastille.

Figure 3-15. InteractiveBastille session

3.2.4 Some Notes on InteractiveBastille

InteractiveBastille explains itself extremely well during the course of a Bastille session. This verbosity notwithstanding, the following general observations on certain sections may prove useful to the beginner:

Module 1: *Firewall.pm*

> Bastille has one of the better facilities I've seen for automatically generating packet filters. By answering the questions in this section, you'll gain a new script in */etc/init.d*, called `bastillefirewall`, which can be used to initialize ipchains or iptables, whichever your kernel supports. Note that you must manually review and activate this script (i.e., double check the script with your text editor of choice, and then create symbolic links to it with chkconfig).

Module 2: *FilePermissions.pm*

This module restricts access to certain utilities and files, mainly by disabling their SUID status. The SUID problem is discussed earlier in this chapter.

Module 3: *AccountSecurity.pm*

This module allows you to create a new administration account and generally tighten up the security of user-account management via password aging, tty restrictions, etc. These are all excellent steps to take; I recommend using them all.

Module 4: *BootSecurity.pm*

If it's possible for unknown or untrusted persons to sit in front of your system, reboot or power-cycle it, and interrupt the boot process, these settings can make it harder for them to compromise the system.

Module 5: *SecureInetd.pm*

inetd and *xinetd* can pose numerous security problems. This Bastille module configures access controls for inetd or xinetd services, depending on which is installed on your system. If you're using inetd, Bastille will configure tcpwrappers; otherwise, it will use xinetd's more granular native-access controls.

Module 6: *DisableUserTools.pm*

The "User Tools" in question here are the system's programming utilities: compilers, linkers, etc. Disabling these is a good idea if this is a bastion host. Note that as in most other cases, when Bastille says "disable," it actually means "restrict to root-access only."

Module 7: *ConfigureMiscPAM.pm*

Several useful restrictions on user accounts are set here. Note, however, that the file-size restriction of 40MB that Bastille sets may

cause strange behavior on your system. Be prepared to edit */etc/security/limits.conf* later if this happens to you.

Module 8: *Logging.pm*

Too little logging is enabled by default on most systems. This module increases the overall amount of logging and allows you to send log data to a remote host. Process accounting (i.e., tracking all processes) can also be enabled here, but is overkill for most systems.

Module 9: *MiscellaneousDaemons.pm*

In this section, you can disable a number of services that tend to be enabled by default, despite being unnecessary for most users.

Module 10: *Sendmail.pm*

This Bastille module performs some rudimentary tweaks to *sendmail*: notably, disabling its startup script if the system is not an SMTP gateway and disabling dangerous SMTP commands such as EXPN and VRFY if it is.

Module 11: *Apache.pm*

This module addresses several aspects of Apache (web server) security, including interface/IP bindings, server-side includes, and CGI.

Module 12: *Printing.pm*

It's common for *lpd*, the Line Printer Daemon, to be active even if no printers have been configured. That may not sound too frightening, but there have been important security exposures in lpd recently and in the past. This module disables printing if it isn't needed.

Module 13: *TMPDIR.pm*

Since */tmp* is world-readable and -writable, there have been security problems associated with its use. This module sets up `TMPDIR` and `TMP`

environment variables for your user accounts; these variables define alternate temporary directories that are less likely to be abused than */tmp*.

3.2.5 Bastille's Logs

So, after InteractiveBastille is finished and the system is rebooted, what then? How do we know what happened? Thanks to Bastille's excellent logging, it's easy to determine exactly which changes were successful and, equally important, which failed.

It's probably a good idea to review these logs regardless of whether you think something's gone wrong; meaningful logging is one of Bastille's better features. Whether a beginner or a security guru, you should know not only what changes Bastille makes, but how it makes them.

Bastille writes its logs into */root/Bastille/log/*. Two logs are created by *BackEnd.pl*: action-log and error-log. action-log provides a comprehensive and detailed accounting of all of Bastille's activities. Errors and other unexpected events are logged to error-log.

3.2.6 Hooray! I'm Completely Secure Now! Or Am I?

Okay, we've carefully read and answered the questions in InteractiveBastille, we've rebooted, and we've reviewed Bastille's work by going over its logs. Are we there yet?

Well, our system is clearly much more secure than it was before we started. But as Bruce Schneier is fond of saying, security is a process, not a product: while much of the work necessary to bastionize a system only needs to be performed once, many important security tasks, such as applying security patches and monitoring logs, must be performed on an ongoing basis.

Also, remember our quest for "Defense in Depth": having done as much as possible to harden our base operating system, we still need to leverage any and all security features supported by our important applications and services. That's what the rest of this book is about.

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 Clear-Text 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 packet

sniffers (which can be used to eavesdrop the passwords and data that these applications transmit unencrypted) were rare and people who knew how to use them were even rarer.

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 *clear-text username/password authentication is*

obsolete

. (So is clear-text 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.

tar -xzvf openssh-2.9p1.tar.gz

cd openssh-2.9p1

./configure --sysconfdir=/etc/ssh

make

make install

cat ./rc.config.ssd >> /etc/rc.config

ssh remote.host.net

ssh mbauer@kong-fu.mutantmonkeys.org

[mick@kolach stash]# sftp crueller

Connecting to crueller...

mick@crueller's password:

sftp> dir

drwxr-x--- 15 mick users 1024 May 17 19:35 .

drwxr-xr-x 17 root users 1024 May 11 20:02 ..

-rw-r--r-- 1 mick users 1126 Aug 23 1995 baklava_recipe.txt

-rw-r--r-- 1 mick users 124035 Jun 10 2000 donut_cntrfold.jpg

-rw-r--r-- 1 mick users 266 Mar 26 17:40 blintzes_faq

-rw-r--r-- 1 mick users 215 Oct 22 2000 exercise_regimen.txt

sftp> get blintzes_faq

Fetching /home/mick/blintzes_faq to blintzes_faq

sftp> put bakery_maps.pdf

Uploading bakery_maps.pdf to /home/mick

sftp> quit

[mick@kolach stash]#

scp <tt>[options] sourcefilestring destfilestring</tt>

<tt>username</tt>@<tt>remote.host.name:path</tt>/<tt>filename</tt>

crueller: > scp ./recipe kolach:~

mick@kolach's password: *******

recipe 100% |*****************************>| 13226 00:00

crueller: >

crueller: > scp ./recipe mbauer@kolach:/data/recipies/pastries/

crueller: > scp mbauer@kolach:/etc/oven.conf .

Compression yes

Compression no

ssh -o Compression=yes <tt>remote.host.net</tt>

<tt>parameter-name</tt> <tt> parameter-value1(,parameter-value2, etc.)</tt>

In other words, a parameter and its first value are separated by whitespace and additional values are separated by commas. Some parameters are Boolean and can have a value of either "yes" or "no." Others can have a list

of values separated by commas. Most parameters are self-explanatory, and all are explained in the *ssh(1)* manpage. Table 4-1 lists a few of the most useful and important ones (italicized text indicates possible values).

Table 4-1. Important ssh_config parameters

Parameter	Possible values	Description
`CheckHostIP`	*`Yes`*, *`No`* (Default=*`Yes`*)	Whether to notice unexpected source IPs for known host keys. Warns user each time discrepancies are found.
`Cipher`	*`3des`*, *`blowfish`*(Default=*`3des`*)	Which block cipher should be used for encrypting ssh v.1 sessions.
`Ciphers`	*`3des-cbc`*, *`blowfish-cbc`*, *`arcfour`*, *`cast128-cbc`*	Order in which to try block ciphers that can be used for encrypting ssh v.2 sessions.
`Compression`	*`Yes`*, *`No`* (Default=*`No`*)	Whether to use `gzip` to compress encrypted session data. Useful over

		limited-bandwidth connections, but otherwise only adds delay.
`ForwardX11`	*`Yes`*, *`No`* (Default=*`No`*)	Whether to redirect X connections over the encrypted tunnel and to set DISPLAY variable accordingly. Very handy feature!
`PasswordAuthentication`	*`Yes`*, *`No`* (Default=*`Yes`*)	Whether to attempt (encrypted) Unix password authentication in addition to or instead of trying RSA/DSA.

There are many other options in addition to these; some of them are covered in Section 4.3 (later in this chapter). Refer to the *ssh(1)* manpage for a complete list.

4.2.6 Configuring and Running sshd, the Secure Shell Daemon

Editing *ssh_config* is sufficient if the hosts you connect to are administered by other people. But we haven't yet talked about configuring your own host to accept ssh connections.

Like the ssh client, sshd's default behavior is configured in a single file, *sshd_config*, that resides either in */etc* or wherever else you specified in SSH's configuration directory. As with the ssh client, settings in its configuration file are overridden by command-line arguments. Unlike ssh,

however, there are no configuration files for the daemon in individual users' home directories; ordinary users can't dictate how the daemon behaves.

Table 4-2 lists just a few of the things that can be set in *sshd_config*.

Table 4-2. Some sshd_config parameters

Parameter	Possible values	Description
`Port`	*`1-65535`* (Default=*`22`*)	TCP port on which the daemon should listen. Being able to change this is handy when using Port Address Translation to allow several hosts to hide behind the same IP address.
`PermitRootLogin`	*`Yes`*, *`No`*	Whether to accept root logins. This is best set to *`No`*; administrators should connect the server with unprivileged accounts, and then *`su`* to root.
`PasswordAuthentication`	*`Yes`*, *`No`*	Whether to allow (encrypted) username/password authentication or to insist on DSA- or RSA-key-based authentication.
`PermitEmptyPasswords`	*`Yes`*, *`No`*(Default=*`no`*)	Whether to allow accounts to log in whose system password

		is empty. Does not apply if `PasswordAuthentication=`*`no`*; also, does not apply to passphrase of DSA or RSA keys (i.e., null passwords on keys is okay)
`X11Forwarding`	*`Yes`*, *`No`*(Default=*no*)	Whether to allow clients to run X Windows applications over the SSH tunnel.[1]

[1] There really is nothing to be gained by leaving `X11Forwarding` set to *`No`* in *sshd_config*, since a determined user can simply use generic TCP forwarding to forward X11. The only reason it's even in the chart is because people usually expect X11 forwarding to be allowed, and you'll certainly get calls from your users if you have it turned off just because you forgot to change the default value of *No*.

There are many other parameters that can be set in *sshd_config*, but understanding the previous concepts is enough to get started (assuming your immediate need is to replace Telnet and ftp). See the *sshd(8)* manpage for a complete reference for these parameters.

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 Free S/WAN to connect to an "access point" in your DMZ or internal network e.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.

mbauer@homebox:~/.ssh > ssh-keygen -d -b 1024 -C mbauer@homebox.pinheads.com

Generating DSA parameter and key.

Enter file in which to save the key (/home/mbauer/.ssh/id_dsa): Enter passphrase (empty for no passphrase): ********** Enter same passphrase again: ********** Your identification has been saved in /home/mbauer/.ssh/id_dsa.

Your public key has been saved in /home/mbauer/.ssh/id_dsa.pub.

The key fingerprint is:

95:a9:6f:20:f0:e8:43:36:f2:86:d0:1b:47:e4:00:6e mbauer@homebox.pinheads.com

scp ./id_dsa.pub remotehostname:/your/homedir

cat id_dsa.pub >> .ssh/authorized_keys2

bauer@homebox:~/ > ssh -2 zippy.pinheads.com

Enter passphrase for DSA key '/home/mbauer/.ssh/id_dsa': Last login: Wed Oct 4 10:14:34 2000 from homebox.pinheads.com Have a lot of fun...

mbauer@zippy:~ > _

ssh-keygen -b 1024 -C mbauer@homebox.pinheads.com

mbauer@pinheads:~ > ssh-agent

SSH_AUTH_SOCK=/tmp/ssh-riGg3886/agent.3886; export SSH_AUTH_SOCK; SSH_AGENT_PID=3887; export SSH_AGENT_PID;

echo Agent pid 3887;

mbauer@pinheads:~ > _

mbauer@pinheads:~ > ssh-agent > temp

mbauer@pinheads:~ > chmod u+x temp mbauer@pinheads:~ > . ./temp

mbauer@pinheads:~ > ssh-add /home/mbauer/.ssh/id_dsa

scp -i /etc/script_dsa_id /var/log/messages.* scriptboy@archive.g33kz.org

chmod go-rwx private_key_filename

mbauer@homebox > ssh mbauer@zippy.pinheads.com cat /var/log/messages | more Oct 5 16:00:01 zippy newsyslog[64]: logfile turned over Oct 5 16:00:02 zippy syslogd: restart

Oct 5 16:00:21 zippy ipmon[29322]: 16:00:20.496063 ep0 @10:1 p \ 192.168.1.103,33247 -> 10.1.1.77,53 PR udp len 20 61 K-S K-F

etc.

mick@homebox:~/ > ssh -2 mbauer@zippy.pinheads.com Enter passphrase for DSA key '/home/mick/.ssh/id_dsa': Last login: Wed Oct 4 10:14:34 2000 from homebox.pinheads.com Have a lot of fun...

mbauer@zippy:~ > xterm &

mick@homebox:~/ > ssh -2 -f mbauer@zippy -L 7777:zippy:110 sleep 600 Enter passphrase for DSA key '/home/mick/.ssh/id_dsa': mick@homebox:~/ > mutt

LocalForward 7777 zippy.pinheads.com:110

In other words, after the parameter name `LocalForward`, you should have a space or tab, the local port number, another space, the remote host's name or IP address, a colon but no space, and the remote port number. You can also use this parameter in */etc/ssh/ssh_config* if you wish it to apply to all *ssh* processes run on the local machine. In either case, you can define as many

local forwards as you need e.g., one for POP3, another on a different local port for IRC, etc.

[mick@kolach mick]$ su -c "ifconfig eth0" - Password: (superuser password entered here) eth0 Link encap:Ethernet HWaddr 00:10:C3:FE:99:08

inet addr:192.168.201.201 Bcast:192.168.201.255 Mask:255.255.255.0

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:989074 errors:0 dropped:0 overruns:0 frame:129

TX packets:574922 errors:0 dropped:0 overruns:0 carrier:0

[mick@kolach mick]$

[mick@kolach mick]$ sudo ifconfig eth0 We trust you have received the usual lecture from the local System Administrator. It usually boils down to these two things: #1) Respect the privacy of others.

#2) Think before you type.

Password: (mick's password entered here) eth0 Link encap:Ethernet HWaddr 00:10:C3:FE:99:08

inet addr:192.168.201.201 Bcast:192.168.201.255 Mask:255.255.255.0

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:989074 errors:0 dropped:0 overruns:0 frame:129

TX packets:574922 errors:0 dropped:0 overruns:0 carrier:0

collisions:34 txqueuelen:100

Interrupt:3 Base address:0x290 Memory:d0000-d4000

[mick@kolach mick]$

sudoers file.

#

This file MUST be edited with the 'visudo' command as root.

See the sudoers manpage for the details on how to write a sudoers file.

#

Host, User, and Cmnd alias specifications not used in this example, # but if you use sudo for more than one command for one user you'll want # some aliases defined [mdb]

User privilege specification

root ALL=(root) ALL

mick ALL=(root) /sbin/ifconfig

jeeves ALL=(root) /sbin/ifconfig -a

This sort of granular delegation is highly recommended if you use *sudo* for privilege delegation: the more unnecessary privilege you grant nonroot accounts, the less *sudo* is actually doing for you.

That's as far as we're going to go here in exploring *sudo*, though, since my main angle is remote administration and the intelligent execution thereof. In summary: be sneaky when administering Linux servers, and, whenever possible, don't be root while you're doing it.

Chapter 5. Tunneling

Most of the previous chapters in this book have concerned specific services you may want your bastion hosts to provide. These include "infrastructure services" such as DNS and SMTP, "end-user" services such as FTP and HTTP, and "administrative services"

such as SSH. This chapter falls both technologically and literally between the service-intensive part of the book and the behind-the-scenes section, since it concerns tools that are strictly means to other ends.

The means is tunneling, as this chapter's title indicates, and the ends to which we apply it involve enhancing the security of other applications and services. These applications and services may be either end-user-oriented or administrative. The tools we'll focus on in this chapter are the Stunnel encryption wrapper and the OpenSSL encryption and authentication toolkit, not because they're the only tools that do what they do, but because both are notably flexible, strong, and popular.

[root openssl-0.9.6c]# ./config --prefix=/usr/local \

--openssldir=/usr/local/ssl shared

these are the only important lines in this sample...

dir = ./CA

default_bits = 2048

...changing these saves typing when generating new certificates

countryName_default = ES

stateOrProvinceName_default = Andalucia

localityName_default = Sevilla

0.organizationName_default = Mesòn Milwaukee

organizationalUnitName_default =

commonName_default =

emailAddress_default =

I don't use unstructuredName, so I comment it out:

unstructuredName = An optional company name

[root ssl]# /usr/local/ssl/misc/CA.pl -newca

[root ssl]# /usr/local/ssl/misc/CA.sh -newca

[root@tamarin ssl]# /usr/local/ssl/misc/CA.pl -newca

CA certificate filename (or enter to create)

Making CA certificate ...

Using configuration from /usr/local/ssl/openssl.cnf

Generating a 2048 bit RSA private key

.....++++++

....++++++

writing new private key to './CA/private/cakey.pem'

Enter PEM pass phrase: **********

Verifying password - Enter PEM pass phrase: **********

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

Country Name (2 letter code) [ES]:

State or Province Name (full name) [Andalucia]:

Locality Name (eg, city) [Sevilla]:

Organization Name (eg, company) [Mesòn Milwaukee]:

Organizational Unit Name (eg, section) [ ]:

Common Name (eg, YOUR name) [ ]:Mick's Certificate Authority

Email Address [ ]:certmaestro@mesonmilwaukee.com

bash-# openssl req -nodes -new -keyout elfiero_key.pem \

-out elfiero_req.pem -days 365 -config ./openssl.cnf

bash-# openssl ca -config ./openssl.cnf -policy policy_anything \

-out elfiero_pubcert.pem -infiles elfiero_req.pem

bash-# cat ./<tt>elfiero_key.pem</tt> ./<tt> elfiero_pubcert.pem</tt> > ./<tt>elfiero_cert.pem</tt>

ALL: ALL

<tt>daemon1 [daemon2 etc.] </tt>: <tt>host1 [host2 etc.] </tt>

ssyncd 273/tcp # Secure Rsync daemon

ssync: ALL

ssync: ALL: ALLOW

[root@elfiero etc]# stunnel -d ssyncd -r localhost:rsync -p \

/etc/stunnel/elfiero_cert.pem -N ssync

ssyncd 273/tcp # Secure Rsync daemon

ssync 272/tcp # Secure Rsync forwarder

ssync: ALL

ssync: ALL: ALLOW

[root@skillet etc]# stunnel -c -d rsync -r elfiero:ssyncd -N ssync

[schmoe@skillet ~]$ rsync elfiero::

[schmoe@skillet ~]$ rsync localhost::

toolz Free software for organizing your skillet recipes

recipes Donuts, hush-puppies, tempura, corn dogs, pork rinds, etc.

images Pictures of Great American Fry-Cooks in frisky poses

medical Addresses of angioplasty providers

[schmoe@skillet ~]$ rsync --port=272 localhost::

toolz Free software for organizing your skillet recipes

recipes Donuts, hush-puppies, tempura, corn dogs, pork rinds, etc.

images Pictures of Great American Fry-Cooks in frisky poses

[root@elfiero etc]# stunnel -d telnets -p /etc/stunnel/elfiero_cert.pem -l /usr/

sbin/in.telnetd

bash-# grep SSL /etc/services

[root@skillet /root]# stunnel -c -d telnets -r elfiero:telnets

[schmoe@skillet ~]$ telnet localhost telnets

[root@elfiero etc]# stunnel -d ssyncd -r rsync -p /etc/stunnel/elfiero_cert.pem -N ssync

-v 2 -a /etc/stunnel

[root@skillet etc]# stunnel -c -d rsync -r ssyncd -p /etc/stunnel/skillet_cert.pem -N

ssync

The command on *skillet* to run the Rsync query command is exactly the same as in Example 5-5. Although in this case, the transaction is more secure; the added security is *completely transparent* to the end user.

To increase *elfiero*'s level of certificate verification from 2 to 3 (i.e., checking not only for valid signatures, but also for known certificates), there are only two additional steps:

1. Put a copy of *skillet*'s signed certificate (*skillet_pubcert.pem*, the version without *skillet*'s key) in */etc/stunnel* and rerun the command *c_rehash /etc/stunnel*.

2. Run *elfiero*'s Stunnel process with *-v* set to `3` rather than `2`.

Although it may be tempting to copy *skillet_cert.pem* (the combined key/certificate file) over to *elfiero* in addition to or instead of *skillet_pubcert.pem*, please resist this temptation: unnecessarily copying private keys is a very bad habit to get into.

5.1.4 Using Stunnel on the Server and Other SSL Applications on the Clients

Stunnel isn't the only SSL application capable of establishing a connection to an Stunnel daemon. For example, it's possible to run Stunnel on a POP3 server listening on the standard "pop3s" port TCP 995 and forwarding to a local POP3 mail daemon. It's then possible to connect to it using popular SSL-capable POP3 clients, such as Outlook Express and Eudora on client systems that don't run Stunnel.

This is actually simpler than the examples I've presented in this chapter: the server side is the same, and configuring the client side amounts to enabling SSL in your client application. See the Stunnel FAQ (http://www.stunnel.org/faq/) for more hints if you need them.

5.1.4.1 One final pointer on Stunnel: chrooting it

Although Stunnel isn't designed to be run from a chroot jail, this can be made to work with a bit of preparation. See Dave Lugo's detailed instructions at http://www.etherboy.com/stunnel/stunnelchroot if you wish to further secure Stunnel in this way. My own opinion is that this is overkill, but overkill is in the eye of the beholder.

5.1.5 Other Tunneling Tools

In addition to Stunnel, other applications can be used to create encrypted tunnels. These include Rick Kaseguma's program SSLwrap, which is similar to Stunnel, and SSH, the subject of the previous chapter. SSLwrap's home page is http://www.quiltaholic.com/rickk/sslwrap, and Chapter 4 addresses tunneling as well.

