Building Internet Firewalls

By D. Brent Chapman & Elizabeth D. Zwicky; ISBN 1-56592-124-0, 517 pages.
First Edition, November 1995.
(See the catalog page for this book.)

Search the text of Building Internet Firewalls.

Index

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | T | U | V | W | X | Y | Z

Table of Contents

Part IV: Apendixes

The Networking CD Bookshelf Navigation

Index: A

3.5 Fail-Safe Stance

Another fundamental principle of security is that, to the extent possible, systems should *fail safe*; that is, if they're going to fail, they should fail in such a way that they deny access to an attacker, rather than letting the attacker in. The failure may also result in denying access to legitimate users as well, until repairs are made, but this is usually an acceptable tradeoff.

Safe failures are another principle with wide application in familiar places. Electrical devices are designed to go off - to stop - when they fail in almost any way. Elevators are designed to grip their cables if they're not being powered. Electric door locks generally unlock when the power fails, to avoid trapping people in buildings.

Most of the applications we discuss automatically fail safely. For example, if a packet filtering router goes down, it doesn't let any packets in. If a proxying program goes down, it provides no service. On the other hand, some host-based packet filtering systems are designed such that packets are allowed to arrive at a machine that runs a packet filtering application and separately runs applications providing services. The way some of these systems work, if the packet filtering application crashes (or is never started at boot time), the packets will be delivered to the applications providing services. This is not a fail-safe design and should be avoided.

The biggest application of this principle in network security is in choosing your site's *stance* with respect to security. Your stance is, essentially, your site's overall attitude towards security. Do you lean towards being restrictive or permissive? Are you more inclined to err in the direction of safety (some might call it paranoia) or freedom?

There are two fundamental stances that you can take with respect to security decisions and policies:

- The default deny stance: Specify only what you allow and prohibit everything else.
- The default permit stance: Specify only what you prohibit and allow everything else.

It may seem obvious to you which of these is the "right" approach to take; from a security point of view, it's the default deny stance. Probably, it will also seem obvious to your users and management; from their point of view, it's the default permit stance. It's important to make your stance clear to users and management, as well as to explain the reasons behind that stance. Otherwise, you're likely to spend a lot of unproductive time in conflict with them, wondering "How could they be so foolish as to even suggest that?" time and again, simply because they don't understand the security point of view.

3.5.1 Default Deny Stance: That Which Is Not Expressly Permitted Is Prohibited

The *default deny stance* makes sense from a security point of view because it is a fail-safe stance. It recognizes that what you don't know *can* hurt you. It's the obvious choice for most security people, but it's usually not at all obvious to users.

With the default deny stance, you prohibit everything by default; then, to determine what you are going to allow, you:

- Examine the services your users want.
- Consider the security implications of these services and how you can safely provide them.
- Allow only the services that you understand, can provide safely, and see a legitimate need for.

Services are enabled on a case-by-case basis. You start by analyzing the security of a specific service, and balance its security implications against the needs of your users. Based on that analysis and the availability of various remedies to improve the security of the service, you settle on an appropriate compromise.

For one service, you might determine that you should provide the service to all users and can do so safely with commonly available packet filtering or proxy systems. For another service, you might determine that the service cannot be adequately secured by any currently available means, but that only a small number of your users or systems require it. In the latter case, perhaps its use can be restricted to that small set of users (who can be made aware of the risks through special training) or systems (which you may be able to protect in other ways; e.g., through host security). The whole key is to find a compromise that is appropriate to your particular situation.

3.5.2 Default Permit Stance: That Which Is Not Expressly Prohibited Is Permitted

Most users and managers prefer the *default permit stance*. They tend to assume that everything will be, by default, permitted, and that certain specific, troublesome actions and services will then be prohibited as necessary. For example:

- NFS is not permitted across the firewall.
- WWW access is restricted to users who have received awareness training about its security problems.
- Users are not allowed to set up unauthorized servers.

They want you to tell them what's dangerous; to itemize those few (they think) things that they can't do, and to let them do everything else. This is definitely not a fail-safe stance.