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 recent consensus document, "The Twenty Most Critical Internet Security Vulnerabilities" (http://www.sans.org/top20.htm), the number-three 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, over 50% of BIND installations are vulnerable to well-known (and in many cases, old) exploits.

So many hosts with such

vulnerabilities

in an essential service are 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' 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 name server *ns.someisp.com* for

DNS

look-ups. Since the name

"www.dogpeople.org" has no meaning

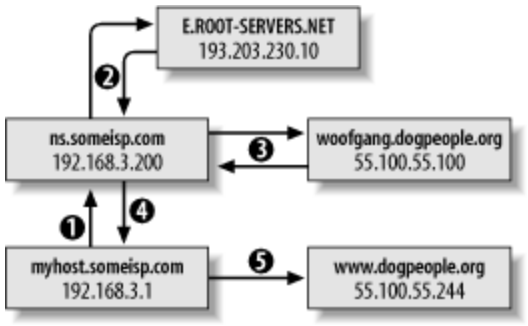
to the routers through which the web query and its responses will pass, the user's web browser needs to learn the Internet Protocol (IP) address associated with *http://www.dogpeople.org* before attempting the web query.

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. Making one or more queries in order to answer a previous query is called *recursion*.

Figure 6-1. A recursive DNS query



Ns.someisp.com begins its recursive query by asking a *root name server* for the IP address of a host that's authoritative for the zone *dogpeople.org*. (All Internet DNS servers use a static "hints" file to identify the

13 or so official root name servers. 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 *dogpeople.org* is handled by *woofgang.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 look-up. It and other single-host type look-ups are simply called *queries*; DNS

queries are handled on UDP port 53.

Not all DNS transactions involve single-host look-ups, 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

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 name servers 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*.

Name servers 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

startup that is, almost all. 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 name server'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 look-ups 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 Section 2.3 refers to the practice of maintaining both "public" and

"private" databases of each local

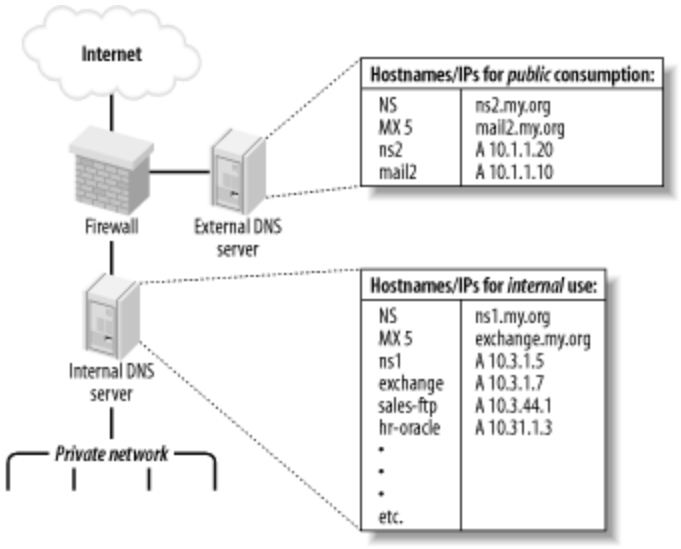
name domain (zone). The public-zone database contains as little as possible: it should have NS records for publicly accessible name servers, MX

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 may 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 zones!

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' 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, 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 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.

If, on the other hand, you wish to use DNSSEC, are already familiar with and competent at administering BIND, or need to interoperate with other DNS servers running BIND (and feel you can mitigate BIND's known and yet-to-be-discovered security issues by configuring it carefully and keeping current with security advisories and updates), then I don't think BIND is that bad a choice.

In other words, I think each has its own merits: you'll have to decide for yourself which better meets your needs. BIND is by far the most ubiquitous DNS software on the Internet, and most of my experience securing DNS servers has been with BIND. Therefore, a good portion of this chapter will focus on DNS security as it pertains to BIND Versions 8 and 9. My esteemed friend and colleague Bill Lubanovic has written most of the second half of the chapter, which covers the basic use of djbdns.

If neither BIND nor djbdns appeals to you and you choose something else altogether, you may wish to skip ahead to Section 6.4.5. That section applies to all DNS servers, regardless of what software they run.

rpm -q -v <tt>package-name</tt>

rpm -q -v -p <tt>/path/to/package.rpm</tt>

rpm -U <tt>/path/to/package.rpm</tt>

rpm -U --force <tt>/path/to/package.rpm</tt>

make depend

make all

make install

'DESTDIR=<tt>/path/to/installation_root'</tt>

./configure

make

make install

./configure --prefix= /path/to/installation_root

#! /bin/bash

(Change the above path if your bash binary lives elsewhere) # Commands to create BIND v8 chroot jail, adapted # from a script by Kyle Amon

(http://www.gnutec.com/~amonk)

YOU MUST BE ROOT TO RUN THIS SCRIPT!

First, define some paths. BINDJAIL is the root of BIND's # chroot jail.

BINDJAIL = <tt>/var/named</tt> # BINDBIN is the directory in which named, rndc, and other BIND

executables reside

BINDBIN = /usr/sbin

Second, create the chroot jail and its subdirectories mkdir -m 2750 -p $BINDJAIL/dev $BINDJAIL/etc mkdir -m 2750 -p $BINDJAIL/usr/local/libexec mkdir -m 2770 -p $BINDJAIL/var/run

mkdir -m 2770 $BINDJAIL/var/log $BINDJAIL/var/tmp mkdir -m 2750 $BINDJAIL/master

mkdir -m 2770 $BINDJAIL/slave $BINDJAIL/stubs # Third, create unprivileged user & group for named # (may already exist if you use SuSE or Mandrake, but # you should ensure that passwd entry uses # /bin/false rather than a real shell) echo "named:x:256: " >> /etc/group echo "named:x:256:256:BIND:$BINDJAIL:/bin/false" \ >> /etc/passwd

Fourth, change some permissions & ownerships chmod 2750 $BINDJAIL/usr $BINDJAIL/usr/local chmod 2750 $BINDJAIL/var

chown -R root:<tt>named</tt> $BINDJAIL

Fifth, copy some necessary things into the jail # Next line may be omitted in most cases cp $BINDBIN/named $BINDJAIL

Remaining lines, however, usually necessary -

these are things BIND needs in the chroot jail in # order to work properly.

cp $BINDBIN/named-xfer $BINDJAIL/usr/local/libexec cp $BINDBIN/ndc $BINDJAIL/ndc

cp /etc/localtime $BINDJAIL/etc

mknod $BINDJAIL/dev/null c 1 3

chmod 666 $BINDJAIL/dev/null

daemon syslogd -m 0

daemon syslogd -m 0 -a /var/named/dev/log

chroot <tt>/path/to/chroot_jail</tt> ./ndc <tt>[ndc command] </tt>

#!/bin/bash

(Change the above path if your bash binary lives elsewhere)

#

Commands to create BIND v9 chroot jail, adapted # from a script by Kyle Amon (http://www.gnutec.com/~amonk) # and from the Chroot-BIND-HOWTO (http://www.linuxdoc.org) # YOU MUST BE ROOT TO RUN THIS SCRIPT!

First, define some paths. BINDJAIL is the root of BIND's # chroot jail.

BINDJAIL = /var/named

BINDBIN is the directory in which named, rndc, and other BIND

executables reside

BINDBIN = /usr/sbin

Second, create the chroot jail and its subdirectories.

mkdir -m 2750 -p $BINDJAIL/dev $BINDJAIL/etc mkdir -m 2770 -p $BINDJAIL/var/run

mkdir -m 2770 $BINDJAIL/var/log $BINDJAIL/var/tmp mkdir -m 2750 $BINDJAIL/master

mkdir -m 2770 $BINDJAIL/slave $BINDJAIL/stubs # Third, create unprivileged user & group for named # (may already exist if you use SuSE or Mandrake, but # you should ensure that passwd entry uses # /bin/false rather than a real shell) echo "named:x:256:" >> /etc/group echo "named:x:256:256:BIND:$BINDJAIL:/bin/false" \ >> /etc/passwd

Fourth, give named some control over its own volatile files chown -R root:named $BINDJAIL

Fifth, copy some necessary things into the jail # Next line may be omitted in most cases cp $BINDBIN/named $BINDJAIL

Remaining lines, however, usually necessary -

these are things BIND needs in the chroot jail in # order to work properly.

cp /etc/localtime $BINDJAIL/etc

mknod $BINDJAIL/dev/null c 1 3

chmod 666 $BINDJAIL/dev/null

named -u named -g wheel -t /var/named -c /etc/named.conf

By the way, comments in named.conf can look like this...

// or like this...

/* or like this. */

acl trustedslaves { 192.168.20.202; 192.168.10.30}; acl bozos { 10.10.1.17; 10.10.2.0/24; }; acl no_bozos { localhost; !bozos; }; options {

directory "/"; listen-on { 192.168.100.254; }; recursion no; fetch-glue no; allow-transfer { trustedslaves; }; };

logging {

channel seclog {

file "var/log/sec.log" versions 5 size 1m; print-time yes; print-category yes; };

category xfer-out { seclog; }; category panic { seclog; }; category security { seclog; }; category insist { seclog; }; category response-checks { seclog; }; };

zone "coolfroods.ORG" {

type master;

file "master/coolfroods.hosts"; };

zone "0.0.127.in-addr.arpa" {

type master;

file "master/0.0.27.rev"; };

zone "100.168.192.in-addr.arpa" {

type master;

file "master/100.168.192.rev"; };

acl <tt>acl_name</tt> { <tt>IPaddress</tt>; <tt>Networkaddress</tt>; <tt>acl_name</tt>; <tt>etc.</tt> };

{!10.1.2.3; 10.0.0.0/8; }

{ 10.0.0.0/8; !10.1.2.3; }

acl bozos { 10.10.1.17; 10.10.2.0/24; }; acl no_bozos { localhost; !bozos; };

channel <tt>channel-name</tt> {

<tt>filename</tt> <tt>[file-options-list]</tt> | syslog <tt>syslog-facility</tt> | <tt>null</tt> ; <tt>[</tt> print-time <tt>yes</tt>|<tt>no</tt>; <tt>]</tt> <tt>[</tt> print-category <tt>yes</tt>|<tt>no</tt>; <tt>]</tt> <tt>[</tt> print-severity <tt>yes</tt>|<tt>no</tt>; <tt>]</tt> <tt> [</tt>severity <tt>severity-level</tt>; <tt>]</tt> };

logging {

channel seclog {

file "var/log/sec.log" versions 3 size 1m; print-time yes; print-category yes; };

category <tt>category-name</tt> { <tt>channel-list</tt> ; };

view "<tt>view-name</tt>" {

match-clients { <tt>match-list</tt>; }; recursion yes|no; zone "<tt>domain.name</tt>" {

<tt> // standard BIND 8/9 zone{} contents here</tt> };

<tt>// additional zones may be defined for this view as well</tt> };

view "inside" {

// Our internal hosts are: match-clients { 192.168.100.0/24; }; // ...and for them we'll do recursive queries...

recursion yes; // Here are the zones we'll serve for them: zone "ourorg.ORG" {

type master;

file "master/ourorg_int.hosts"; };

// Here's a subdomain that isn't searchable in any form by outsiders zone "intranet.ourorg.ORG" {

type master;

file "master/intranet.ourorg.hosts"; };

};

view "outside" {

//Client view for "none of the above"

match-clients { any; }; // We don't recurse for the general public recursion no; // Answer outside queries from a stripped-down zone file zone "ourorg.ORG" {

type master;

file "master/ourorg_ext.hosts"; };

};

$TTL 86400

// Note: global/default TTL must be specified above. BIND v8 didn't check for this, // but BIND v9 does.

@ IN SOA cootie.boneheads.com. hostmaster.boneheads.com. (

2000060215 ; serial 10800 ; refresh (3H) 1800 ; retry (30m) 120960 ; expiry (2w) 43200) ; RR TTL (12H) IN NS ns.otherdomain.com.

IN NS cootie.boneheads.com.

IN MX 5 cootie.boneheads.com.

blorp IN A 10.13.13.4

cootie IN A 10.13.13.252

cootie IN HINFO MS Windows NT 3.51, SP1

@ IN RP john.smith.boneheads.com. dumb.boneheads.com.

dumb IN TXT "John Smith, 612/231-0000"

dnskeygen -H 512 -h -n <tt>keyname</tt>

key <tt>keyname</tt> {

algorithm hmac-md5; secret "<tt>insert key-string from either keyfile here</tt>"; }

server <tt>IP address of remote server </tt>{

transfer-format many-answers; # (send responses in batches rather than singly) keys { <tt>keyname</tt>; }; };

key mon_key {

algorithm hmac-md5; secret

"ff2342AGFASsdfsa55BSopiue/u2342LKJDJlkjVVVvfjweovzp2OIPOTXUEdss2jsdfAAlskj=="; }

acl goodmonkeys { 10.10.100.13; key mon_key ; };

allow-transfer { goodmonkeys; };

This statement, which could be nested in either an *options{}* statement or a *zone{}* statement (depending on whether it's global or zone specific), says that zone-transfer requests will only be honored if they match the acl *goodmonkeys*, i.e., only if the requests come from 10.10.100.13 and are signed with the key *mon_key*.

6.4.7 Sources of BIND (and IS Security) Information

The guidelines and techniques we've covered here should give you a good start on securing your BIND server(s). For more in-depth understanding of these techniques, I strongly recommend you read the BIND v8 Operators' Guide and the BIND v9 Administrators' Reference Manual. For me at least, these are among the most useful documents provided in any OSS package. Another excellent source of BIND security information is Liu's "DNS Security" slideshow. Section 6.6 at the end of this chapter lists information about these and other BIND resources.

Equally important, every BIND user should subscribe to at least one security-advisory email list. BUGTRAQ is my personal favorite, since it's

both timely and inclusive (but it's also high volume; I recommend the digest version). See http://www.securityfocus.com/cgi-bin/subscribe. pl for an online subscription form. Another excellent list is VulnWatch, which has no digest but is much lower volume than BUGTRAQ. See http://www.vulnwatch.org/subscribe.html for more details.

I also recommend that you look up and read the CERT advisories listed in Section 6.6 at the end of this chapter. Understanding past BIND vulnerabilities is essential to understanding BIND security.

$ wget http://cr.yp.to/daemontools/daemontools-0.76.tar.gz

$ tar xvzf daemontools-0.76.tar.gz

$ rm daemontools-0.76.tar.gz

$ cd admin/daemontools-0.76

./package/install

$ wget http://cr.yp.to/djbdns/djbdns-1.05.tar.gz

$ tar xvzf djbdns-1.05.tar.gz

$ rm djbdns-1.05.tar.gz

$ cd djbdns-1.05

$ make

make setup check

adduser -s /bin/false dnscache

adduser -s /bin/false dnslog

dnscache-conf

acct logacct dir ip

/usr/local/bin/dnscache-conf dnscache dnslog /etc/dnscache 127.0.0.1

ln -s /etc/dnscache

/service

nameserver 127.0.0.1

$ tail /service/dnscache/log/main/current

@400000003bd238e539184794 rr 401c4337 86400 ns slashdot.org. ns1.andover.net.

@400000003bd238e539185f04 rr 401c4337 86400 ns slashdot.org. ns2.andover.net.

@400000003bd238e53918728c rr 401c4337 86400 ns slashdot.org. ns3.andover.net.

@400000003bd238e539188614 rr 401c4337 86400 cname www.slashdot.org. slashdot.org.

@400000003bd238e539189d84 cached 1 slashdot.org.

@400000003bd238e53918a93c sent 627215 64

@400000003bd238f62b686b4c query 627216 7f000001:1214:a938 12 20.113.25.24.in-addr.arpa.

@400000003bd238f62b689644 cached 12 20.113.25.24.in-addr.arpa.

@400000003bd238f62b68a9cc sent 627216 88

10-20 21:54:19 rr 64.28.67.55 086400 a slashdot.org. 64.28.67.150

10-20 21:54:19 rr 64.28.67.55 086400 ns slashdot.org. ns1.andover.net.

10-20 21:54:19 rr 64.28.67.55 086400 ns slashdot.org. ns2.andover.net.

10-20 21:54:19 rr 64.28.67.55 086400 ns slashdot.org. ns3.andover.net.

10-20 21:54:19 rr 64.28.67.55 086400 cname www.slashdot.org. slashdot.org.

10-20 21:54:19 cached a slashdot.org.

10-20 21:54:19 sent 627215

10-20 21:54:36 query 627216 127.0.0.1:4628:43320 ptr 20.113.25.24.in-addr.arpa.

10-20 21:54:36 cached ptr 20.113.25.24.in-addr.arpa.

10-20 21:54:36 sent 627216

adduser -s /bin/false dnscache

adduser -s /bin/false dnslog

/usr/local/bin/dnscache-conf dnscache dnslog /etc/dnscachex 192.168.100.9

ln -s /etc/dnscachex /service

touch /etc/dnscachex/root/ip/192.168.100

nameserver 192.168.100.9

adduser -s /bin/false tinydns

adduser -s /bin/false dnslog

tinydns-conf acct

logacct dir

ip

/usr/local/bin/tinydns-conf tinydns dnslog /etc/tinydns 208.209.210.211

ln -s /etc/tinydns /service

cd /service/tinydns/root

./add-ns hackenbush.com 192.193.194.195

./add-childns hackenbush.com 200.201.202.203

./add-host hugo.hackenbush.com 192.193.194.200

./add-alias another.hackenbush.com 192.193.194.200

./add-mx mail.hackenbush.com 192.193.194.201

make

.hackenbush.com:192.193.194.195:a:259200

&hackenbush.com:200.201.202.203:a:259200

=hugo.hackenbush.com:192.193.194.200:86400

+another.hackenbush.com:192.193.194.200:86400

@mail.hackenbush.com:192.193.194.201:a::86400

.hackenbush.com:192.193.194.195:a

&hackenbush.com:200.201.202.203:a

=hugo.hackenbush.com:192.193.194.200

+another.hackenbush.com:192.193.194.200

@mail.hackenbush.com:192.193.194.201:a

.hackenbush.com:192.193.194.195:a

&hackenbush.com::ns.flywheel.com

.hackenbush.com::ns.flywheel.com

$ wget http://cr.yp.to/ucspi-tcp/ucspi-tcp-0.88.tar.gz

$ tar xvzf ucspi-tcp-0.88.tar.gz

$ cd ucspi-tcp.0.88

$ make

make setup check

axfr-get <tt>dom</tt> <tt>file</tt> <tt>file.tmp </tt>

all: data.cdb

flywheel.data:

    /usr/local/bin/tcpclient -i \

    a.ns.hackenbush.com \

    53 \

/usr/local/bin/axfr-get \

flywheel.com \

flywheel.data \

flywheel.tmp

data: hackenbush.data flywheel.data

cat *.data > data

data.cdb: data

usr/local/bin/tinydns-data

axfrdns-conf axfrdns dnslog /etc/axfrdns /etc/tinydns 192.193.194.195

cd / etc /axfrdns

200.201.202.203:allow,AXFR="hackenbush.com,194.193.192.in-addr.arpa"

make

ln -s /etc/axfrdns /service

remote: data.cdb

```
rsync -az -e ssh data.cdb 192.193.194.195:/service/tinydns/root/data.cdb

data.cdb: data


	/usr/local/bin/tinydns-data
```

You will normally be prompted for a passphrase by *ssh*. To avoid this, create a key pair and copy the public key to the user's directory on the secondary server. Details can be found in SSH, The Definitive Guide (O'Reilly).

That's it! Now, whenever you make changes to *tinydns*, whether through the helper applications or by directly editing zone files and typing *make* to publish them, the database *data.cdb* will be copied to the secondary server. Using *rsync* guarantees that only changed portions will be copied. Using *ssh* guarantees that the data will be encrypted in transit and protected against snooping or modification.

Alternatively, you can *rsync* the datafile rather than the *data.cdb* database and then run *make* on the secondary server to create the database.I

6.5.9 Migrating from BIND

If you are only using BIND as a caching server, then installing *dnscache* will replace BIND completely. Don't forget to turn off the *named* process.

If BIND is serving data on your domains and it's configured like most, it can be replaced by *tinydns*. Some newer features like DNSSEC and IXFR are not supported, but *ssh* and *rsync* provide simpler and better functionality.

Bernstein describes at length how to migrate your site from BIND to *tinydns* in http://cr.yp.to/djbdns/frombind.html. This description includes the following:

- Using *axfr-get* to get zone data from a BIND server and convert it to *tinydns-data* format.
- Replacing serial numbers and TTLs with automatic values.
- Merging record types.
- Testing your setup while BIND is running and replacing it gracefully.

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

1. *comp.protocols.tcp-ip.domains* USENET group: "FAQ." Web site:

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

   Frequently Asked Questions about DNS.

2. Rowland, Craig. "Securing BIND." Web site:

   http://www.psionic.com/papers/whitep01.html.

   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-2002-15

"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

1. 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.

2. Internet Software Consortium. "BIND 9 Administrator Reference Manual." Included with BIND 9 source-code distributions in the directory *doc/arm*, filename *Bv9ARM.html*. Also available in PDF format from http://www.nominum.com/content/documents/bind9arm.pdf.

   The ARM is the most important and useful piece of official BIND 9

   documentation.

3. Internet Software Consortium. "Internet Software Consortium: BIND." Web site: http://www.isc.org/products/BIND/. Definitive source of all BIND software and documentation.

4. Liu, Cricket.

   "Securing an Internet Name Server."

   Slide show, available at http://www.acmebw.com/papers/securing.pdf. A presentation by Cricket Liu, coauthor of *DNS and BIND* (a.k.a. "The Grasshopper

   Book").

6.6.3 djbdns Resources

1. Bernstein, D. J. "djbdns: Domain Name System Tools." Web site: http://cr.yp.to/djbdns.html. The definitive source of *djbdns* software and documentation.

2. Brauer, Henning. "Life with djbdns." Web

   site: http://lifewithdjbdns.org.

   A comprehensive guide to using *djbdns*, including sample configurations and links to other sites.

3. Nelson, Russell. "djbdns Home Page." Web

   site: http://www.djbdns.org.

   Official source of *axfr* tool, with lots of other useful information and links.

4. "FAQTS Knowledge Base...

   djbdns." Web site: http://www.faqts.com/knowledge_base/index.phtml/fid/699/.

Frequently asked questions about *djbdns*.

5. "Linux notebook/djbdns." Web site: http://binarios.com/lnb/djbdns.html#djbdns.

   Notes on running *djbdns* under Linux, by a user in Portugal.

Chapter 7. 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 mailbox or crash their email server
- 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

The scope and severity of these threats are not helped by the complication inherent in running an Internet email server, specifically a

Mail Transfer Agent (MTA).

It requires a working understanding of the

Simple Mail Transfer Protocol (SMTP), as well as a mastery of your MTA application of choice. There really aren't any shortcuts around either requirement (although some MTAs 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.

7.1 Background: MTA and SMTP Security

MTAs move email from one host or network to another. These are in contrast to 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, or MUAs, of which Mutt and Outlook Express are popular examples. There is no real-world simile for 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. Nonetheless, for the remainder of this chapter, I'll assume you're interested in

using your MTA for SMTP transactions, since SMTP is the dominant mail-transfer protocol of the Internet.

7.1.1 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 that 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.

7.1.2 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 http://www.cert.org/advisories/CA-1997-05.html).

Another danger is abuse of the SMTP server's configuration, that is, using the server in ways not anticipated or desired by its owners. The most widespread form of SMTP abuse is relaying. Spammers and system crackers alike rejoice when they find an SMTP server that blindly accepts mail from external entities for delivery to other external entities.

Such

"

open

relays" can be used to obfuscate the true origin of a message and to forward large quantities of

Unsolicited Commercial Email (UCE) and other undesirable email. For example, open SMTP relays were an important attack vector for the

"Hybris" worm as described in CERT® Incident Note IN-2001-02 (*Open mail relays used to deliver "Hybris*

Worm," http://www.cert.org/incident_notes/IN-2001-02.html).

Still another security risk in SMTP is that one's MTA will leak user and system information to prospective intruders.

Like SMTP abuse, SMTP "intelligence

gathering" usually capitalizes on sloppy or incorrect software configuration rather than bugs per se.

The main difference between abuse and probes is intent: somebody who relays UCE through your server probably doesn't care about the server itself or the networks to which it's connected; they care only about whether they can use them for their own purposes. But somebody who probes an SMTP

server for usernames, group memberships, or debugging information is almost certainly interested in compromising that SMTP server and the network on which it resides.

Historically, two SMTP commands specified by RFC 2821

(*Simple Mail Transfer Protocol*, available at ftp://ftp.isi.edu/in-notes/rfc2821.txt) have been prolific leakers of such information: *VRFY* , which verifies whether a given username is valid on the system and, if so, what the user's full name is; and

EXPN , which expands the specified mailing-list name into a list of individual account names.

A third SMTP command,

VERB , can be used to put some remote MTAs into

"verbose" mode.

VERB is an Extended SMTP command and was introduced in RFC 1700 (*Assigned Numbers*). Since one of the guiding principles in IS security is "never reveal anything to strangers

unnecessarily," you should *not* allow any publicly accessible MTA server to run in verbose mode.

EXPN, *VRFY*, and *VERB* are throwbacks to a simpler time when legitimate users wanting such information were far more numerous than mischievous strangers up to no good. Your MTA should be configured either to ignore *VRFY* and *EXPN*

requests or to falsify its responses to them, and to disregard *VERB* requests.

7.1.3 Unsolicited Commercial Email

Unsolicited Commercial Email (UCE) isn't a security threat in the conventional sense: sending UCE generally isn't illegal, nor is it a direct threat to the integrity or confidentiality of anyone's data. However, if somebody uses

your bandwidth and *your* computing resources (both of which can be costly) to send you something you don't want sent,

isn't this actually a kind of theft? I think it is, and many people agree. Rather than being a mere annoyance, UCE is actually a serious threat to network availability, server performance, and bandwidth optimization.

Unfortunately, UCE is difficult to control. Restricting which hosts or networks may use your SMTP

gateway as a relay helps prevent that particular abuse, but it doesn't prevent anyone from delivering UCE

to your network. Blacklists, such as the Realtime Blackhole List (http://mail-abuse.org/rbl/), that identify and reject email from known sources of UCE can help a great deal, but also tend to result in a certain amount of legitimate mail being rejected, which for some organizations, is

unacceptable. Anyhow, blacklists are a somewhat crude way to address UCE.

A much better approach is to use scripts such as SpamAssassin

(available at http://www.spamassassin.org) to evaluate each incoming email message against a database of known UCE

characteristics. With some fine tuning, such scripts can radically reduce one's UCE load. Depending on the volume of email arriving at your site, however, they can also increase CPU

loads on your SMTP gateway.

7.1.4 SMTP AUTH

SMTP exploits, relaying, and abuse, including UCE, are all SMTP problems; they're risks endemic to the SMTP protocol and thus to many SMTP Mail Transfer Agents. But surely there's *some* proactive security feature in SMTP?

Until recently, there wasn't: SMTP was designed with no security features at all, not even the most rudimentary authentication mechanism. But that changed in 1999 with the introduction of RFC 2554, *SMTP Service Extension for Authentication* (known more simply as "SMTP *AUTH*"),

which provided the SMTP protocol with a modular authentication framework based on the generic Simple Authentication and Security Layer (SASL) described in RFC 2222.

SMTP *AUTH* allows your MTA to authenticate prospective clients via one of several authentication schemes. In this way, you can more effectively control such activities as SMTP

relaying, and you can also provide SMTP services to remote users, even if their IP address is unpredictable.

It's far from a panacea, and it isn't even supported by all MTAs, but SMTP

A*UTH* is a badly needed improvement to the venerable SMTP protocol. Both MTAs we discuss in this chapter support SMTP *AUTH*.

$ 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.

helo woofgang.dogpeople.org

mail from:<mick@dogpeople.org>

rcpt to:<groucho@hackenbush.com>

data

quit

> *QUIT* closes the SMTP session.

My own procedure to test any SMTP server I set up is first to deliver a message this way from the server to itself i.e., `telnet localhost 25`. If that succeeds, I then try the same thing from a remote system.

This technique doesn't work for advanced setups like SMTP over TLS (covered later in this chapter), but it's a fast, simple, and reliable test for basic SMTP server configurations, especially when you need to verify that antirelaying and other controls have been set correctly.

7.3 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 was 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 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

also written a book on it, *Exim: The Mail Transfer Agent* (O'Reilly).

I mention Qmail and Exim because they 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, respectively. 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.

rpm -qv sendmail

dpkg -s sendmail

divert(-1)

dnl This is a comment line

include(`/usr/lib/sendmail-cf/m4/cf.m4') VERSIONID(`Mail server')dnl

OSTYPE(`linux')

define(`confDEF_USER_ID',``8:12")dnl define(`confPRIVACY_FLAGS',
`authwarnings,needmailhelo,noexpn,novrfy')dnl
define(`confSMTP_LOGIN_MSG', ` Sendmail')dnl
define(`confSAFE_FILE_ENV', `/var/mailjail')dnl
define(`confUNSAFE_GROUP_WRITES')dnl
undefine(`UUCP_RELAY')dnl

undefine(`BITNET_RELAY')dnl

FEATURE(`access_db',`hash -o /etc/mail/access.db')dnl
FEATURE(`smrsh',`/usr/sbin/smrsh')dnl FEATURE(`dnsbl')dnl

FEATURE(`blacklist_recipients')dnl

FEATURE(`mailertable',`hash -o /etc/mail/mailertable.db')dnl
FEATURE(`virtusertable',`hash -o /etc/mail/virtusertable.db')dnl
FEATURE(`use_cw_file')dnl

FEATURE(`masquerade_entire_domain')dnl
FEATURE(`masquerade_envelope')dnl

FEATURE(`nouucp')dnl

MASQUERADE_AS(`hackenbush.com')dnl

MASQUERADE_DOMAIN(`.hackenbush.com')dnl
EXPOSED_USER(`root')dnl

MAILER(smtp)dnl

MAILER(procmail)dnl

Cwlocalhost.localdomain

groupadd -g 12 mail

useradd -u 8 -g 12 -d /var/spool/mail -s /bin/false mail

bash$ mkdir -p /var/mailjail/var/spool/mqueue bash$ chown -R 8:12 /var/mailjail* bash$ chmod -R 1755 /var/mailjail/var/spool/mqueue

bash-# makemap -l hash

btree

bash-# m4 /etc/mail/sendmail.mc > /etc/sendmail.cf

bash-# /etc/init.d/sendmail restart

localhost

localhost.localdomain

polkatistas.org

mail.polkatistas.org

tubascoundrels.net

mail.tubascoundrels.net

fake.polkatistas.org local:postmaster .polkatistas.org smtp:%2

polkatistas.org smtp:internalmail.polkatistas.org .
smtp:internalmail.polkatistas.org

bash-# makemap hash /etc/mail/mailertable.db <
/etc/mail/mailertable

bash-# make mailertable

localhost.localdomain RELAY

localhost RELAY

127.0.0.1 RELAY

192.168 RELAY

bash-# makemap hash /etc/mail/mailertable.db < /etc/mail/mailertable

postmaster@tubascoundrels.net root @polkatistas.org polkawrangler

@lederhosendudes.net %1@anniefauxfanny.edu

bash-# makemap hash /etc/mail/virtusertable.db < /etc/mail/virtusertable

postmaster: root

root: mick

michael: mick@visi.com

mailstooges: mick, larry, curly

postmaster: root

root: postmaster

pwcheck_method: sasldb

saslpasswd <tt>username</tt>

bash-# saslpasswd maildroid Password: Ch1mp? ,03fuzz fl0ppi Again (for verification): Ch1mp? ,03fuzz fl0ppi

bash-# sasldblistusers user: maildroid realm: dmzmail.polkatistas.org mech: PLAIN

user: maildroid realm: dmzmail.polkatistas.org mech: CRAM-MD5

user: maildroid realm: dmzmail.polkatistas.org mech: DIGEST-MD5

saslpasswd -d <tt>username</tt>

pwcheck_method: pam

TRUST_AUTH_MECH(`CRAM-MD5 DIGEST-MD5')dnl
define(`confAUTH_MECHANISMS', `CRAM-MD5 DIGEST-MD5')dnl

define(`confAUTH_MECHANISMS', `CRAM-MD5 DIGEST-MD5 LOGIN PLAIN')dnl define(`confDEF_AUTH_INFO', `/etc/mail/default-auth-info')dnl

authorization_identity # (i.e., username) authentication_identity # (usually identical to username) secret # (password created on other host with <tt> saslpasswd</tt> realm # (usually the FQDN of the other host)

maildroid

maildroid

Ch1mp? ,03fuzz fl0ppi

dmzmail.polkatistas.org

TRUST_AUTH_MECH(`CRAM-MD5 DIGEST-MD5 LOGIN PLAIN')dnl define(`confAUTH_MECHANISMS', `CRAM-MD5 DIGEST-MD5 LOGIN PLAIN')dnl

dmzmail:/etc/mail/certs # ls -l total 30

drwxr-x--- 2 root root 272 Feb 16 20:39 .

drwxr-xr-x 4 root root 1293 Feb 16 20:38 ..

-rw------- 1 root root 1367 Feb 16 18:55 cacert.pem -rw------- 1 root root 2254 Feb 16 20:36 newcert_key.pem -rw------- 1 root root 3777 Feb 16 20:32 newcert_signed.pem

define(`CERT_DIR', `/etc/mail/certs')dnl define(`confCACERT_PATH', `CERT_DIR')dnl define(`confCACERT', `CERT_DIR/cacert.pem')dnl define(`confSERVER_CERT', `CERT_DIR/newcert_signed.pem')dnl define(`confSERVER_KEY', `CERT_DIR/newcert_key.pem')dnl define(`confCLIENT_CERT', `CERT_DIR/newcert_signed.pem')dnl define(`confCLIENT_KEY', `CERT_DIR/newcert_key.pem')dnl

After you set these directives, regenerate *sendmail.cf*, and restart *sendmail*, your server will accept encrypted SMTP sessions via the *STARTTLS* command.

myhostname = fearnley.polkatistas.org

myorigin = $mydomain

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

root: mick

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

mydestination = <tt>/path/to/mydests.txt</tt>

bash-# export POSTFIX_DIR=/var/postfix

bash-# ./LINUX2

==========

service type private unpriv chroot wakeup maxproc command + args

(yes) (yes) (yes) (never) (50)

==========

smtp inet n - y - - smtpd

pickup unix n n y 60 1 pickup

cleanup unix - - y - 0 cleanup

qmgr unix n - y 300 1 qmgr

#qmgr fifo n - n 300 1 nqmgr

tlsmgr fifo - - n 300 1 tlsmgr

rewrite unix - - y - - trivial-rewrite

bounce unix - - y - 0 bounce

defer unix - - y - 0 bounce

flush unix - - n 1000? 0 flush

smtp unix - - y - - smtp

showq unix n - y - - showq

error unix - - y - - error

local unix - n n - - local

lmtp unix - - y - - lmtp

procmail unix - n n - - pipe

flags=R user=cyrus argv=/usr/bin/procmail -t -m

USER=${user} EXT=${extension} /etc/procmailrc

postmaster: root

mailer-daemon: root

hostmaster: root

root: bdewinter

mailguys: bdewinter,mick.bauer

mick.bauer: mbauer@biscuit.stpaul.dogpeople.org

clients: :include:/etc/postfix/clientlist.txt

spam-reports: /home/bdewinter/spambucket.txt

smtpd_recipient restrictions =

permit_mynetworks

hash:/etc/postfix/access

reject_unauth_destination

-

Create */etc/postfix/access* (do a `man 5 access` for format/syntax)

-

Run `postmap hash:/etc/postfix/access` to convert the file into a database. Repeat Step 4 each time you edit */etc/postfix/access*.

smtpd_client_restrictions

Use this parameter to block mail from specific senders or originating domains. Senders to block may be named both specifically, via an external mapfile such as the access database, and generally, via values such as the following:

reject_maps_rbl

Enables use of the Real Time Blackhole List described in Section 7.6.2 of this chapter; this requires `maps_rbl_domains` to be set

reject_unknown_client

Rejects mail from clients whose hostname can't be determined

See the file */etc/postfix/sample-smtpd.cf* for a full list of valid `smtpd_client_restrictions` settings.

maps_rbl_domains

Specifies one or more Blackhole database providers, e.g. blackholes.mail-abuse.org.

7.6 Resources

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

7.6.1 SMTP Information

1. ftp://ftp.isi.edu/in-notes/rfc2821.txt. RFC

   2821, "Simple Mail Transfer

   Protocol." (Useful for making sense of mail logs, SMTP headers, etc.)

2. http://www.sendmail.org/~ca/email/other/cagreg.html.

   Shapiro, Gregory Neil.

   "Very brief introduction to create a CA and a CERT.". (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.)

7.6.2 Sendmail Information

1. Costales, Bryan, with Eric Allman. *sendmail*, Sebastopol, CA: O'Reilly & Associates, 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).

2. http://www.itworld.com/Net/3314/swol-0699-security/.

Fennelly, Carole. "Setting up Sendmail on a Firewall, Part III." Unix Insider 06/01/1999. (Excellent article on running Sendmail 8.9 and later in a chroot environment.)

3. http://www.sendmail.net/000705securitygeneral.shtml.

   Allman, Eric and Greg Shapiro. "Securing Sendmail." (Describes many built-in security features in Sendmail and offers security tips applicable to most Sendmail installations.)

4. http://www.sendmail.net/000710securitytaxonomy.shtml.

   Durham, Mark.

   "Securing Sendmail on Four Types of Systems."

5. http://www.sendmail.net/usingsmtpauth.shtml.

   Durham, Mark. "Using SMTP AUTH in Sendmail 8.10."

6. http://www.sendmail.net/810usingantispam.shtml.

   "Using New AntiSpam Features in Sendmail 8.10."

7. http://www.sendmail.org/~ca/email/starttls.html.

   "SMTP STARTTLS in sendmail/Secure

   Switch."

8. http://mail-abuse.org/rbl. Home of the Realtime Blackhole List, which is a list of known sources of UCE.

7.6.3 Postfix Information

1. http://www.postfix.org. (The definitive source for Postfix and its documentation.)

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

   (Archive site for the Postfix mailing list.)

Chapter 8. Securing Web Services

You've toiled for hours crafting your firewall rules and hardening your email and DNS

services. You believe that no evil force could breach your fortress walls. But now you blast a hole straight through those walls to a port on your server. Then you let anyone in the world run programs on your server at that port, *using their own input*.

These are signs of an unbalanced mind or of a web administrator.

The Web has many moving parts and is a frequent source of security problems. In this chapter, I assume that you are hosting web servers and are responsible for their security. I dwell on servers exposed to the Internet, but most of the discussion applies to intranets and extranets as well. The platform is

LAMP: *L*inux, *A*pache, *M*ySQL, *P*HP

(and

*P*erl).

I'll talk about *A*, *M*, and *P* here (with no slight intended to Java, Python, or other good tools). Protect your whole web environment server, content, applications and keep the weasels out of your web house.

For other views and details on

web

security, see Lincoln Stein's *World Wide Web Security FAQ*

(http://www.w3.org/Security/Faq/) and the book *Web Security, Privacy and Commerce* by Simson Garfinkel

with Gene Spafford (O'Reilly).

8.1 Web Server Security

Bad things happen to good servers. What can happen? Where should you look? The Web has the same problems as the other important Internet services discussed in this book, differing mainly in the details.

8.1.1 Problems and Goals

Malice or mistake, whether local or remote, can foil the security goals mentioned in the first chapter. Table 8-1 lists some security problems you may encounter, as well as the desired goals.

Table 8-1. Web-security problems and goals

Sample problems	Security goals
Theft of service Warez or pornography uploads Pirate servers and applications Password sniffing Rootkit and trojan program installation Denial of service targeting or 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 attacks	System and network availability

Crash or freeze from resource exhaustion (e.g., memory, disk, process space, file descriptors, or database connections)	

8.1.2 What, When, and Where to Secure

Vulnerabilities exist everywhere, but some are more frequently targeted:

Code

: Buffer overflows, string-format hacks, race conditions, logic errors, memory leaks

Files

: Ownership, permissions, symbolic links, setuid/setgid

Authentication and authorization

: Coverage gaps, data leaks, spoofing

Network

: Promiscuous mode, denial of service; connectivity

System

: User accounts, passwords

I'll describe web-server security more or less in chronological order, pointing out the problems and best practices as we go:

Build time

: Obtaining and installing Apache

Setup time

: Configuring Apache

Runtime

: Securing CGI scripts, with PHP and Perl examples

Special topics

: Issues spanning the operating system, web server, and CGI scripts: authentication, authorization, sessions, SSL, and others

8.1.3 Some Principles

Many times, I'll invoke one or more of these security mantras:

Simplify

> Configure with *least privilege*. Avoid using *root* and restrict file ownership and permissions. Provide the bare minimum to serve files, run CGI scripts, and write logs.

Reduce

> Minimize *surface area*; a smaller target is harder to hit. Disable or remove unneeded accounts, functions, modules, and programs. Things that stick out can break off.

Strengthen

> *Never trust user input*. Secure access to external files and programs.

Diversify

> Use layers of protection. Don't rely on security by the obscurity of a single mechanism, such as a password.

Document

> Write down what you've done because you won't remember it. Trust us on this one.

8.2 Build Time: Installing Apache

A secure web service starts with a secure web server, which in turn, starts with good code no buffer overflows, race conditions, or other problems that could be exploited to gain root privileges. It should be immune to remote root exploits by the swarming script kiddies. By any criteria, Apache is pretty good. No serious exploit has been reported since January 1997; security patches have addressed minor vulnerabilities.

Apache's main competition among web servers, Microsoft's Internet Information Server (IIS), has had many critical and ongoing security problems. A Microsoft Security Bulletin issued in April 2002 describes *ten* 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.

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.

As this was written, Apache 2.0 was released for general availability after years of development and testing. It will take a while for this to settle down and percolate into Linux distributions and existing systems, so the 1.3 family is still maintained. I'll cover 1.3 configuration here, with mentions of 2.x where it differs.

See http://www.apacheweek.com/security/ for Apache security news.

8.2.1 Starting Installation

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.

8.2.1.1 Setting up Your firewall

A public web server is commonly located with email and name servers in a DMZ, between outer and inner firewalls. If you're doing your own hosting, you need at least one layer of protection between your public web server and your internal network.

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 should then reopen access to the world.

8.2.1.2 Checking Your Apache version

If you have Linux, you almost certainly already have Apache somewhere. Check your version with the following command:

```
httpd -v
```

Check the Apache mirrors (http://www.apache.org/mirrors/) or your favorite Linux distribution site for the most recent stable release of Apache, and keep up with security updates as they're released. Red Hat publishes overall security updates at http://www.redhat.com/apps/support/errata/.

If you're running an older version of Apache, you can build a new version and test it with another port, then install it when ready. If you plan to replace any older version, first see if another copy of Apache (or another web server) is running:

```
service httpd status
```

or:

```
ps -ef | grep httpd
```

If Apache is running, halt it by entering the following: apachectl stop

or (Red Hat only):

```
service httpd stop
```

or:

```
/etc/init.d/apache stop
```

Make sure there aren't any *other* web servers running on port 80: netstat -an | grep ':80'

If you see one, `kill -9` its process ID, and check that it's really most sincerely dead. You can also prevent it from starting at the next reboot with this command: chkconfig httpd off

8.2.2 Installation Methods

Should you get a binary installation or source? A binary installation is usually quicker, while a source installation is more flexible and current. I'll look at both, but emphasize source, since security updates usually should not wait.

8.2.2.1 RPM installation

Of the many Linux package managers, RPM may be the most familiar, so I'll use it for this example. Grab the most current stable version of Apache from http://www.apache.org, your favorite Linux distribution, or any RPM site. Here's an example of obtaining and installing an older Apache RPM from Red Hat's site:

```
# wget ftp://ftp.redhat.com/pub/redhat/linux/updates/7.0/en/\

os/i386/apache-1.3.22-1.7.1.i386.rpm

# rpm -Uvh apache-1.3.22-1.7.1.i386.rpm
```

Depending on whose RPM package you use, Apache's files and directories will be installed in different places. This command prints where the package's files will be installed:

```
rpm -qpil apache-1.3.22-1.7.1.i386.rpm
```

We'll soon see how to make Apache's file hierarchy more secure, no matter what it looks like.

8.2.2.2 Source installation

Get the latest stable tarball:

```
# wget http://www.apache.org/dist/httpd/apache_1.3.24.tar.gz
```

```
# tar xvzf apache_1.3.24.tar.gz

# cd apache_1.3.24
```

If the file has an MD5 or GPG signature, check it (with *md5sum* or *gpgv*) to ensure you don't have a bogus distribution or a corrupted download file.

Then, run the GNU `configure` script. A bare: # ./configure

will install everything in directories under `/usr/local/apache` (Apache 2 uses `/usr/local/apache2`). To use another directory, use `--prefix`:

```
# ./configure --prefix=/usr/other/apache
```

Apache includes some standard *layouts* (directory hierarchies). To see these and other script options, enter the following:

```
# ./configure --help
```

Next, run good old `make`:

```
# make
```

This will print pages of results, eventually creating a copy of Apache called `httpd` in the `src` subdirectory. We'll look at what's actually there in the next section. When you're ready to install Apache to the target directory, enter the following: # make install

8.2.2.3 Linking methods

Does the preceding method produce a statically linked or dynamically linked executable? What modules are included? By including fewer modules, you use less memory and have fewer potential problems. "Simplify, simplify," say Thoreau, the least privilege principle, and the Web Server Diet Council.

Dynamic linking provides more flexibility and a smaller memory footprint. Your copy of Apache is dynamically linked if you see something like this:

```
# httpd -l

Compiled-in modules:

  http_core.c

  mod_so.c
```

Dynamically linked versions of Apache are easy to extend with some configuration options and an Apache restart. Recompilation is not needed. I prefer this method, especially when using the Perl or PHP modules. See http://httpd.apache.org/docs/dso.html for details on these Dynamic Shared Objects (*DSO*s).

A *statically linked* Apache puts the modules into one binary file, and it looks something like this:

```
# httpd -l

Compiled-in modules:

  http_core.c

  mod_env.c
```

```
  mod_log_config.c
  mod_mime.c
  mod_negotiation.c
  mod_status.c
  mod_include.c
  mod_autoindex.c
  mod_dir.c
  mod_cgi.c
  mod_asis.c
  mod_imap.c
  mod_actions.c
  mod_userdir.c
  mod_alias.c
  mod_access.c
  mod_auth.c
  mod_setenvif.c
suexec: disabled; invalid wrapper /usr/local/apache/bin/suexec
```

Specify `--activate-module` and `--add-module` to modify the module list. Changing any of the modules requires recompilation and relinking.

8.2.3 Securing Apache's File Hierarchy

Wherever your installation scattered Apache's files, it's time to make sure they're secure at runtime. Loose ownership and permission settings are a common cause of security problems.

We want the following:

- A user ID and group ID for Apache to use
- User IDs for people who will provide content to the server

Least privilege suggests we create an Apache user ID with as little power as possible. You will often see use of user ID `nobody` and group ID `nobody`. However, these IDs are also used by NFS, so it's better to use dedicated IDs. Red Hat uses user ID `apache` and group ID `apache`. The `apache` user has no shell and few permissions just the kind of guy we want, and the one we'll use here.

There are different philosophies on how to assign permissions for web user IDs. Here are some solutions for content files (HTML and such):

- Add each person who will be modifying content on the web site to the group `apache`. Make sure that others in the group (including the user ID `apache`) can read but not write one another's files (run `umask 137; chmod 640` for each content file and directory). These settings allow developers to edit their own files and let others in the group view them. The web server (running as user `apache`) can read and serve them. Other users on the web server can't access the files at all. This is important because scripts may contain passwords and other sensitive data. The `apache` user can't overwrite files, which is also useful in case of a lapse.

- The previous settings may be too extreme if you need to let web developers overwrite each other's files. In this case, consider mode `660`. This is a little less secure because now the `apache` user can also overwrite content files.

- A common approach (especially for those who recommend user ID `nobody` and group ID `nobody`) is to use the *other* permissions for the apache user (mode `644`). I think this is less safe, since it also gives read access to other accounts on the server.

Table 8-2 lists the main types of files in an Apache distribution, where they end up in a default RPM installation or a source installation, and recommended ownership and permissions.

Table 8-2. File locations for apache installations

File types	Notable files	Red Hat RPM directories	Source directories	Owner/modes
Initialization script	*Httpd*	*/etc/init.d*	(no standard)	Should be owned by `root`, with directory mode 755 and file mode 755
Configuration files	*httpd.confaccess.confsrm.conf*	*/etc/httpd/conf*	*/usr/local/apache/conf*	Should be owned by `root`, with directory mode 755 and file mode 644
Logs	*access_logerror_log*	*/etc/httpd/logs*	*/usr/local/apache/logs*	Should be owned by `root`, with directory mode 755 and file mode 644

Apache programs	*httpdapachectl*	*/usr/sbin*	*/usr/local/apache/bin*	Should be owned by `root`, with directory mode 755 and file mode 511
Apache utilities	*htpasswdapxsrotatelogs*	*/usr/sbin*	*/usr/local/apache/bin*	Should be owned by `root`, with directory mode 755 and file mode 755
Modules	*mod_perl.so*	*/usr/lib/apache*	*/usr/local/apache/libexec*	Should be owned by `root`, with directory mode 755 and file mode 755
CGI programs	(CGI scripts)	*/var/www/cgi-bin*	*/usr/local/apache/cgi-bin*	Directory should be owned by user `root` with mode 755; files should be owned by users in group `apache`, with mode 750
Static content	(HTML files)	*/var/www/html*	*/usr/local/apache/htdocs*	Directories should be owned by user `apache` with mode 470; files should be owned by users in group `apache`, with mode 640
Password/data files	(Varies)	(No standard)	(No standard)	Directories should be owned by user `apache` with mode 470; files should be owned by users in group

				apache, with mode 640

```
# apachectl configtest

# apachectl start

httpd -L
```

User apache

Group apache

ServerRoot /usr/local/apache

DocumentRoot /usr/local/apache/htdocs

Listen 127.0.0.1

Listen 81

Listen 202.203.204.205:82

<Directory />

Options none

AllowOverride none

Order deny,allow

Deny from all

</Directory>

<Directory /usr/local/apache/htdocs>

Deny from all

Allow from 127.0.0.1

</Directory>

Add Indexes to current options:

Options +Indexes

Remove Indexes from current options:

Options Indexes

Make Indexes the only current option, disabling the others:

Options Indexes

MaxClients

MaxRequestsPerChild

KeepAlive

on

MaxKeepAliveRequests

KeepAliveTimeout

RLimitCPU

<tt>soft</tt>,[<tt>max</tt>]

RLimitMEM

<tt>soft</tt>,[<tt>max</tt>]

RLimitNPROC

<tt>soft</tt>,[<tt>max</tt>]

LimitRequestBody

LimitRequestFields

LimitRequestFieldSize

LimitRequestLine

UserDir disabled

UserDir disabled

UserDir enabled good_user_1, careful_user_2

UserDir enabled

UserDir disabled root

UserDir /home/*/public_html

<Directory /home/*/public_html>

AllowOverride None

</Directory>

<Location /<tt>ssi_dir</tt>>

Options IncludesNoExec

</Location>

AddType application/x-server-parsed .shtml

AddHandler server-parsed .shtml

AddHandler server-parsed .htm

<Location /<tt>ssi_dir</tt>>

Options +IncludesNoExec

XBitHack full

</Location>

chmod +x changeling.html

<!--#include virtual="header.html"-->

. . . variable content goes here . . .

<!--#include virtual="footer.html"-->

<!--#include virtual="included_file.html"-->

<!--#include virtual="/cgi-bin/script"-->

<!--#exec cmd="ls -l /"-->

<!--#exec cmd="mail evil@weasel.org < /etc/passwd"-->

<Location /ssi_dir>

Options IncludesNoExec

</Location>

ScriptAlias /cgi-bin /usr/local/apache/cgi-bin

<Location /usr/local/apache/mixed>

Options ExecCGI

</Location>

Mixing static files and scripts is dangerous, since a configuration typo could cause Apache to treat a script file as a normal file and allow users to view its contents. If you do mix files and scripts, you need to tell Apache which files are CGI scripts and which are static files. Use a file suffix or some other naming convention for the script. We'll see how to protect files shortly.

Don't put a script interpreter program in a CGI directory. For instance, don't put the binary for Perl or a standalone PHP in `/usr/local/apache/cgi-bin`. This lets anyone run them without restrictions. CGI scripts should be as simple and focused as possible.

Expect trouble if users can upload files to a directory and execute them as CGI scripts. Consider using suEXEC (described next) or limiting CGI scripts to directories where you can see them.

8.3.5.3 suEXEC

Normally, CGI programs will all be run with Apache's user ID and group. If you have multiple users and virtual hosts, this lets them run each other's scripts and access each other's data. What if you have a web-hosting service and want to let your customers run their own CGI scripts, but no one else's? That's a job for Apache's `suEXEC` facility.

`suEXEC` is a setuid root program that wraps scripts to run with a specified user ID and group ID. Scripts need to pass a number of security guidelines before they will be accepted. As with any setuid root program, beware of

potential dangers from any exploit or botched configuration. Documentation is at http://httpd.apache.org/docs-2.0/suexec.html.

8.3.5.4 FastCGI

`FastCGI` is an alternative for creating CGI programs without the startup time of a standalone program, but also without the complexity of an Apache module. The protocol is language independent, and libraries are available for the most common web languages. Details are available at www.fastcgi.com.

`FastCGI` falls somewhere between standalone and module-based CGI. It starts an external CGI program, but maintains a persistent connection through the Apache module `mod_fastcgi`.

Scripts need slight modification to work with `FastCGI`. You must have set `Options ExecCGI` in *httpd.conf* to enable a `FastCGI` application, just as you would any other CGI program. If you want to allow use of `suEXEC` with `FastCGI`, set `FastCGIWrapper On`. `FastCGI` scripts are vulnerable to the same problems as any CGI scripts.

<tt>Method URI HTTP-Version</tt> \r\n

$ telnet www.apache.org 80

Trying 63.251.56.142...

Connected to daedalus.apache.org (63.251.56.142).

Escape character is '^]'.

HEAD / HTTP/1.1

Host: www.apache.org

HTTP/1.1 200 OK

Date: Sat, 13 Apr 2002 03:48:58 GMT

Server: Apache/2.0.35 (Unix)

Cache-Control: max-age=86400

Expires: Sun, 14 Apr 2002 03:48:58 GMT

Accept-Ranges: bytes

Content-Length: 7790

Content-Type: text/html

Connection closed by foreign host.

$

ServerTokens ProductOnly

$ telnet www.apache.org 80

Trying 63.251.56.142...

Connected to daedalus.apache.org (63.251.56.142).

Escape character is '^]'.

OPTIONS * HTTP/1.1

Host: www.apache.org

HTTP/1.1 200 OK

Date: Sat, 13 Apr 2002 03:57:10 GMT

Server: Apache/2.0.35 (Unix)

Cache-Control: max-age=86400

Expires: Sun, 14 Apr 2002 03:57:10 GMT

Allow: GET,HEAD,POST,OPTIONS,TRACE

Content-Length: 0

Content-Type: text/plain

Connection closed by foreign host.

$

http://www.hackenbush.com/

http://www.hackenbush.com/cgi-bin/groucho.pl?
day=jan%2006&user=zeppo

ScriptAlias /fakedir/ "/usr/local/apache/real_cgi_bin/"

http://www.hackenbush.com/fakedir/whyaduck/day/jan%2006/user/zeppo

#!/bin/sh

echo "Content-type: text/html"

```
echo

echo "Hello, world"
<?php ... ?>

<? ... ?>

<% ... %>
<? echo "<b>string<b> = <I>$string</I>\n"; ?>
<b>string</b> = <i><?=$string?></i>
display_errors = off
php_admin_flag display_errors off
php_value <tt><i>name value</i></tt>
php_flag <tt><i>name</i></tt> on|off
php_admin_value <tt><i>name value</i></tt>
php_admin_flag <tt><i>name</i></tt> on|off
ini_set("display_errors", "0");
#!/usr/bin/perl -w

use strict;
```

```
use CGI qw(:standard);

my $string = param("string");
echo header;

echo "<b>string</b> = <I>$string</I>\n";
<form name="user_form" method="post" action="/cgi-bin/echo">
<input type="text" name="string">
<input type="submit" value="submit">
</form>
<? echo "string = $string\n"; ?>
#!/usr/bin/perl -w

use strict;

use CGI qw(:standard);

print header;
```

print "string = ", param("string"), "\n";

string = quack

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

<script>

<embed>

<object>

<applet>

$HTTP_ENV_VARS

$_ENV

$HTTP_GET_VARS

$_GET

$HTTP_POST_VARS

$_POST

$HTTP_POST_FILES

$_FILES

$HTTP_COOKIE_VARS

$_COOKIE

$HTTP_SERVER_VARS

$_SERVER

#!/usr/bin/perl -wT

```
use strict;

use CGI qw(:standard);

my $user = param("user");
if ($user =~ /^(\w+)$/) { $user = $1; }
<FilesMatch ~ /\.inc$/>
order allow, deny
```

deny from all

</Files>

<FilesMatch ~ /(~,\.bak)$/>

order allow, deny

deny from all

</Files>

../../../etc/passwd

include_path .:/usr/local/lib/php:/usr/local/my_php_lib

<? include_once "db_login.inc"; ?>

<VirtualHost 192.168.102.103>

ServerName a.test.com

DocumentRoot /usr/local/apache/hosts/a.test.com

php_admin_value open_basedir /usr/local/apache/hosts/a.test.com

</VirtualHost>

register_globals

off

off

safe_mode

off

on

safe_mode_exec_dir

/usr/local/apache/<tt>host</tt>/cgi

open_basedir

/usr/local/apache/<tt>host</tt>/files

display_errors

on

off

log_errors

off

on

allow_url_fopen

on

off

session.save_path

/tmp

/usr/local/apache/sessions

$safer_input = escapeshellarg($input);

system("<tt>some_command</tt> $safer_input");

escapeshellcmd("<tt>some_command</tt> $input");

<form

method="post"

enctype="multipart/form-data"

action="/cgi-bin/process_form.php">

<input type="text" name="photo_name">

<input type="file" name="upload">

<input type="submit" value="submit">

</form>

<?

```
// $name is the original file name from the client

$name = $_FILES['photo_file']['name'];

// $type is PHP's guess of the MIME type

$type = $_FILES['photo_file']['type'];

// $size is the size of the uploaded file (in bytes)

$size = $_FILES['photo_file']['size'];

// $tmpn is the name of the temporary uploaded file on the server

$tmpn = $_FILES['photo_file']['tmp_name'];

// If everything looks right, move the temporary file

// to its desired place.


if (is_uploaded_file($tmpn))

   move_uploaded_file($tmpn, "/usr/local/apache/<tt><i>host</i>
</tt>/photos");

#!/usr/bin/perl -wT


use strict;


use CGI qw(:standard);


my $handle = param("photo_file");
```

```
my $tmp_file_name = tmpFileName($handle);

# Copy the file somewhere, or rename it

# ...

mysql -u root

GRANT SELECT ON sample_table

TO "sample_user@sample_machine"


IDENTIFIED BY "sample password"

<?


$link = mysql_connect("db.test.com", "dbuser", "dbpassword");

if (!$link)


   echo "Error: could not connect to database\n";

?>

// my_connect.inc


// PHP database connection function.


// Put this file outside the document root!
```

```
// Makes connection to database.

// Returns link id if successful, false if not.

function my_connect( )

{

$database = "db.test.com";
$user = "db_user";

$password = "db_password";
$link = mysql_connect($database, $user, $password);
return $link;

}
// client.php

// PHP client example.
```

```
// Include path is specified in include_path in php.ini.

// You can also specify a full pathname.

include_once "my_connect.inc";
$link = my_connect( );

// Do error checking in client or library function
if (!$link)

   echo "Error: could not connect to database\n";
// ...
$sql = "SELECT COUNT(*) FROM users WHERE\n" .

   "user = '$user' AND\n".

   "password = '$password'";
$result = mysql_query($sql);
if ($result && $row = mysql_fetch_array($result) && $row[0] == 1)
```

```
return true;
```

```
else

    return false;
' OR " = "
' OR " = "
SELECT COUNT(*) FROM users WHERE

user = " OR " = " AND

password = " OR " = "
# my_connect.pl

sub my_connect

{

my $server = "db.test.com";
my $db = "db_name";

my $user = "db_user";
```

```
my $password = "db_password";

my $dbh = DBI->connect(


   "DBI:mysql:$db:$server",

   $user

   $password,

   { PrintError => 1, RaiseError => 1 })

   or die "Could not connect to database $db.\n";

return $dbh;



}



1;
```

```
require "/usr/local/myperllib/my_connect.pl";
```

If your connection logic is more complex, it could be written as a Perl package or a module.

Taint mode won't protect you from entering tainted data into database queries. You'll need to check the data yourself. Perl's outstanding regular-expression support lets you specify patterns that input data must match before going into a SQL statement.

8.4.8 Checking Other Scripts

Once you've secured Apache and your own scripts, don't forget to check any other old scripts that may be lying around. Some demo scripts and even commercial software have significant holes. I suggest disabling or removing any CGI scripts if you aren't certain about them.

`whisker` (http://www.wiretrip.net/rfp/p/doc.asp/i2/d21.htm) is a Perl script that checks for buggy CGI scripts against a vulnerability database.

8.4.9 Continuing Care

Check your `error_log` regularly for bad links, attacks, or other signs of trouble. You are sure to see many IIS-specific exploit attempts such as Code Red and Nimda, but someone might actually be targeting a LAMP component.

<Location /auth_demo_dir> AuthName "My Authorization"

AuthType Basic

Note: Keep the password files in their own directory AuthUserFile /usr/local/apache/auth_dir/auth_demo_password "

Order deny, allow

Require valid-user

</Location>

require valid-user

require user <tt>user1</tt> <tt>user2</tt> ...

require group <tt>group1 group2</tt> ...

htpasswd -c /usr/local/apache/auth_dir/auth_demo_password

htpasswd /usr/local/apache/auth_dir/auth_demo_password -u raoul ... (prompt for password for raoul) ...

Client form

<form method="get" action="https://a.test.com/auth.php"> User: <input type="text" name="user"> Password: <input type="password" name="password"> <input type="submit">

</form>

<?

// a.test.com/auth.php

$time_arg = Date( );

$secret_string = "babaloo";

```
$hash_arg = md5($time_arg . $secret_string); $url =
"http://b.test.com/login.php" .

   "?" .

   "t=" . urlencode($time_arg) .

   "&h=" . urlencode($hash_arg); header("Location: $url");

?>

<?

// b.test.com/login.php

// Get the CGI variables:

$time_arg = $_GET['t'];

$hash_arg = $_GET['h'];

// Servers A and B both know the secret string, // the variable(s) it is
combined with, and their // order:

$secret_string = "babaloo";

$hash_calc = md5($time_arg . $secret_string); if ($hash_calc ==
$hash_arg)

   {

   // Check $time_arg against the current time.

   // If it's too old, this input may have come from a // bookmarked URL, or
may be a replay attack; reject it.

   // If it's recent and the strings match, proceed with the login...

   }
```

```
else
    {
    // Otherwise, reject with some error message.
    }
?>
<Location />
order deny, allow
deny from .evil.com
```

allow from all

</Location>

deny from evil.com

SetEnvIf Referer "^<tt>www.hackenbush.com</tt>" local
<Location /image_dir>

order deny,allow

deny from all

allow from env=local

</Location>

<Location />

AllowOverride false

</Location>

<Location />

order deny, allow

deny from all

Here's the required domain:

allow from .good.com

Any user in the password file: require valid-user

This does an "or" instead of an "and": satisfy any

</Location>

enable_trans_sid on

<a href="sample_link.html?<?=SID?>">link

link

<a href="sample_link.html?
PHPSESSID=379d65e3921501cc79df7d02cfbc24c3">link

in php.ini:

session.save_path=/usr/local/apache/sessions # or in apache's httpd.conf:

php_admin_valuesession.save_path /usr/local/apache/sessions

chmod 700 /usr/local/apache/sessions

./configure --with-apxs=/usr/local/apache/bin/apxs

Loadmodule dav_module libexec/libdav.so Addmodule mod_dav.c

htpasswd -s /usr/local/apache/passwords/dav.htpasswd <tt>user
password</tt>

The directory part of this must be writeable # by the user ID running
apache: DAVLockDB /usr/local/apache/davlock/

```
DAVMinTimeout 600

# Use a Location or Directory for each DAV area.

# Here, let's try "/DAV":

<Location /DAV>

# Authentication:

AuthName "DAV"

AuthUserFile /usr/local/apache/passwords/dav.htpasswd"
```

AuthType Basic

Some extra protection

AllowOverride None

Allow file listing

Options indexes

Don't forget this one!:

DAV On

Let anyone read, but

require authentication to do anything dangerous: <LimitExcept GET HEAD OPTIONS> require valid-user

</Limit>

</Location>

User-agent: *

Disallow: /image_dir

Disallow: /cgi-bin

BrowserMatch <tt>^evil_robot_name</tt> begone <Location />

order allow,deny

```
allow from all

deny from env=begone

</Location>
```

An evil robot may lie about its identity in the `UserAgent` HTTP request header and then make a beeline to the directories it's supposed to ignore. You can craft your *robots.txt* file to lure it into a tarpit, which is described in the next section.

8.5.8 Detecting and Deflecting Attackers

The more attackers know about you, the more vulnerable you are. Some use *port 80 fingerprinting* to determine what kind of server you're running. They can also pass a HEAD request to your web server to get its version number, modules, etc.

Script kiddies are not known for their precision, so they will often fling IIS attacks such as Code Red and Nimda at your Apache server. Look at your `error_log` to see how often these turn up. You can exclude them from your logs with Apache configuration tricks. A more active approach is to send email to the administrator of the offending site, using a script like NimdaNotifyer (see http://www.digitalcon.ca/nimda/). You may even decide to exclude these visitors from your site. Visit http://www.snort.org to see how to integrate an IP blocker with their intrusion detector.

The harried but defiant administrator might enjoy building a *tarpit*. This is a way to turn your network's unused IP addresses into a TCP-connection black hole. Attempts to connect to these addresses instead connect with something that will not let go. See http://www.hackbusters.net/LaBrea/ for details of a tarpit implementation.

8.5.9 Caches, Proxies, and Load Balancers

A proxy is a man in the middle. A caching proxy is a man in the middle with a memory. All the security issues of email apply to web pages as they stream about: they can be read, copied, forged, stolen, etc. The usual answer is to apply end-to-end cryptography.

If you use sessions that are linked to a specific server (stored in temporary files or shared memory rather than a database), you must somehow get every request with the same session ID directed to the same server. Some load balancers offer *session affinity* to do this. Without it, you'll need to store the sessions in some shared medium, like an NFS-mounted filesystem or a database.

8.5.10 Logging

The Apache log directories should be owned by *root* and visible to no one else. Logs can reveal sensitive information in the URLs (GET parameters) and in the referrer. Also, an attacker with write access can plant cross-site scripting bugs that would be triggered by a log analyzer as it processes the URLs.

Logs also grow like crazy and fill up the disk. One of the more common ways to clobber a web server is to fill up the disk with log files. Use *logrotate* and *cron* to rotate them daily.

8.6 Other Servers and Web Security

I'll finish the chapter with some brief notes about other servers used with or instead of Apache.

8.6.1 Web Servers

Apache has the largest market share, but it isn't the only web server available for Linux. An organization that is more comfortable with commercial software might consider an Apache derivative like Covalent or an independent product like Zeus or iPlanet.

There are also some interesting open source alternatives. `tux` is a new open source web and FTP server, developed by Ingo Molnar and others at Red Hat. It takes advantage of improvements in recent (2.4+) Linux kernels to provide an extremely fast server. (It set some benchmark records for SPECWeb99 as much as three times faster than Apache or IIS on the same hardware). `tux` can operate in user and kernel space, serving static and dynamic content, with optional caching. It can work in front of Apache or behind it, so you can assign tasks to the appropriate server. The frontend server serves port 80, and the back-end server serves port 8080 or another unused value. Usually, `tux` serves static content and passes everything else to Apache.

`tux` is still quite new, and little is yet known of any specific security issues. The `tux` manual details the checks it makes before serving a file:

> TUX only serves a file if:
>
> The URL does not contain ?.
>
> The URL does not start with /.
>
> The URL points to a file that exists.
>
> The file is world-readable.
>
> The file is not a directory.
>
> The file is not executable.
>
> The file does not have the sticky-bit set.
>
> The URL does not contain any forbidden substrings such as ..

`simplefile` is a read-only HTTP and FTP server by Daniel Bernstein, the author of `djbdns` and `qmail`. It serves only static files. If your site has static pages and stringent security requirements, it may be easier to install and configure this server than to close all the doors in Apache.

`aolserver`, `wn`, and `xitami` are other open source contenders.

8.6.2 Application Servers

A mini-industry has sprouted up in the territory between web servers and databases. *Application servers* provide connection pooling and other services. Oracle touted its servers as "unbreakable" until buffer overflows and other flaws were found. Generally, anything that increases the surface area of web services also increases the complexity, security risks, and maintenance costs. It isn't clear that there is a proportional gain in performance or uptime.

Chapter 9. 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 RCP, SCP, and rsync. These, unlike FTP, can be encrypted with the help of Secure Shell or Stunnel, covered in Chapter 4 and Chapter 5,

respectively. (Chapter 4 also covers SFTP, an FTP-like frontend for the Secure Shell.)

/var/ftp:

total 12

d--x--x--x 2 root root 4096 Apr 16 00:19 bin dr--r--r-- 2 root root 4096 Apr 16 00:27 etc drwxr-xr-x 2 root wheel 4096 Apr 16 06:56 pub /var/ftp/bin:

total 44

---x--x--x 1 root root 43740 Apr 16 00:19 ls /var/ftp/etc:

total 12

-r--r--r-- 1 root root 63 Apr 16 00:26 group -r--r--r-- 1 root root 1262 Apr 16 00:19 localtime -r--r--r-- 1 root root 106 Apr 16 00:27 passwd /var/ftp/pub:

total 1216

-rw-r--r-- 1 root root 713756 Apr 16 06:56 hijinks.tar.gz -rw-r--r-- 1 root root 512540 Apr 16 06:56 hoohaw.tar.gz -rw-r--r-- 1 root root 568 Apr 16 06:43 welcome.msg

drwxr-xr-x 2 root root 4096 Apr 16 00:06 ftp

[root@myron proftpd-1.2.4]# ./configure --with-modules=mod_readme:mod_quota

ftp:x:14:50:FTP User:/home/ftp:/bin/true

ftp:x:50:

[root@myron etc]# useradd -g ftp -s /bin/true ftp

[root@myron etc]# useradd -g ftp -s /bin/true -d /var/ftp ftp

/home:

drwxrwxr-x 2 root wheel 4096 Apr 21 16:56 ftp /home/ftp:

total 12

-rwxrwx-wx 1 root wheel 145 Apr 21 16:48 incoming -rwxrwxr-x 1 root wheel 145 Apr 21 16:48 pub -rw-rw-r-- 1 root wheel 145 Apr 21 16:48 welcome.msg /home/ftp/incoming:

total 0

/home/ftp/pub:

total 8

-rw-rw-r-- 1 root wheel 145 Apr 21 16:48 hotdish_recipe_no6132.txt -rw-rw-r-- 1 root wheel 1235 Apr 21 16:48 pretty_good_stuff.tgz

Base Settings:

ServerType standalone

MaxInstances 30

TimeoutIdle 300

TimeoutNoTransfer 300

TimeoutStalled 300

```
UseReverseDNS no

LogFormat uploadz "%t %u\@*l \"%r\" %s %b bytes"

SyslogFacility LOCAL5

# Base-server settings (which can also be defined in <VirtualHost> blocks):
ServerName "FTP at Polkatistas.org"

Port 21

MasqueradeAddress firewall.polkatistas.org <Limit LOGIN>

   DenyAll

</Limit>

# Global Settings: shared by base server AND virtual servers <Global>

   ServerIdent off AllowRetrieveRestart on MaxClients 20 "Sorry, all lines
are busy (%m users max)."

   MaxClientsPerHost 1 "Sorry, your system is already connected."

   Umask 022

   User nobody Group nogroup </Global>

# Anonymous configuration, uploads permitted to "incoming"

<Anonymous ~ftp>

   User ftp

   Group ftp

   UserAlias anonymous ftp MaxClients 30
```

```
    DisplayLogin welcome.msg ExtendedLog /var/log/ftp_uploads WRITE
uploadz AllowFilter "^[a-zA-Z0-9 ,.+/_\-]*$"

    <Limit LOGIN> AllowAll

    </Limit> <Limit WRITE> DenyAll

    </Limit> <Directory incoming/*> <Limit READ DIRS CWD> DenyAll

    </Limit> <Limit STOR> AllowAll

    </Limit> </Directory> </Anonymous>

<VirtualHost 55.44.33.23>

    Port 21

    <Limit LOGIN> DenyAll

    </Limit> <Anonymous /home/ftp_hohner> User ftp

    Group ftp

    UserAlias anonymous ftp MaxClients 30

    DisplayLogin welcome_hohner.msg AllowFilter "^[a-zA-Z0-9 ,]*$"

    <Limit LOGIN> AllowAll

    </Limit> <Limit WRITE> DenyAll

    </Limit> </Anonymous> </VirtualHost>

ifconfig ifacename:n alias

ifconfig eth0:0 55.44.33.23
```

You can add such a command to your */etc/init.d/network* startup script to make the IP alias persistent across reboots. Alternatively, your Linux

distribution may let you create IP aliases in its network-configuration utility or GUI.

yodeldiva@near:~ > rsync -vv -e ssh ./thegoods.tgz far:~

opening connection using ssh -l yodeldiva far rsync --server -vv . "~"

yodeldiva@far's password: **********

expand file_list to 4000 bytes, did move

thegoods.tgz

total: matches=678 tag_hits=801 false_alarms=0 data=11879

wrote 14680 bytes read 4206 bytes 7554.40 bytes/sec

total size is 486479 speedup is 25.76

[[<tt>username</tt>@]<tt>hostname</tt>:]/<tt>path</tt>/<tt>to</tt>/<tt>filename</tt>

"global-only" options

syslog facility = local5

global options which may also be defined in modules

use chroot = yes

```
uid = nobody

gid = nobody

max connections = 20

timeout = 600

read only = yes

# a module:

[public]

    path = /home/public_rsync

    comment = Nobody home but us tarballs

    hosts allow = near.echo-echo-echo.org, 10.18.3.12

    ignore nonreadable = yes

    refuse options = checksum
```

```
    dont compress = *
[incoming]

    path = /home/incoming
    comment = You can put, but you can't take
    read only = no

    ignore nonreadable = yes
    transfer logging = yes
[audiofreakz]

    path = /home/cvs
    comment = Audiofreakz CVS repository (requires authentication)
    list = no

    auth users = watt, bell
    secrets file = /etc/rsyncd.secrets
watt:shyneePAT3

bell:d1ngplunkB00M!
```

[root@near darthelm]# rsync far::

public Nobody home but us tarballs

incoming You can put, but you can't take

[yodeldiva@near ~]# rsync far::public/newstuff.tgz .

[yodeldiva@near ~]# rsync -e ssh far:/home/public_rsync/newstuff.tgz .

These two aren't exactly equivalent, of course, because whereas the rsync daemon process on *far* is configured to serve files in this directory to anonymous users (i.e., without authentication), SSH always requires authentication (although this can be automated using null-passphrase RSA or DSA keys, described in Chapter 4). But it does show the difference between how paths are handled.

What About Recursion?

I've alluded to rsync's usefulness for copying large bodies of data, such as software archives and CVS trees, but all my examples in this chapter show single files being copied. This is because my main priority is showing how to configure and use rsync securely.

I leave it to you to explore the many client-side (command-line) options rsync supports, as fully documented in the *rsync(8)* manpage. Particularly noteworthy are *-a* (or *archive*), which is actually shorthand for *-rlptgoD* and which specifies recursion of most file types (including devices and symbolic links); and also *-C* (or *cvs-exclude*), which tells rsync to use CVS-style file-exclusion criteria in deciding which files not to copy.

9.2.2.5 Tunneling rsync with Stunnel

The last rsync usage I'll mention is the combination of rsync, running in daemon mode, with Stunnel. Stunnel is a general-purpose TLS or SSL wrapper that can be used to encapsulate any simple TCP transaction in an encrypted and optionally X.509-certificate-authenticated session. Although rsync gains encryption when you run it in SSH mode, it loses its daemon features, most notably anonymous rsync. Using Stunnel gives you encryption as good as SSH's, while still supporting anonymous transactions.

Stunnel is covered in depth in Chapter 5, using rsync in most examples. Suffice it to say that this method involves the following steps on the server side:

1. Configure *rsyncd.conf* as you normally would.

2. Invoke *rsync* with the *port* flag, specifying some port *other* than 873 (e.g., *rsync daemon port=8730*).

3. Set up an Stunnel listener on TCP port 873 to forward all incoming connections on TCP 873 to the local TCP port specified in the previous step.

4. If you don't want anybody to connect "in the clear," configure *hosts.allow* to block nonlocal connections to the port specified in Step 2. In addition or instead, you can configure iptables to do the same thing.

On the client side, the procedure is as follows:

1. As *root*, set up an Stunnel listener on TCP port 873 (assuming you don't have an rsync server on the local system already using it), which forwards all incoming connections on TCP 873 to TCP port 873 on the remote server.

2. When you wish to connect to the remote server, specify *localhost* as the remote server's name. The local *stunnel* process will now open a connection to the server and forward your rsync packets to the remote

stunnel process, and the remote *stunnel* process will decrypt your rsync packets and deliver them to the remote rsync daemon. Reply packets, naturally, will be sent back through the same encrypted connection.

As you can see, rsync itself isn't configured much differently in this scenario than anonymous rsync: most of the work is in setting up Stunnel forwarders.

9.3 Resources

1. Bernstein, D. J. "PASV Security and PORT Security." Online article at http://cr.yp.to/ftp/security.html.
2. http://cr.yp.to/publicfile.html. (15 April 2002) (The home of publicfile, D. J. Bernstein's secure FTP/HTTP server. Like djbdns, it uses Bernstein's daemontools and ucspi-tcp packages.)
3. Carnegie Mellon University (CERT Coordination Center). "Anonymous FTP Abuses." Online article at http://www.cert.org/tech_tips/anonymous_ftp_abuses.html (15 April 2002).
4. Carnegie Mellon University (CERT Coordination Center). "Anonymous FTP Configuration Guidelines." Online article at http://www.cert.org/tech_tips/anonymous_ftp_config.html (15 April 2002).
5. Carnegie Mellon University (CERT Coordination Center). "Problems with the FTP PORT Command or Why You Don't Want Just Any PORT in a Storm." Online article at http://www.cert.org/tech_tips/ftp_port_attacks.html (15 April 2002).
6. Garfinkel, Simson and Gene Spafford. *Practical Unix and* Internet Security, Sebastopol, CA:

O'Reilly & Associates, 1996.

7. 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.

8. http://www.proftpd.org. (The

   official ProFTPD home page.)

9. http://rsync.samba.org. (The

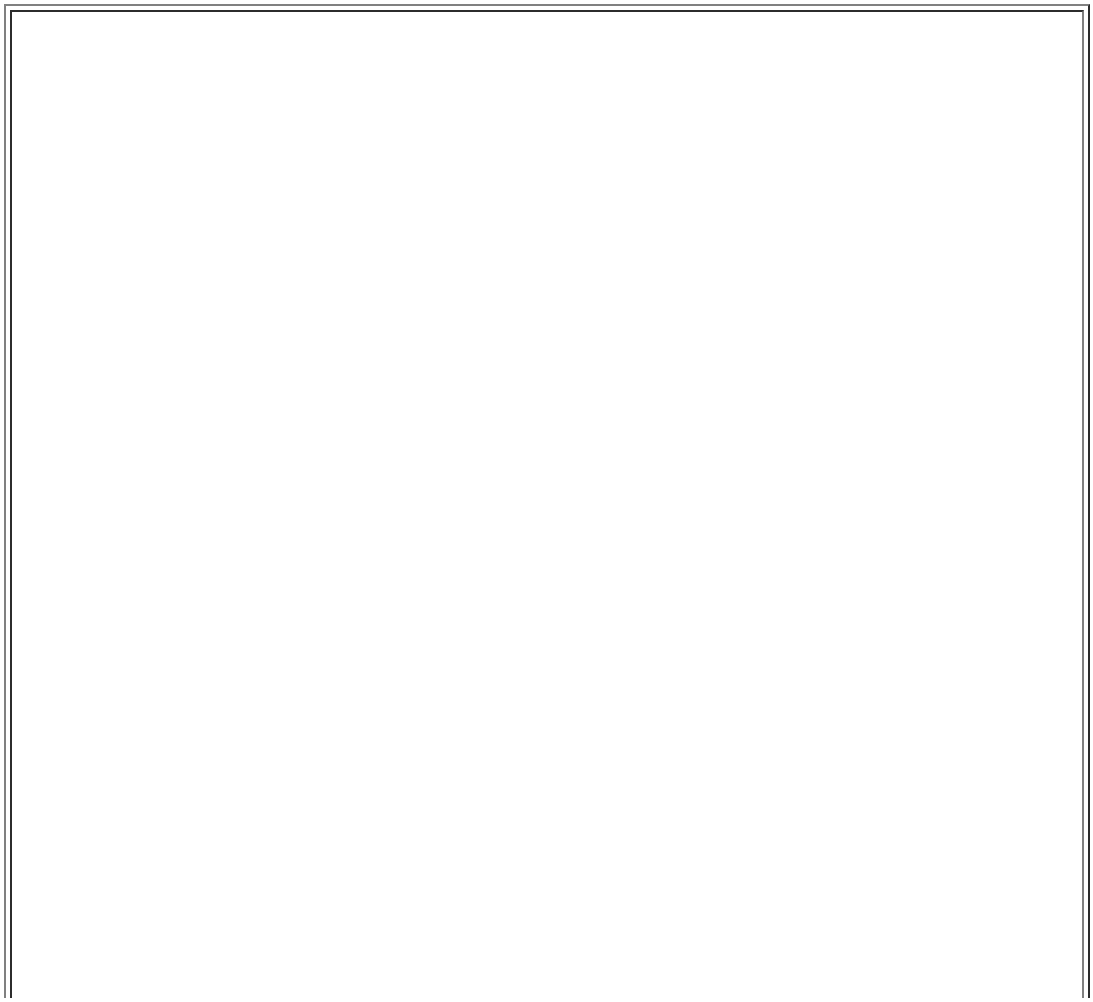
   official rsync home page.)

Chapter 10. 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 us troubleshoot virtually 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.



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 log file. 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.

mail.notice /var/log/mail

*.emerg @mothership.mydomain.org

mail,uucp.notice /var/log/mail

arp -s 192.168.192.168 03:03:03:31:33:77

mail,uucp.notice;uucp.!=alert /var/log/mail

mail,uucp.notice;uucp.!alert /var/log/mail

Sample syslog.conf file that sorts messages by mail, kernel, and "other,"

and broadcasts emergencies to all logged-in users

print most sys. events to tty10 and to the xconsole pipe, and emergencies to everyone

kern.warn;*.err;authpriv.none |/dev/xconsole

*.emerg *

send mail, news (most), & kernel/firewall msgs to their respective logfiles

mail.* -/var/log/mail

kern.* -/var/log/kernel_n_firewall

save the rest in one file

.;mail.none -/var/log/messages

-r

mark.* -/var/log/messages

mylinuxbox:/etc/init.d# ./syslogd -m 30

daemon syslogd -m 0 -a /var/named/dev/log

syslogd -a /var/named/dev/log -a /var/otherchroot/dev/log -a /additional/dev/log

mylinuxbox:/etc/init.d# ./syslogd -f ./syslog.conf.test

We've already covered use of the *-r* flag, which tells syslogd to accept log messages from remote hosts, but we haven't talked about the security ramifications of this. On the one hand, security is clearly enhanced when you use a centralized log server or do anything else that makes it easier for you to manage and monitor your logs.

On the other hand, you must take different threat models into account. Are your logs sensitive? If log messages traverse untrusted networks and if the inner workings of the servers that send those messages are best kept secret, then the risks may outweigh the benefit (at least, the specific benefit of syslogd's unauthenticated clear-text remote logging mechanism).

If this is the case for you, skip to this chapter's section on Syslog-ng. Syslog-ng can send remote messages via the TCP protocol and can therefore be used in conjunction with *stunnel*, *ssh*, and other tools that greatly enhance its security. Since syslog uses only the connectionless UDP protocol for remote logging and therefore can't "tunnel" its messages though *stunnel* or *ssh*, syslog is inherently less securable than Syslog-ng.

If your log messages aren't sensitive (at least the ones you send to a remote logger), then there's still the problem of Denial of Service and message forgery attacks. If you invoke *syslogd* with the *-r* flag, it will accept all remote messages without performing *any checks whatsoever* on the validity

of the messages themselves or on their senders. Again, this risk is most effectively mitigated by using Syslog-ng.

But one tool you can use with syslog to partially mitigate the risk of invalid remote messages is TCPwrappers. Specifically, TCPwrappers' "hosts access" authentication mechanism provides a simple means of defining which hosts may connect and via which protocols they may connect to your log server. Hosts-access authentication is easily tricked by source-IP-spoofing (especially since syslog transactions are strictly one way), but it's better than nothing, and it's probably sufficient to prevent mischievous but lazy attackers from interfering with syslog.

If you're willing to bet that it is, obtain and install TCPwrappers and refer to its *hosts_access(5)* manpage for details. Note that despite its name, TCPwrappers' hosts access can be used to control UDP-based applications.

cd <tt>packagename</tt>

./configure

./make

./make install

mylinuxbox:/usr/src/libol-0.2.23# ./configure --prefix=/your/dir/here

-d

-v

-f<tt> filename</tt>

-V

-p <tt>pidfilename</tt>

Simple syslog-ng.conf file.

options {

use_fqdn(no);

sync(0);

};

source s_sys { unix-stream("/dev/log"); internal( ); };

source s_net { udp( ); };

destination d_security { file("/var/log/security"); };

destination d_messages { file("/var/log/messages"); };

destination d_console { usertty("root"); };

filter f_authpriv { facility(auth, authpriv); };

filter f_messages { level(info .. emerg)

and not facility(auth, authpriv); };

filter f_emergency { level(emerg); };

log { source(s_sys); filter(f_authpriv); destination(d_security); };

log { source(s_sys); filter(f_messages); destination(d_messages); };

log { source(s_sys); filter(f_emergency); destination(d_console); };

s

chain_hostnames(<tt>yes </tt>|<tt> no</tt>)

sskeep_hostname(<tt>yes </tt>|<tt> no</tt>)

ssuse_fqdn(<tt>yes </tt>|<tt> no</tt>)

ssuse_dns(<tt>yes </tt>|<tt> no</tt>)

ssuse_time_recvd(<tt>yes </tt>|<tt> no</tt>)

sstime_reopen(<tt>NUMBER</tt>)

sstime_reap(<tt>NUMBER</tt>)

sslog_fifo_size(

<tt>NUMBER</tt>)^{[1]}

sssync(<tt>NUMBER</tt>)1

ssowner(<tt>string</tt>)1

ssgroup(<tt>string</tt>)1

ssperm(<tt>NUMBER</tt>)1

sscreate_dirs(<tt>yes </tt>|<tt> no</tt>)1

ssdir_owner(<tt>string</tt>)1

ssdir_group(<tt>string</tt>)1

ssdir_perm(<tt>NUMBER</tt>)1

options { use_dns(yes); };

Sep 13 19:56:56 s_sys@10.9.8.7 sshd[13037]: Accepted publickey for ROOT from

10.9.8.254 port 1355 ssh2

Sep 13 19:56:56 s_sys@joebob sshd[13037]: Accepted publickey for ROOT from

10.9.8.254 port 1355 ssh2

Original log entry on host1:

Sep 19 22:57:16 s_loc@linux syslog-ng[1656]: syslog-ng version 1.4.13 starting

Entry as sent to and recorded by host2:

Sep 19 22:57:16 s_loc@linux/host1 syslog-ng[1656]: syslog-ng version 1.4.13 starting

Same log entry as relayed from host2 to host3:

Sep 19 22:57:16 s_loc@linux/host1/host2 syslog-ng[1656]: syslog-ng version 1.4.13 starting

options { };

source s_loc { unix-stream("/dev/log"); internal( ); };

destination d_host2 { udp("host2" port(514)); };

destination d_local { file("/var/log/messages"); };

log { source(s_loc); source(s_net); destination(d_host2); destination(d_local); };

source <tt>sourcelabel</tt> { <tt>driver1</tt>(<tt>[options]</tt>); <tt>driver2</tt>(<tt>[options]</tt>); <tt>etc.</tt> };

source s_loc { unix-stream("/dev/log"); internal( ); };

internal( )

file("<tt>filename</tt>" <tt>[options]</tt>)

pipe("<tt>filename</tt>"<tt> </tt>)

unix_stream("<tt>filename</tt>"<tt> [options]</tt>)

unix_dgram("<tt>filename</tt>"<tt> [options]</tt>)

tcp(<tt>[</tt>ip(<tt>address</tt>)<tt>]</tt> <tt>[</tt>port(<tt>#</tt>)<tt>]</tt>

<tt>[</tt>max-connections(<tt>#</tt>)<tt>] </tt>)

udp(<tt>[</tt>ip(<tt>address</tt>)<tt>]</tt> <tt>[</tt>port(<tt>#</tt>)<tt>] </tt>)

source s_loc { unix-dgram("/dev/log"); internal( ); };

source s_tcpmessages { tcp(ip(192.168.190.190) port(10514)); };

source s_udpmessages { udp( ); };

source s_tcpmessages { tcp(ip(192.168.190.190) port(10514) max-connections(100)); };

file("<tt>filename[$MACROS]</tt>")

tcp("<tt>address</tt>" <tt>[</tt>port(<tt>#</tt>); <tt>] </tt>)

udp("<tt>address</tt>" <tt>[</tt>port(<tt>#</tt>); <tt>] </tt>)

pipe("<tt>pipename</tt>")

unix_stream("<tt>filename</tt>"<tt> [options]</tt>)

unix_dgram("<tt>filename</tt>"<tt> [options]</tt>)

usertty(<tt>username </tt>)

program("<tt>/path/to/program</tt>")

destination d_dailylog { file("/var/log/messages.$WEEKDAY"); };

PROGRAM

HOST

FACILITY

PRIORITY <tt>or</tt> LEVEL <tt>(synonyms)</tt>

YEAR

MONTH

DAY

WEEKDAY

HOUR

MIN

SEC

destination d_mylog { file("/var/log/ngfiles/mylog" create_dirs(yes) dir_owner(root) \

dir_group(root) dir_perm(0700)); };

destination d_micklog { file("/var/log/micklog" owner(mick) group(wheel) perm(0640)); };

facility(<tt>facility-name</tt>)

priority(<tt>priority-name</tt>)

priority(<tt>priority-name1</tt>,

<tt>priority-name2</tt>, <tt>etc.</tt>)

priority(<tt>priority-name1</tt> ..

<tt>priority-name2</tt>)

level(<tt>priority-name</tt>)

program(<tt>program-name</tt>)

host(<tt>hostname</tt>)

match(<tt>regular-expression</tt>)

filter(<tt>filter-name</tt>)

filter f_mail { facility(mail); };

```
filter f_debug { not facility(auth, authpriv, news, mail); };

filter f_messages { level(info .. warn) and not facility(auth, authpriv, cron,
daemon,

mail, news); };


filter f_cother { level(debug, info, notice, warn) or facility(daemon, mail);
};

source s_loc { unix-stream("/dev/log"); internal( ); };

source s_tcpmessages { tcp( ip(192.168.190.190); port(10514);); };

destination d_dailylog { file("/var/log/messages.$WEEKDAY"); };

destination d_micklog { file("/var/log/micklog" owner(mick) perm(0600));
};

filter f_mail { facility(mail); };


filter f_messages { level(info .. warn) and not facility(auth, authpriv, cron,
daemon,

mail, news); };


log { source(s_tcpmessages); destination(d_micklog); };

log { source(s_loc); filter(f_mail); destination(d_micklog); };

log { source(s_loc); filter(f_messages); destination(d_dailylog); };

log { source(s_loc); filter(DEFAULT); destination(d_dailylog); };
```

```
# WARNING: while this syslog-ng.conf file is syntactically correct and complete, it is

# intended for illustrative purposes only entire categories of message

# are ignored!


source s_local { unix_stream("dev/log"); internal( ); };

filter f_denials { match("[Dd]enied|[Ff]ail"); };


destination d_mailtomick { program("/usr/local/sbin/mailtomick.sh"); };

log { source(s_local); filter(f_denials); destination(d_mailtomick); };

#!/bin/bash


# mailtomick.sh


# Script which listens for standard input and emails each line to mick
```

#

while read line;

do

echo $line | mail -s "Weirdness on that Linux box" mick@pinheads-on-ice.net

done

The most important lines in Example 10-20 are the filter *f_denials* and the destination *d_mailtomick*. The filter uses a *match( )* directive containing a regular expression that matches the strings "denied," "Denied," "Fail," and "fail."[3] The destination *d_mailtomick* sends messages via a *program( )* declaration to the standard input of a script I wrote called */usr/local/sbin/mailtomick.sh*.

[3] If you're completely new to regular expressions, I highly recommend *Mastering Regular Expressions* by Jeffrey E. F. Friedl (O'Reilly).

Before we go further in the analysis, here's an important caveat: *program( )* opens the specified program once and leaves it open until syslog-ng is stopped or restarted. Keep this in mind when deciding whether to use *pipe( )* or *program( )* (i.e., *pipe( )* doesn't do this), and in choosing what sort of applications you invoke with *program( )*.

In some cases, keeping a script open (actually a *bash* process) is a waste of resources and even a security risk (if you run *syslog-ng* as *root*). Furthermore, the particular use of email in Examples 10-19 and 10-20 introduces the possibility of Denial of Service attacks (e.g., filling up the system administrator's mailbox). But under the right circumstances, such as on a non-Internet-accessible host that has a few CPU cycles to spare, this is a legitimate use of Syslog-ng.

The script itself, */usr/local/sbin/mailtomick.sh*, simply reads lines from the standard input and emails each line to *mick@pinheads-on-ice.net*. Since *syslog-ng* needs to keep this script open, the *read* command is contained in an endless loop. This script will run until the *syslog-ng* process that invoked it is restarted or killed.

In the interest of focusing on the most typical uses of Syslog-ng, I've listed some *syslog-ng.conf* options without giving examples of their usage and omitted a couple of other options altogether. Suffice it to say that the global/file option *log_fifo_size( )* and the global options *time_reap( )*, *time_reopen( )*, *gc_idle_threshold( )*, and *gc_busy_threshold( )* are useful for tuning *syslog-ng*'s performance to fit your particular environment.

The official (maintained) documentation for Syslog-ng is the *Syslog-ng Reference Manual*. PostScript, SGML, HTML, and ASCII text versions of this document are included in the */doc* directory of Syslog-ng's source-code distribution.

For advanced or otherwise unaddressed issues, the best source of Syslog-ng information is the Syslog-ng mailing list and its archives. See http://lists.balabit.hu/mailman/listinfo/syslog-ng for subscription information and archives.

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

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

#!/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.

```
# Very simple logrotate.conf file
# Global options: rotate logs monthly, saving four old copies and sending
# error-messages to root. After "rotating out" a file, touch a new one
monthly

rotate 4

errors root

create

# Keep an eye on /var/log/messages
/var/log/messages {

    size 200k
    create
    postrotate
    /bin/kill -HUP `cat /var/run/syslog-ng.pid 2> /dev/null` 2> /dev/null ||
true
    endscript
```

}

<tt>/path/to/logfile</tt> {

<tt> directive1</tt>

<tt> directive2</tt>

<tt> etc.</tt>

}

compress

copytruncate

create <tt>[octalmode owner group]</tt>

ifempty | notifempty

include <tt>file_or_directory</tt>

missingok | nomissingok

olddir <tt>dir</tt> | noolddir

postrotate

<tt>line1</tt>

<tt> line2</tt>

<tt> etc.</tt>

endscript

prerotate

<tt>line1</tt>

<tt> line2</tt>

<tt> etc.</tt>

endscript

/usr/sbin/logrotate /etc/logrotate.conf

/etc/logfiles - This file tells cron.daily, which log files have to be watched

#

File max size mode ownership service

(reload if changed)

/var/log/mgetty.* +1024k 644 root.root

/var/log/messages +4096k 640 root.root

/var/log/httpd/access_log +4096k 644 root.root apache

/var/squid/logs/access.log +4096k 640 squid.root

In the first noncomment line, all log files whose name begins */var/log/mgetty* will be rotated after exceeding 1,024 kilobytes, after which they'll be rotated to new files whose permissions are `-rw-r--r--` and that are owned by user *root* and group *root*.

The third line states that the file */var/log/httpd/access_log* should be rotated after exceeding 4,096 kilobytes, should be recreated with permissions `-rw-r--r--`, owned by user *root* and group *root*, and after rotation is done, the startup script */etc/init.d/apache* should be restarted.

Since the maximum age of all log files is set globally in */etc/rc.config*, take care not to set the maximum size of a frequently written-to file (such as */var/log/messages*) too high. If this happens and if the maximum age is high enough, your logs may fill their volume.

Speaking of which, I highly recommend the use of a dedicated */var* partition on any machine that acts as a server; a full */var* partition is much less likely to cause disruptive system behavior (e.g., crashing) than a full root partition.

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


Checking for Time::HiRes 1.12 ... ok

Checking for Date::Calc ... ok

Checking for Date::Format ... ok

Checking for File::Tail ... ok

Checking if your kit is complete...
```

Looks good

Writing Makefile for swatch

[root@barrelofun swatch-3.0.1]#

```
make

   make test
   make install
   make realclean
perl -MCPAN -e "install Example::Module"
perl -MCPAN -e "install Example::PreRequisite"
perl ./Makefile.PL
```

make

make test

make install

watchfor /File name too long/

    mail addresses=mick\@visi.com,subject=BufferOverflow_attempt

echo=<tt>normal, underscore, blue,

inverse, etc.</tt>

bell <tt>N</tt>

exec <tt>command</tt>

pipe <tt>command</tt>

throttle <tt>HH:MM:SS</tt>

watchfor /wuzza.html/

    echo=red

    bell 2

/reject|failed/

/reject/i

/file system full/

    echo=red

    mail addresses=mick\@visi.com,subject=Volume_Full,when=7-1:1-24

mylinuxbox:~# swatch -c /var/log/.swatchrc.access --script-dir=/var/log &

mylinuxbox:~# swatch -c /var/log/.swatchrc.access --script-dir=/var/log

\ -t /var/log/apache/access_log &

swatch generally doesn't clean up after itself very well; it tends to leave watcher-process scripts behind. Keep an eye out and periodically delete these in your home directory or in the script directories you tend to specify with *script-dir*.

Again, if you want swatch to monitor multiple files, you'll need to run swatch multiple times, with at least a different tailing target (*-t* value) specified each time and probably a different configuration file for each as well.

10.5.5 Fine-Tuning swatch

Once swatch is configured and running, we must turn our attention to the Goldilocks Goal: we want swatch to be running neither "too hot" (alerting us about routine or trivial events) nor "too cold" (never alerting us about anything). But what constitutes "just right?" There are as many answers to this question as there are uses for Unix.

Anyhow, you don't need me to tell you what constitutes nuisance-level reporting: if it happens, you'll know it. You may even experience a scare or two in responding to events that set off alarms appropriately but turn out to be harmless nonetheless. Read the manual, tweak *.swatchrc*, and stay the course.

The other scenario, in which too little is watched for, is much harder to address, especially for the beginning system administrator. By definition, anomalous events don't happen very frequently, so how do you anticipate

how they'll manifest themselves in the logs? My first bit of advice is to get in the habit of browsing your system logs often enough to get a feel for what the routine operation of your systems looks like.

Better still, "tail" the logs in real time. If you enter the command *tail -f /var/log/messages*, the last 50 lines of the system log will be printed, plus *all subsequent lines, as they're generated*, until you kill tail with a *Control-c*. This works for any file, even a log file that changes very rapidly.

Another good thing you can do is to "beat up on" (probe/attack) your system in one virtual console or xterm while tailing various log files in another. nmap and Nessus, which are covered in Chapter 3 (Hardening Linux), are perfect for this.

By now you may be saying, "Hey, I thought the whole reason I installed swatch was so I wouldn't have to watch log files manually!" Wrong. Swatch minimizes, but does not eliminate, the need for us to parse log files.

Were you able to quit using your arithmetic skills after you got your first pocket calculator? No. For that matter, can you use a calculator in the first place unless you already know how to add, multiply, etc.? Definitely not. The same goes for log file parsing: you can't tell swatch to look for things you can't identify yourself, no more than you can ask for directions to a town whose name you've forgotten.

10.5.6 Why You Shouldn't Configure swatch Once and Forget About It

In the same vein, I urge you to not be complacent about swatch silence. If swatch's actions don't fire very often, it could be that your system isn't getting probed or misused very much, but it's at least as likely that swatch isn't casting its net wide enough. Continue to periodically scan through your logs manually to see if you're missing anything, and continue to tweak *.swatchrc*.

Don't forget to periodically reconsider the auditing/logging configurations of the daemons that generate log messages in the first place. Swatch won't catch events that aren't logged at all. Refer to the *syslogd(8)* manpage for

general instructions on managing your *syslogd* daemon, and the manpages of the various things that log to syslog for specific instructions on changing the way they log events.

10.6 Resources

1. http://www.stanford.edu/~atkins/swatch.

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

2. http://www.stanford.edu/~atkins/swatch/lisa93.html.

   Hansen, Stephen and Todd Atkins, creators of swatch.

   "Centralized System Monitoring with Swatch." (Old, but still useful.)

3. http://www.enteract.com/~lspitz/swatch.html.

   Spitzner, Lance. "Watching Your Logs." (A brief introduction to swatch.)

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

Chapter 11. Simple Intrusion Detection Techniques

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.

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 Section 11.5 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.

11.1 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.

11.1.1 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

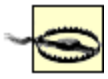
http://www.chkrootkit.org/ for a script that can be used to detect installed

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.

Another thing to keep in mind with integrity checkers is that they are *not proactive* (unless one or more of your perimeter systems is a honeypot a "sacrificial lamb" that will set 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've only got 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.

However, if you wish to do everything possible to detect attacks before they succeed, you'll also need to deploy something more sophisticated i.e., something *in addition to* integrity checking systems, which truly are your last line of defense.

11.1.2 NIDS: Scanning for Signatures Versus Anomalies

Whereas host-based IDSes tend to be of a single type (integrity checkers), Network IDSes come in two main flavors: those that rely on *attack signatures*

(network traffic patterns characteristic of specific attacks) and NIDS that are intelligent enough to detect potential attacks based on variances from some concept of *normal network state*. Commonly used NIDSes rely most heavily on

signature scanning, but many also possess some degree of anomaly detection functionality as well.

There are other types of network-based systems besides signature scanners and anomaly detectors. Most of these other types fall into what Marcus Ranum calls the "Audit Based" category, in which as much data as possible is logged but is not analyzed until well *after* the events in question have transpired. In a holistic sense, this is a very powerful method, as it implies the ability to construct highly locale-specific signatures for very subtle and complicated attacks.

The payoff of an Audit Based IDS, however, comes only after the system has witnessed complete attacks, which, in most settings, is too late. Audit Based systems are thus beyond the scope of this chapter, due to these practical limitations: we're most concerned with detecting (and perhaps even preventing) attacks, and much less so with studying them after the fact.

11.1.2.1 Signature-based systems

Signature-based systems are the most common type of network-based IDS, for several reasons. First, they're the simplest: they compare network transactions to known attack signatures, and if a given transaction sufficiently resembles a known attack, the IDS logs an alert (and possibly sends it to someone's pager, too). Second, they're low maintenance: all you generally need to do is keep the signature database current. Third, they tend to register a relatively small percentage of *false positives*, an attribute highly prized by system administrators (who usually receive plenty of email and pager alerts as it is!).

Signature-based systems, which are also called "Misuse Detectors" in

Ranum's lexicon, are a successful and practical approach to network-based intrusion detection. However, they have one important limitation: by relying on signatures of known attacks, they're of little use against new attacks and variations on known attacks that are sufficiently different so as to not match existing signatures. It's worth

considering that most attack signatures are written after someone *has already fallen victim* to that attack.

11.1.2.2 Anomaly-detection systems

Anomaly-detection systems, which I also sometimes call *state-based systems*, are much less widely used. First, they tend to be complex: determining what constitutes "normal" traffic

on a given network is a nontrivial task for humans, so it follows that a high degree of artificial intelligence (AI) is required for any automated system that does this. (Maybe your experience is different from mine, but clueful human network engineers are rare enough; why would robotic ones be any less so?) Second, they're high maintenance: even when coded with good AI and sophisticated statistical modeling mechanisms, state-based IDSes typically require a lengthy and sometimes difficult "initialization" period, during

which they collect enough network data to create a statistically meaningful profile of normal network states. The system requires frequent (and endless) fine-tuning afterwards.

Third, even after all this work, anomaly-detection systems tend to register many more false positives than

signature-based systems do (though presumably, this problem diminishes over time). This can result in a great deal of inconvenience.

What About False Negatives?

In discussing *false positives* (alerts that aren't really caused by attacks) as an undesirable trait of IDSes, I'm making an important assumption: that *false negatives* (attacks that trigger *no* alert) aren't even an issue.

This is an important assumption.

We don't like false positives because they're annoying, inconvenient, and have the potential to distract our attention from alerts triggered by real attacks. But in configuring and fine-tuning any IDS, you must *always err on the side of false positives* when given the choice.

In many peoples' opinions, including Marcus Ranum's, anomaly-detection systems are the most promising approach for future IDS technologies. As noted earlier, signature-based systems are limited to *known attacks*, specifically those for which your IDS

has signatures. State-based anomaly detection is the only approach with the potential to detect both known and new types of attacks.

tar -xzvf ./tripwire-2.3.1-2.tar.gz.

make release

cp ./install/install.cfg .

cp ./install/install.sh .

sh ./install.sh

ROOT =/usr/sbin

POLFILE =/etc/tripwire/tw.pol

DBFILE =/var/lib/tripwire/$(HOSTNAME).twd REPORTFILE =/var/lib/tripwire/report/$(HOSTNAME)-$(DATE).twr SITEKEYFILE =/etc/tripwire/site.key LOCALKEYFILE =/etc/tripwire/squeezebox-local.key EDITOR =/bin/vi

LATEPROMPTING =false

LOOSEDIRECTORYCHECKING =false

MAILNOVIOLATIONS =true

EMAILREPORTLEVEL =3

REPORTLEVEL =3

MAILMETHOD =SMTP

SYSLOGREPORTING =false

SMTPHOST =mail.polkatistas.org

SMTPPORT =25

twadmin --create-cfgfile --site-keyfile ./site.key twcfg.txt

twadmin --print-cfgfile > myconfig.txt

twadmin --print-polfile > mypolicy.txt

twadmin --print-cfgfile

twadmin -m f

WEBROOT=/home/mick/www;

CGIBINS=/home/mick/www/cgi-bin;

TWPOL="/etc/tripwire";

TWDB="/var/lib/tripwire";

BINS = $(ReadOnly) ; # Binaries that should not change
DIR_SEMISTATIC = +tpug ; # Dir.s that shouldn't change
perms/ownership SIG_MED = 66 ; # Important but not system-critical files

Mick's Web Junk

(

rulename = "MickWeb", severity = $(SIG_MED), emailto =
mick@uselesswebjunk.com)

{

$(WEBROOT) -> $(ReadOnly) (recurse=1) ;
!$(WEBROOT)/guestbook.html ; $(CGIBINS) -> $(BINS) ; /var/log/httpd -
> $(Growing) ; /home/mick -> $(DIR_SEMISTATIC) (recurse=0) }

$(WEBROOT) -> $(ReadOnly) (recurse=1) ;

!$(WEBROOT)/guestbook.html ;

WEBROOT=/home/mick/www;

```
CGIBINS=/home/mick/www/cgi-bin;

TWPOL="/etc/tripwire";

TWDB="/var/lib/tripwire";

BINS = $(ReadOnly) ; # Binaries that should not change
DIR_SEMISTATIC = +tpug ; # Directories that shouldn't change
perms/ownership SIG_MED = 66 ; # Important but not system-critical files
# Mick's Web Junk

(

   rulename = "MickWeb", severity = $(SIG_MED), emailto =
mick@uselesswebjunk.com )

{

   $(TWPOL) -> $(Readonly) ; $(WEBROOT) -> $(ReadOnly) (recurse=1)
; !$(WEBROOT)/guestbook.html ; $(CGIBINS) -> $(BINS) ; /var/log/httpd
-> $(Growing) ; /home/mick -> $(DIR_SEMISTATIC) (recurse=0) }

+pinugtsdbmCM-rlacSH

+pinugtd-srlbamcCMSH

+pinugtdl-srbamcCMSH

+pugsdr-intlbamcCMSH

-pinugtsdrlbamcCMSH

+pinugtsdrbamcCMSH-l

/usr/sbin/siggen # tripwire binaries...

/usr/sbin/tripwire # ...

/usr/sbin/twadmin # ...
```

/usr/sbin/twprint # ...

/bin/ # all core system binaries

/sbin/ # all core admin. binaries /usr/bin/ # user binaries, especially: /usr/bin/at /usr/bin/awk /usr/bin/bzcat /usr/bin/bzgrep /usr/bin/bzip2 /usr/bin/crontab /usr/bin/csh /usr/bin/diff /usr/bin/dir /usr/bin/du /usr/bin/Emacs /usr/bin/expect /usr/bin/file /usr/bin/find /usr/bin/finger /usr/bin/flex /usr/bin/gawk /usr/bin/gdb /usr/bin/grep /usr/bin/gruff /usr/bin/gzip /usr/bin/ident /usr/bin/idle /usr/bin/less /usr/bin/lsof /usr/bin/nm /usr/bin/nroff /usr/bin/passwd /usr/bin/perl /usr/bin/pdksh /usr/bin/php /usr/bin/pico /usr/bin/quota /usr/bin/rexec /usr/bin/rlogin /usr/bin/ssh /usr/bin/strings /usr/bin/strip /usr/bin/sudo /usr/bin/swatch /usr/bin/sz /usr/bin/tail /usr/bin/tailf /usr/bin/tcsh /usr/bin/top /usr/bin/troff /usr/bin/up2date /usr/bin/users /usr/bin/vi /usr/bin/vim /usr/bin/which /usr/bin/yacc /usr/bin/zsh

/usr/libexec/ # some core system daemons /usr/sbin/ # superuser binaries, especially: /usr/sbin/anacron /usr/sbin/atd

/usr/sbin/chroot /usr/sbin/crond

/usr/sbin/httpd /usr/sbin/identd

/usr/sbin/in.fingerd /usr/sbin/in.rexecd /usr/sbin/in.rlogind /usr/sbin/in.rshd /usr/sbin/in.telnetd /usr/sbin/iptables /usr/sbin/lpd /usr/sbin/lsof

/usr/sbin/named /usr/sbin/ntpd

/usr/sbin/postfix /usr/sbin/pppd

/usr/sbin/rpc.rstatd /usr/sbin/safe_finger /usr/sbin/sendmail /usr/sbin/showmount /usr/sbin/smrsh /usr/sbin/snmpd

/usr/sbin/snmptrapd /usr/sbin/squid /usr/sbin/sshd /usr/sbin/stunnel

/usr/sbin/suexec /usr/sbin/tcpd

/usr/sbin/tmpwatch /usr/sbin/visudo /usr/sbin/xinetd /usr/sbin/xinetd-ipv6

/usr/local/bin/ # local system binaries /usr/local/sbin/ # local superuser binaries /usr/local/libexec/ # some local system daemons /etc/ # system configuration files /var/log/ # system logs (use "Growing"

built-in property mask!) /lib/ # system libraries, especially: /lib/libc.so.6

/lib/modules/ # use recurse=0 -- this is large /lib/security/ # PAM lives here

/usr/lib/ # more libraries, especially: /usr/lib/libc.a

/usr/lib/libc.so

/usr/lib/libc_nonshared.a

/usr/local/lib/ # local apps' libraries

/dev/console -> $(Dynamic)-u ; # Dynamic, but UID can change

/dev/console -> +pingutd-srlbamcCMSH-u ; # Dynamic, but UID can change

twadmin --create-polfile policyfile.txt

twadmin --print-polfile > mypol.txt

tripwire --init

tripwire --check

twprint --print-report --report-level N --twrfile /path/file

tripwire --check --email-report

#!/bin/sh

HOST_NAME=`uname -n`

TWHOME = /var/lib/tripwire

if [! -e $TWHOME/${HOST_NAME}.twd] ; then echo "**** Error: Tripwire database for ${HOST_NAME} not found. **

**"

echo "**** Run "/etc/tripwire/twinstall.sh" and/or "tripwire --init". **

**"

else

test -f /etc/tripwire/tw.cfg && /usr/sbin/tripwire --check fi

test -f /etc/tripwire/tw.cfg && /usr/sbin/tripwire --check --email-report --no-tty-output --silent

30 1,5,14 * * * /usr/sbin/tripwire -m c -M -n -s

tripwire --update --twrfile /path/to/report/myhost-date.twr

Remove the "x" from the adjacent box to prevent updating the database with the new values for this object.

Modified:

[x] "/home/mick/www"

tripwire --check --interactive

tripwire --check tripwire --update --twrfile /path/to/reportname.twr

twadmin --print-polfile > mypolicy.txt # dump current installed policy vi mypolicy.txt # make changes to policy ...

tripwire --update-policy mypolicy.txt # install the updated policy

When you use the -- *update-policy* directive, Tripwire will parse the specified policy text file, generate a new database, and compare all records that the old and new databases have in common. If those records match, Tripwire will encrypt, sign, and install your new policy and apply the corresponding changes to its database.

If, however, any of the common records don't match, Tripwire will not update the policy or the database. You'll need to run a Tripwire check, followed by a database update (now is the perfect time to use *tripwire* -- *check* -- *interactive*) and then run the policy update again.

A Tip from Ron Forrester

Here's a Tripwire tip from Ron Forrester, Tripwire Open Source Project Manager:

> I always leave a violation or two (say /etc/sendmail.st) in this makes it more difficult for an intruder to forge a report it is quite easy to forge a report with no violations, but add a known violation or two, and it gets much more difficult.

I think this is excellent advice. The whole point of using Tripwire is because you acknowledge the possibility that a host may be compromised; you therefore need to take what measures you can to protect the burglar alarm from the burglars. Intentionally leaving or even creating a violation or two (e.g., by adding an extra comment line to a Tripwire-protected file in /etc) is a simple way to do so.

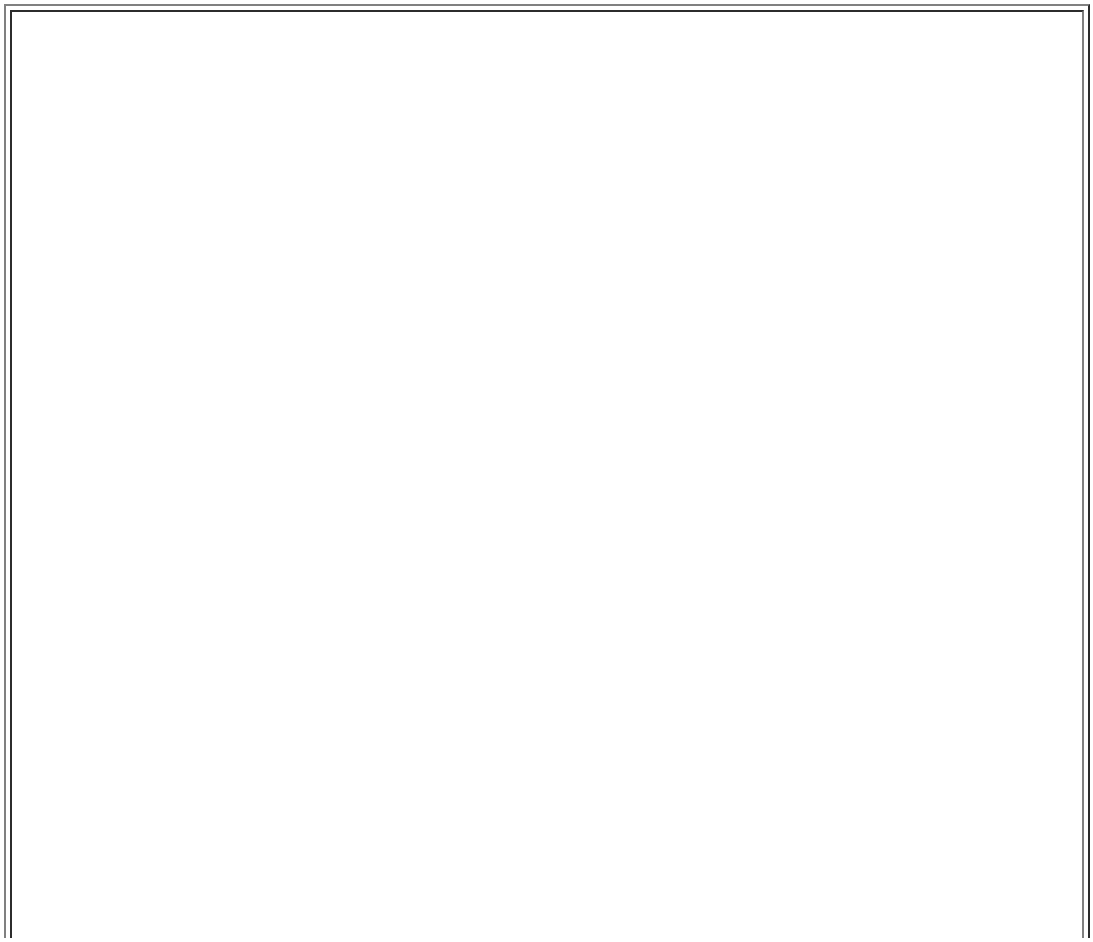
11.3 Other Integrity Checkers

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. As of this writing its version number is 0.8, 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, including me (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*, 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).

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 like 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 down side, 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.

Accordingly, 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 run checks more often, this might be a sensible tradeoff.

If, on the other hand, the system in question has a large number of local users (i.e., shell accounts), I strongly recommend against it; such users may be able to learn a lot about the system without triggering a violation. The weak hash-check method, insofar as it's ever justifiable, is only good against external attackers.

By the way, running an integrity checker very frequently is *not* likely to help you catch an attacker "in the act." This is for the

simple reason that there is an inevitable lag between the time an integrity checker sends a report and the time when someone actually gets around to

reading and responding to it. Rather, the practical value of frequent checks lies in the fact that the more frequently your checker writes reports, the more granularity with which you'll be able to analyze a successful attack after the fact, which may improve your ability to recover from it.

Of the three tools I've covered here, Tripwire is the most mature but also the most encumbered from a software-license perspective. AIDE is completely free, and it has some additional functionality, but is much less mature than Tripwire. Fcheck is fast, free, highly portable, and simple, but also makes some notable tradeoffs at security's expense.

bash-# ./configure

bash-# make && make install

bash-# ./configure --with-mysql

bash-# make && make install

bash-# echo "CREATE DATABASE snort;" | mysql -u snortsql -p

Enter password: <ENTER>

bash-# cd /usr/src/snort-1.8.4

bash-# mysql snort < ./contrib/create_mysql

bash-# snort -dvi eth0

Log directory = /var/log/snort

Initializing Network Interface eth0

--== Initializing Snort ==--

Checking PID path...

PATH_VARRUN is set to /var/run/ on this operating system

PID stat checked out ok, PID set to /var/run/

Writing PID file to "/var/run/"

Decoding Ethernet on interface eth0

--== Initialization Complete ==--

-*> Snort! <*-

Version 1.8.3 (Build 88)

By Martin Roesch (roesch@sourcefire.com, www.snort.org)

03/22-22:25:26.041707 192.168.100.20:1052 -> 10.10.117.13:80

TCP TTL:63 TOS:0x10 ID:10528 IpLen:20 DgmLen:60 DF

******S* Seq: 0x8651A4AB Ack: 0x0 Win: 0x16D0 TcpLen: 40

TCP Options (5) => MSS: 1460 SackOK TS: 1805707 0 NOP WS: 0

=+
=+=+=+=+=+=+=+=+=+=+

03/22-22:25:26.046576 10.10.117.13:80 -> 192.168.100.20:1052

TCP TTL:64 TOS:0x0 ID:33016 IpLen:20 DgmLen:60 DF

***A**S* Seq: 0x6D4A1B04 Ack: 0x8651A4AC Win: 0x7D78 TcpLen: 40

TCP Options (5) => MSS: 1460 SackOK TS: 63072524 1805707 NOP

TCP Options => WS: 0

=+
=+=+=+=+=+=+=+=+=+=+

03/22-22:25:26.047354 192.168.100.20:1052 -> 10.10.117.13:80

TCP TTL:63 TOS:0x10 ID:10529 IpLen:20 DgmLen:52 DF

A* Seq: 0x8651A4AC Ack: 0x6D4A1B05 Win: 0x16D0 TcpLen: 32

TCP Options (3) => NOP NOP TS: 1805707 63072524

=+

03/22-22:25:44.282136 192.168.100.20:1052 -> 10.10.117.13:80

TCP TTL:63 TOS:0x10 ID:10530 IpLen:20 DgmLen:95 DF

AP Seq: 0x8651A4AC Ack: 0x6D4A1B05 Win: 0x16D0 TcpLen: 32

TCP Options (3) => NOP NOP TS: 1807530 63072524

bash-# snort -dv host 192.168.100.200

bash-# snort -dv not port 22

bash-# snort -d -l ./snort/ -h <i>10.10.20.0/24 </i>

bash-# snort -l /var/log/snort/ -b

bash-# snort -dv -r /var/log/snort/snort-0324\@2146.log

output database: log, mysql, user=root dbname=snort host=localhost

include bad-traffic.rules

include /etc/snort/rules/bad-traffic.rules

include $RULE_PATH/bad-traffic.rules

bash-# snort -T -c /etc/snort/snort.conf

bash-# snort -Dd -z est -c /etc/snort/snort.conf

bash-# snort -b -A fast -c /etc/snort/snort.conf

[**] [100:2:1] spp_portscan: portscan status from 192.168.100.20: 7 connections acr

oss 1 hosts: TCP(7), UDP(0) [**]

03/25-23:05:21.524291

[**] [100:2:1] spp_portscan: portscan status from 192.168.100.20: 7 connections acr

oss 1 hosts: TCP(7), UDP(0) [**]

03/25-23:05:43.057380

[**] [100:2:1] spp_portscan: portscan status from 192.168.100.20: 7 connections acr

oss 1 hosts: TCP(7), UDP(0) [**]

03/25-23:05:53.635274

[**] [100:2:1] spp_portscan: portscan status from 192.168.100.20: 6 connections acr

oss 1 hosts: TCP(6), UDP(0) [**]

03/25-23:19:17.615096

[**] [100:3:1] spp_portscan: End of portscan from 192.168.100.20: TOTAL time(43s) h

osts(1) TCP(27) UDP(0) [**]

03/25-23:19:21.657371

Mar 25 23:05:46 192.168.100.20:60126 -> 10.10.117.13:751 SYN
******S*

Mar 25 23:05:53 192.168.100.20:60120 -> 10.10.117.13:310 SYN
******S*

Mar 25 23:05:53 192.168.100.20:60121 -> 10.10.117.13:323 SYN ******S*

Mar 25 23:05:53 192.168.100.20:60122 -> 10.10.117.13:41 SYN ******S*

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"MISC Large ICMP Packet"; dsi

ze: >800; reference:arachnids,246; classtype:bad-unknown; sid:499; rev:1;)

Any time this rule is triggered by a large ICMP packet, it logs the message "MISC Large ICMP Packet" to */var/snort/alert*. To receive notification from Swatch every time this rule fires, simply configure Swatch to watch */var/snort/alert* for the phrase "Large ICMP Packet."

In addition to Swatch monitoring Snort for specific events, it's also a good idea to set up a *cron*/*anacron* job in */etc/cron.daily* to email you a snapshot of part or all of */var/log/snort/alert*, or even just the bottom 50 lines or so. That way you'll not only receive real-time alerts of specific events from Snort; you'll also be regularly notified of activity Swatch doesn't catch.

11.4.4.7 Updating Snort's rules automatically

The last tip I'll offer on Snort use is a reminder that the Snort team refreshes the official collection of contributed and tested Snort rules every 30 minutes, 24 hours a day, 7 days a week. That doesn't mean the rules change that frequently; it means that every 30 minutes, the current rules in the Snort CVS tree are recopied to the Snort web site. Thus, any change that anyone on the Snort team makes to those rules at any time will be propagated to http://www.snort.org/dl/snapshots/ within 30 minutes.

Several people have written different scripts you can use to download and update Snort rules automatically on your own system. Many of these scripts target the attack database at Max Vision's arachNIDS project site and are therefore available there (http://www.whitehats.com/ids/).

Since the arachNIDS site has been unavailable at various times, you might also consider one alternative to arachNIDS-oriented scripts: Andreas ...stling's script Oinkmaster v0.2, available at http://www.algonet.se/~nitzer/oinkmaster/. This script automatically downloads the latest "official" rules from http://www.snort.org, filters out ones not relevant to your site, and updates your local rule set. It comes with documentation in the form of a README file and is written in Perl, so it's easy to customize and fine tune for your needs.

Note that the precise download path to the current Snort rules has changed since Oinkmaster's last update; you'll need to edit Oinkmaster to target http://www.snort.org/dl/snapshots/snortrules.tar.gz rather than *http://snort.sourcefire.com/downloads/snortrules.tar.gz*. This URL is set in Oinkmaster's *url* variable.

You probably don't need to schedule Oinkmaster (or whatever script you choose to use) every 30 minutes, but I recommend scheduling it to be run at least twice a day.

11.5 Resources

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

   Excellent introduction to the subject.

2. http://web.mit.edu/tytso/www/linux/ext2intro.html

   Card, Rémy, Theodore Ts'o, and Stephen Tweedie. "Design and Implementation of the Second Extended Filesystem."

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

3. 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.

4. 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.

5. http://sourceforge.net/projects/tripwire

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

6. 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).

7. http://www.tripwire.org

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

8. http://www.tripwire.com/downloads/tripwire_asr/index.cfml?

   Tripwire Academic Source Release download site.

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

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

10. http://www.cs.tut.fi/~rammer/aide.html

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

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

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

12. 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.

13. http://www.snort.org

Official Snort web site: source, binaries, documentation, discussion forums, and amusing graphics.

14. http://www.cert.org/kb/acid

The Analysis Console for Intrusion Databases (ACID) is a PHP application that analyzes IDS data in real time. ACID is a popular companion to Snort because it helps make sense of large Snort data sets; this is its official home page.

15. http://www.algonet.se/~nitzer/oinkmaster

Home of the Oinkmaster auto-Snort rules update script.

16. http://www.whitehats.com

Security news, tools, and the arachNIDS attack signature database (which can be used to update your SNORT rules automatically as new attacks are discovered).

17. http://www.lids.org

The Linux Intrusion Detection System (LIDS) web site. LIDS is a kernel patch and administrative tool that provides granular logging and access controls for processes and for the filesystem.

```
#! /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

Shell functions sourced from /etc/rc.status: # rc_check check and set local and overall rc status # rc_status check and set local and overall rc status # rc_status -v ditto but be verbose in local rc status # rc_status -v -r ditto and clear the local rc status # rc_failed set local and overall rc status to failed # rc_reset clear local rc status (overall remains) # rc_exit exit appropriate to overall rc status . /etc/rc.status

First reset status of this service rc_reset

Return values acc. to LSB for all commands but status: # 0 - success

1 - misc error

2 - invalid or excess args

3 - unimplemented feature (e.g. reload) # 4 - insufficient privilege

5 - program not installed

6 - program not configured

7 - program is not running

#

Note that starting an already running service, stopping # or restarting a not-running service as well as the restart # with force-reload (in case signalling is not supported) are # considered a success.

/End SuSE-specific stuff.

The rest of this script is non-SuSE specific case "$1" in

start)

echo -n "Loading Woofgang's Packet Filters"

SETUP -- stuff necessary for any bastion host # Load kernel modules first

(We like modprobe because it automatically checks for and loads any other # modules required by the specified module.) modprobe ip_tables

modprobe ip_conntrack_ftp

Flush active rules and custom tables $IPTABLES --flush

$IPTABLES --delete-chain

Set default-deny policies for all three default chains

$IPTABLES -P INPUT DROP

$IPTABLES -P FORWARD DROP

$IPTABLES -P OUTPUT DROP

Give free reign to the loopback interfaces, i.e. local processes may connect # to other processes' listening-ports.

$IPTABLES -A INPUT -i lo -j ACCEPT

$IPTABLES -A OUTPUT -o lo -j ACCEPT

Do some rudimentary anti-IP-spoofing drops. The rule of thumb is "drop # any source IP address which is impossible" (per RFC 1918)

#

$IPTABLES -A INPUT -s 255.0.0.0/8 -j LOG --log-prefix "Spoofed source IP"

$IPTABLES -A INPUT -s 255.0.0.0/8 -j DROP

$IPTABLES -A INPUT -s 0.0.0.0/8 -j LOG --log-prefix "Spoofed source IP"

$IPTABLES -A INPUT -s 0.0.0.0/8 -j DROP

$IPTABLES -A INPUT -s 127.0.0.0/8 -j LOG --log-prefix "Spoofed source IP"

$IPTABLES -A INPUT -s 127.0.0.0/8 -j DROP

$IPTABLES -A INPUT -s 192.168.0.0/16 -j LOG --log-prefix "Spoofed source IP"

$IPTABLES -A INPUT -s 192.168.0.0/16 -j DROP

$IPTABLES -A INPUT -s 172.16.0.0/12 -j LOG --log-prefix "Spoofed source IP"

$IPTABLES -A INPUT -s 172.16.0.0/12 -j DROP

$IPTABLES -A INPUT -s 10.0.0.0/8 -j LOG --log-prefix " Spoofed source IP"

$IPTABLES -A INPUT -s 10.0.0.0/8 -j DROP

The following will NOT interfere with local inter-process traffic, whose # packets have the source IP of the local loopback interface, e.g. 127.0.0.1

$IPTABLES -A INPUT -s $IP_LOCAL -j LOG --log-prefix "Spoofed source IP"

```
$IPTABLES -A INPUT -s $IP_LOCAL -j DROP

# Tell netfilter that all TCP sessions do indeed begin with SYN

# (There may be some RFC-non-compliant application somewhere which # begins its transactions otherwise, but if so I've never heard of it)
$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j LOG --log-prefix "Stealth scan attempt?"

$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j DROP

# Finally, the meat of our packet-filtering policy:
```

INBOUND POLICY

(Applies to packets entering our network interface from the network, # and addressed to this host)

Accept inbound packets that are part of previously-OK'ed sessions $IPTABLES -A INPUT -j ACCEPT -m state --state ESTABLISHED,RELATED

Accept inbound packets which initiate SSH sessions $IPTABLES -A INPUT -p tcp -j ACCEPT --dport 22 -m state --state NEW

Accept inbound packets which initiate FTP sessions $IPTABLES -A INPUT -p tcp -j ACCEPT --dport 21 -m state --state NEW

Accept inbound packets which initiate HTTP sessions $IPTABLES -A INPUT -p tcp -j ACCEPT --dport 80 -m state --state NEW

Log and drop anything not accepted above # (Obviously we want to log any packet that doesn't match any ACCEPT rule, for # both security and troubleshooting. Note that the final "DROP" rule is # redundant if the default policy is already DROP, but redundant security is # usually a good thing.)

#

$IPTABLES -A INPUT -j LOG --log-prefix "Dropped by default (INPUT):"

$IPTABLES -A INPUT -j DROP

OUTBOUND POLICY

(Applies to packets sent to the network interface (NOT loopback) # from local processes)

If it's part of an approved connection, let it out $IPTABLES -I OUTPUT 1 -m state --state RELATED,ESTABLISHED -j ACCEPT

Allow outbound ping

(For testing only! If someone compromises your system they may attempt to use ping to identify other active IP addresses on the DMZ. Comment this rule out when you don't need to use it yourself!)

#

$IPTABLES -A OUTPUT -p icmp -j ACCEPT --icmp-type echo-request
Allow outbound DNS queries, e.g. to resolve IPs in logs # (Many network applications break or radically slow down if they # can't use DNS. Although DNS queries usually use UDP 53, they may also use TCP

53. Although TCP 53 is normally used for zone-transfers, DNS queries with # replies greater than 512 bytes also use TCP 53, so we'll allow both TCP and UDP

53 here

#

$IPTABLES -A OUTPUT -p udp --dport 53 -m state --state NEW -j ACCEPT

$IPTABLES -A OUTPUT -p tcp --dport 53 -m state --state NEW -j ACCEPT

Log & drop anything not accepted above; if for no other reason, for troubleshooting

#

NOTE: you might consider setting your log-checker (e.g. Swatch) to # sound an alarm whenever this rule fires; unexpected outbound trans-

actions are often a sign of intruders!

#

$IPTABLES -A OUTPUT -j LOG --log-prefix "Dropped by default (OUTPUT):"

$IPTABLES -A OUTPUT -j DROP

Log & drop ALL incoming packets destined anywhere but here.

(We already set the default FORWARD policy to DROP. But this is # yet another free, reassuring redundancy, so why not throw it in?)

#

$IPTABLES -A FORWARD -j LOG --log-prefix "Attempted FORWARD? Dropped by default:"

$IPTABLES -A FORWARD -j DROP

;;

Unload filters and reset default policies to ACCEPT.

FOR LAB/SETUP/BENCH USE ONLY -- else use `stop'!!

Never run this script `wide_open' if the system is reachable from # the Internet!

```
#
wide_open)
echo -n "DANGER!! Unloading Woofgang's Packet Filters!!"
$IPTABLES --flush
```

$IPTABLES -P INPUT ACCEPT

$IPTABLES -P FORWARD ACCEPT

$IPTABLES -P OUTPUT ACCEPT

;;

stop)

echo -n "Portcullis rope CUT..."

Unload all fw rules, leaving default-drop policies $IPTABLES --flush

;;

status)

echo "Querying iptables status (via iptables --list)..."

$IPTABLES --line-numbers -v --list

;;

*)

echo "Usage: $0 {start|stop|wide_open|status}"

exit 1

;;

esac

#! /bin/sh

init.d/masterfw

#

System startup script for packet filters on a three-homed SuSE 7.1

Linux firewall (Internal network, DMZ network, External network).

#

IMPORTANT BACKGROUND ON THIS EXAMPLE: the internal network is numbered # 192.168.100.0/24; the DMZ network is 208.13.201.0/29; and the external # interface is 208.13.201.8/29. The firewall's respective interface IP

addresses are 192.168.100.1, 208.13.201.1, and 208.13.201.9.

#

All traffic originating on the internal network is hidden behind the # firewall, i.e. internal packets destined for DMZ hosts are given the # source IP 208.13.201.1 and those destined for the Internet are given # the source IP 208.13.201.9.

#

In the interest of minimizing confusion here, traffic between the DMZ and
the Internet is not "NATted," (though it's certainly a good idea # to use
NATted RFC 1918 IP addresses on your DMZ, or even to NAT non-RFC

1918 addresses in order to add a little obscurity to your security ;-)

#

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 section

Let's save typing & confusion with some variables.

These are NOT SuSE-specific in any way.

NET_INT=192.168.100.0/24

NET_DMZ=208.13.201.0/29

IFACE_INT=eth0

IFACE_DMZ=eth1

IFACE_EXT=eth2

IP_INT=192.168.100.1

IP_DMZ=208.13.201.1

IP_EXT=208.13.201.9

WOOFGANG=208.13.201.2

IPTABLES=/usr/sbin/iptables

test -x $IPTABLES || exit 5

The next 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

# Shell functions sourced from /etc/rc.status: # rc_check check and set local
and overall rc status # rc_status check and set local and overall rc status #
rc_status -v ditto but be verbose in local rc status # rc_status -v -r ditto and
clear the local rc status # rc_failed set local and overall rc status to failed #
rc_reset clear local rc status (overall remains) # rc_exit exit appropriate to
overall rc status . /etc/rc.status

# First reset status of this service rc_reset

# Return values acc. to LSB for all commands but status: # 0 - success

# 1 - misc error

# 2 - invalid or excess args

# 3 - unimplemented feature (e.g. reload) # 4 - insufficient privilege

# 5 - program not installed

# 6 - program not configured

# 7 - program is not running
```

#

Note that starting an already running service, stopping # or restarting a not-running service as well as the restart # with force-reload (in case signalling is not supported) are # considered a success.

/End SuSE-specific stuff.

The rest of this script is non-SuSE specific case "$1" in

start)

echo -n "Loading Firewall's Packet Filters"

SETUP

Load kernel modules first

modprobe ip_tables

modprobe ip_conntrack_ftp

modprobe iptable_nat

modprobe ip_nat_ftp

Flush old rules, old custom tables $IPTABLES --flush

$IPTABLES --delete-chain

$IPTABLES --flush -t nat

$IPTABLES --delete-chain -t nat

Set default-deny policies for all three default chains

$IPTABLES -P INPUT DROP

$IPTABLES -P FORWARD DROP

$IPTABLES -P OUTPUT DROP

Give free reign to loopback interfaces $IPTABLES -I INPUT 1 -i lo -j ACCEPT

$IPTABLES -I OUTPUT 1 -o lo -j ACCEPT

Do some rudimentary anti-IP-spoofing drops on INPUT chain

#

$IPTABLES -A INPUT -s 192.168.0.0/16 -i $IFACE_EXT -j LOG --log-prefix #"Spoofed source IP "

$IPTABLES -A INPUT -s 192.168.0.0/16 -i $IFACE_EXT -j DROP

$IPTABLES -A INPUT -s 172.16.0.0/12 -j LOG --log-prefix "Spoofed source IP "

$IPTABLES -A INPUT -s 172.16.0.0/12 -j DROP

$IPTABLES -A INPUT -s 10.0.0.0/8 -j LOG --log-prefix " Spoofed source IP "

$IPTABLES -A INPUT -s 10.0.0.0/8 -j DROP

$IPTABLES -A INPUT -s ! $NET_DMZ -i $IFACE_DMZ -j LOG --log-prefix "Spoofed source IP "

$IPTABLES -A INPUT -s ! $NET_DMZ -i $IFACE_DMZ -j DROP

$IPTABLES -A INPUT -s ! $NET_INT -i $IFACE_INT -j LOG --log-prefix "Spoofed source IP "

$IPTABLES -A INPUT -s ! $NET_INT -i $IFACE_INT -j DROP

$IPTABLES -A INPUT -s $NET_DMZ -i $IFACE_EXT -j LOG --log-prefix " Spoofed source IP "

$IPTABLES -A INPUT -s $NET_DMZ -i $IFACE_EXT -j DROP

$IPTABLES -A INPUT -s $IP_INT -i $IFACE_INT -j LOG --log-prefix #"Spoofed source IP

(firewall's) "

$IPTABLES -A INPUT -s $IP_INT -i $IFACE_INT -j DROP

$IPTABLES -A INPUT -s $IP_DMZ -i $IFACE_DMZ -j LOG --log-prefix #"Spoofed source IP

(firewall's) "

$IPTABLES -A INPUT -s $IP_DMZ -i $IFACE_DMZ -j DROP

$IPTABLES -A INPUT -s $IP_EXT -i $IFACE_EXT -j LOG --log-prefix "Spoofed source IP

(firewall's) "

$IPTABLES -A INPUT -s $IP_EXT -i $IFACE_EXT -j DROP

Do the same rudimentary anti-IP-spoofing drops on FORWARD chain

#

$IPTABLES -A FORWARD -s 192.168.0.0/16 -i $IFACE_EXT -j LOG --log-prefix " Spoofed source IP "

$IPTABLES -A FORWARD -s 192.168.0.0/16 -i $IFACE_EXT -j DROP

$IPTABLES -A FORWARD -s 172.16.0.0/12 -j LOG --log-prefix "Spoofed source IP "

$IPTABLES -A FORWARD -s 172.16.0.0/12 -j DROP

$IPTABLES -A FORWARD -s 10.0.0.0/8 -j LOG --log-prefix "Spoofed source IP "

$IPTABLES -A FORWARD -s 10.0.0.0/8 -j DROP

$IPTABLES -A FORWARD -s ! $NET_DMZ -i $IFACE_DMZ -j LOG --log-prefix "Spoofed source IP "

$IPTABLES -A FORWARD -s ! $NET_DMZ -i $IFACE_DMZ -j DROP

$IPTABLES -A FORWARD -s ! $NET_INT -i $IFACE_INT -j LOG --log-prefix "Spoofed source IP "

$IPTABLES -A FORWARD -s ! $NET_INT -i $IFACE_INT -j DROP

$IPTABLES -A FORWARD -s $NET_DMZ -i $IFACE_EXT -j LOG --log-prefix "Spoofed source IP "

$IPTABLES -A FORWARD -s $NET_DMZ -i $IFACE_EXT -j DROP

$IPTABLES -A FORWARD -s $IP_INT -i $IFACE_INT -j LOG --log-prefix "Spoofed source IP

(firewall's) "

$IPTABLES -A FORWARD -s $IP_INT -i $IFACE_INT -j DROP

$IPTABLES -A FORWARD -s $IP_DMZ -i $IFACE_DMZ -j LOG --log-prefix "Spoofed source IP

(firewall's) "

$IPTABLES -A FORWARD -s $IP_DMZ -i $IFACE_DMZ -j DROP

$IPTABLES -A FORWARD -s $IP_EXT -i $IFACE_EXT -j LOG --log-prefix "Spoofed source IP

(firewall's) "

$IPTABLES -A FORWARD -s $IP_EXT -i $IFACE_EXT -j DROP

INBOUND POLICY

Accept inbound packets that are part of previously-OK'ed sessions
$IPTABLES -A INPUT -j ACCEPT -m state --state ESTABLISHED,RELATED

Tell netfilter that all TCP sessions must begin with SYN

$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j LOG --log-prefix "Stealth scan attempt?"

$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j DROP

Accept packets initiating SSH sessions from internal network to firewall
$IPTABLES -A INPUT -p tcp -s $NET_INT --dport 22 -m state --state NEW -j ACCEPT

Log anything not accepted above

$IPTABLES -A INPUT -j LOG --log-prefix "Dropped by default (INPUT):"

$IPTABLES -A INPUT -j DROP

OUTBOUND POLICY

If it's part of an approved connection, let it out $IPTABLES -A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

Allow outbound ping (comment-out when not needed!) # $IPTABLES -A OUTPUT -p icmp -j ACCEPT

Allow outbound DNS queries, e.g. to resolve IPs in logs $IPTABLES -A OUTPUT -p udp --dport 53 -j ACCEPT

Allow outbound HTTP for Yast2 Online Update $IPTABLES -A OUTPUT -p tcp --dport 80 -j ACCEPT

Log anything not accepted above

$IPTABLES -A OUTPUT -j LOG --log-prefix "Dropped by default (OUTPUT):"

$IPTABLES -A OUTPUT -j DROP

FORWARD POLICY

If it's part of an approved connection, let it out $IPTABLES -I FORWARD 1 -m state --state RELATED,ESTABLISHED -j ACCEPT

Tell netfilter that all TCP sessions must begin with SYN

$IPTABLES -A FORWARD -p tcp ! --syn -m state --state NEW -j LOG --log-prefix "Stealth scan attempt?"

$IPTABLES -A FORWARD -p tcp ! --syn -m state --state NEW -j DROP

Allow all access to Woofgang's web sites $IPTABLES -A FORWARD -p tcp -d $WOOFGANG --dport 80 -m state --state NEW -j ACCEPT

Allow all access to Woofgang's FTP sites $IPTABLES -A FORWARD -p tcp -d $WOOFGANG --dport 21 -m state --state NEW,RELATED -j ACCEPT

Allow dns from Woofgang to external DNS servers $IPTABLES -A FORWARD -p udp -s $WOOFGANG -m state --state NEW,RELATED --dport 53 -j ACCEPT

NOTE: the next few rules reflect a restrictive stance re. internal users: # only a few services are allowed outward from the internal network.

This may or may not be politically feasible in your environment, i.e., you # really shouldn't "allow all outbound," but sometimes you have no choice.

Allow dns queries from internal hosts to external DNS servers # NOTE: in practice this rule should be source-restricted to internal DNS

servers (that perform recursive queries on behalf of internal users)

#

$IPTABLES -A FORWARD -p udp -s $NET_INT -m state --state NEW,RELATED --dport 53 -j ACCEPT

Allow FTP from internal hosts to the outside world $IPTABLES -A FORWARD -p tcp -s $NET_INT -m state --state NEW,RELATED --dport 21 -j ACCEPT

Allow HTTP from internal hosts to the outside world $IPTABLES -A FORWARD -p tcp -s $NET_INT -m state --state NEW --dport 80 -j ACCEPT

Allow HTTPS from internal hosts to the outside world $IPTABLES -A FORWARD -p tcp -s $NET_INT -m state --state NEW --dport 443 -j ACCEPT

Allow SMTP from internal hosts to the outside world # NOTE: in practice this should be source-restricted to internal mail servers

#

$IPTABLES -A FORWARD -p tcp -s $NET_INT -m state --state NEW --dport 25 -j ACCEPT

Allow SSH from internal hosts to Woofgang # NOTE: in practice this should be source-restricted to internal admin systems

#

$IPTABLES -A FORWARD -p tcp -s $NET_INT -d $WOOFGANG -m state --state NEW --dport 22 -j ACCEPT

Log anything not accepted above - if nothing else, for t-shooting $IPTABLES -A FORWARD -j LOG --log-prefix "Dropped by default (FORWARD):"

$IPTABLES -A FORWARD -j DROP

NAT: Post-Routing

Hide internal network behind firewall $IPTABLES -t nat -A POSTROUTING -s $NET_INT -o $IFACE_EXT -j SNAT --to-source $IP_EXT

$IPTABLES -t nat -A POSTROUTING -s $NET_INT -o $IFACE_DMZ -j SNAT --to-source $IP_DMZ

Remember status and be verbose

rc_status -v

;;

The following commented-out section is active in Example A-1 but # SHOULD NOT BE USED on a live firewall. (It's only here so I can tell you not # to use it!) Sometimes you can justify turning off packet filtering on a # bastion host, but NEVER on a firewall # wide_open)

echo -n "DANGER!! Unloading firewall's Packet Filters! ARE YOU MAD?"

```
#
# $IPTABLES --flush
# $IPTABLES -P INPUT ACCEPT
# $IPTABLES -P FORWARD ACCEPT
# $IPTABLES -P OUTPUT ACCEPT
# Remember status and be verbose
rc_status -v
;;
# Unload all fw rules, leaving default-drop policies stop)
echo -n "Stopping the firewall (in a closed state)!"
$IPTABLES --flush
# Remember status and be quiet
rc_status
;;
status)
echo "Querying iptables status..."
echo " (actually doing iptables --list)..."
$IPTABLES --list; rc=$?
if test $rc = 0; then echo "OK"
```

```
else echo "Hmm, that didn't work for some reason. Bummer."
fi
#rc_status
;;
*)
echo "Usage: $0 {start|stop|status}"
exit 1
;;
esac
rc_exit
```

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.

Linley Dolby was the production editor, and Jeff Holcomb was the copyeditor for Building Secure Servers with Linux. Ann Schirmer was the proofreader. Linley Dolby and Claire Cloutier provided quality control. Julie Hawks wrote the index. Genevieve d'Entremont provided production assistance.

The image on the cover of Building Secure Servers with Linux is a caravan. 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 QuarkXPress 4.1

using Adobe's ITC Garamond font.

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

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

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

[SYMBOL]

[**A**]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[See access control]
authorized_keys file

2nd

3rd

automated hardening

axfr-get, djbdns service

running

axfrdns, djbdns service

installing and running

[SYMBOL]

[A]

[**B**]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

bare-metal recovery

2nd

basic scans

[See simple port scans]
Bastille Linux

2nd

[SYMBOL]

[A]

[B]

[**C**]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

c_rehash

cache poisoning

2nd

3rd

best defense against

[SYMBOL]

[A]

[B]

[C]

[**D**]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

daemon
command-line flag support

hijacked

logging and controlling access

persistent

ProFTPD run as a

rsync
default behavior in daemon mode

running in daemon mode

Sendmail

running in daemon mode

Stunnel

example

running in daemon mode

2nd

vulnerabilities

daemon facility, syslog

2nd

daemontools

djbdns, and

download site

Dante

data confidentiality

overview

data integrity

overview

[SYMBOL]

[A]

[B]

[C]

[D]

[**E**]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

EAO (Expected Annual Occurence)

electronic crimes

email

abuse

architecture

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[**F**]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

facilities, syslog

chart summary

FastCGI

Fcheck

download site

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[**G**]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

Garfinkel, Simson

Generic Service Proxy

[See GSP]
GET method, HTTP

gid, rsync option

GIMP

gtk, GIMP Tool Kit

global versus per-package updates

gmp, scripting environment

Gr?nvall, Bj?rn

Group, Apache option

GSP (Generic Service Proxy)

2nd

gtk, GIMP Tool Kit

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[**H**]

[I]

[J]

[K]

[L]

[M]

[N]

installing/running only necessary software

keeping software up-to-date

Principle of Least Privilege

principles of

rootkits

Tripwire, and

unnecessary packages

FTP

inetd

linuxconfd, system administration tool

network monitoring

POP

r-services

rpc services

scanning tools

Sendmail

software-development environments

Telnet

X Window System

utilities, Bastille Linux

CGI, and

firewalls, and

methods
GET

HEAD

OPTIONS

POST

PUT

httpd.conf file

2nd

Hunt, Craig

Hybris worm

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

IDEA

2nd

IDS (Intrusion Detection Systems)

Audit Based

free

2nd

common options used in Rule Specifications

complete documentation

download site

examples of use

script for a multihomed firewall system

script for running FTP & HTTP services

Stunnel, and

iptables kernel
module

ISS RealSecure

Jaenicke, Lutz

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[**K**]

[L]

[M]

[N]

defined

Kilger, Max

Kim, Gene

Klaus, Christopher

klogd (LinuxÕs kernel log daemon)

2nd

Krause, Micki

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[**M**]

[N]

[See also three-homed host]
defined

MX records

MySQL

2nd

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[**N**]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

named

invoking

named.conf file

acl{} sections

BIND

TCP NULL scan

TCP SYN scan

TCP Xmas Tree scan

UDP scan

nmapfe, nmap GUI

nonanonymous FTP

none facility, syslog

nonliability

Northcutt, Stephen

Novak, Judy

NS records

null-passphrase keys

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

simple

stateful

2nd

Stateful Inspection

packet sniffers

2nd

[See also Snort]
packet-filter Rule Specifications

PAM (Pluggable Authentication Modules)

2nd

pam, SASL method

passphrase

CA key

defined

private-key

protected

passphrase-less key

pair

PasswordAuthentication
ssh_config parameter

sshd_config parameter

passwords, POP3

2nd

3rd

4th

defined

public-key infrastructures

2nd

3rd

PUT method, HTTP

pwcheck_method, SASL variable

Qmail 2nd

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[**R**]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

r-services

Ranum, Marcus

2nd

3rd

4th

Raptor

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[**S**]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

S/KEY

SAINT

salt

Samba

2nd

how it works

quick start instructions

RSA/DSA keys, and

tools
scp

sftp

ssh

ssh-add

2nd

ssh-agent

2nd

ssh-askpass

ssh-keygen

sshd

using to execute remote commands

SSH Communications Security

ssh, SSH tool

compared to Telnet

ssh-add, SSH tool

2nd

ssh-agent, SSH tool

2nd

ssh-askpass, SSH tool

ssh-keygen, SSH tool

2nd

ssh_config file

2nd

3rd

parameters
CheckHostIP

Cipher

Ciphers

Compression

ForwardX11

PasswordAuthentication

sshd (Secure Shell Daemon)

configuring and running

sshd_config file

2nd

3rd

4th

parameters
PasswordAuthentication

PermitEmptyPasswords

PermitRootLogin

mark

multiple

none

user

logging
email and uucp messages

remote

stealth

mapping of actions to facilities and priorities

priorities

chart summary

TCPwrappers, and

syslog-ng

2nd

as its own log watcher, example

compiling and installing

configuring

creating new directories for its log files

destination drivers

file synchronization

global options

libol (support library)

log{} statements

message filters

message sources

official (maintained) documentation

running

startup flags

supported source drivers

syslog-ng.conf file
example

options{} section

syslog.conf file

default

multiple facilities

multiple selectors

syslogd

2nd

flags

mark, turning on

running

unpredictable behavior

SyslogFacility, ProFTPD setting

system
log management and monitoring

log monitoring tools

[See Swatch]
system availability

2nd

system integrity

overview

system-integrity checker
Tripwire

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

taint mode, Perl running in

tarpit

TCP Connect scan

TCP FIN scan

TCP handshake

2nd

[See also Stunnel]3rd

defined

rsync sessions example

tux, open source web and FTP server

tw.cfg file

Tweedie, Stephen

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[**U**]

[V]

[W]

[X]

[Y]

[Z]

UCE (Unsolicited Commercial Email)

discussion on

Postfix, and

SMTP AUTH, and

ucspi-tcp

UUCP

logging messages

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[W]

[X]

[Y]

[Z]

Venema, Wietse

2nd

VERB, SMTP command

VeriSign

2nd

mitigation of

Sendmail

VulnWatch

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[**W**]

[X]

[Y]

[Z]

web
security
FAQ

goals

problems

servers

[SYMBOL]

[A]

[B]

[C]

[D]

[E]

[F]

[G]

[H]

[I]

[J]

[K]

[L]

[M]

[N]

Young, Eric A.

<applet>
<embed>
<object>
<script>
.htaccess files in Apache configuration
.swatchrc file
3DES (Triple-DES) 2nd