First, it assumes that you know ahead of time precisely what the specific dangers are, how to explain them so users will understand them, and how to guard against them. Trying to guess what dangers might be in a system or out there on the Internet is essentially an impossible task. There are simply too many possible problems, and too much information (e.g., new security holes, new exploitations of old holes, etc.) to be able to keep up to date. If you don't know that something is a problem, it won't be on your "prohibited" list. In that case, it will go right on being a problem until you notice it and you'll

probably notice it because somebody takes advantage of it.

Second, the default permit stance tends to degenerate into an escalating "arms race" between the firewall maintainer and the users. The maintainer prepares defenses against user action or inaction (or, he just keeps saying, "Don't do that!"); the users come up with fascinating new and insecure ways of doing things; and the process repeats, again and again. The maintainer is forever playing catch up. Inevitably, there are going to be periods of vulnerability between the time that a system is set up, the time that a security problem is discovered, and the time that the maintainer is able to respond to the problem. No matter how vigilant and cooperative everyone may be, some things are going to fall through the cracks forever: because the maintainer has never heard about them, because he has never realized the full security consequences; or because he just plain hasn't had time to work on the problem.

About the only people who benefit from the default permit stance are potential attackers, because the firewall maintainer can't possibly close all the holes, is forever stuck in "fire fighting" mode, and is likely to be far too busy to notice an attacker's activities.

For example, consider the problem of sharing files with collaborators at another site. Your users' first idea will probably be to use the same tool that they use to share files internally - NFS. The problem is, NFS is completely unsafe to allow across a firewall (for reasons discussed in Chapter 2, Internet Services and Chapter 8, Configuring Internet Services). Suppose that your stance is a permissive one, and you haven't specifically told your users that it's not safe to run NFS across your firewall (or even if you have told them, but don't remember or don't care). In this case, you're probably going to find yourself running NFS across your firewall because it seemed like a good idea to somebody who didn't understand (or care about) the security issues. If your stance is default deny, on the other hand, your users' attempts to set up NFS will fail. You'll need to explain why to them, suggest alternatives that are more secure (such as FTP), and look for ways to make those more secure alternatives easier to use without sacrificing security.

Previous: 3.4 Weakest Link | Building Internet Firewalls | Next: 3.6 Universal Participation

3.4 Weakest Link | Book Index | 3.6 Universal Participation

[Library Home | DNS & BIND | TCP/IP | sendmail | sendmail Reference | Firewalls | Practical Security]

Search
Book List
Press Room
Jobs

Perl
Java
Web & Internet
Open Source
XML
Linux
Unix
Python
Macintosh
Windows
.NET
Oracle
Security

Table of Contents
Index
Sample Chapter
Errata
Author's Errata & Website
Colophon
Media Reviews

Register Your Book
Join Email List

Building Internet Firewalls

By D. Brent Chapman, Elizabeth D. Zwicky
1st Edition September 1995

This book is out of print. Please consider the latest edition.

The latest edition is also available on Safari: Tech Books Online.

Buy from O'Reilly: | **View Cart** |

Everyone is jumping on the Internet bandwagon, despite the fact that the security risks associated with connecting to the Net have never been greater. This book is a practical guide to building firewalls on the Internet. It describes a variety of firewall approaches and architectures and discusses how you can build packet filtering and proxying solutions at your site. It also contains a full discussion of how to configure Internet services (e.g., FTP, SMTP, Telnet) to work with a firewall, as well as a complete list of resources, including the location of many publicly available firewall construction tools. [Full Description]

Related O'Reilly Titles:

Computer Crime
Computer Security Basics
Essential System Administration, 2nd Edition

Internet Core Protocols: The Definitive Guide
PGP: Pretty Good Privacy
Practical UNIX & Internet Security, 2nd Edition
Protecting Networks with SATAN
Stopping Spam
The Whole Internet: The Next Generation
Virtual Private Networks, 2nd Edition
Web Security & Commerce

---

webmaster@oreilly.com

Index: B

Index: C

Index: D

Index: E

Index: F

Index: G

Search | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | T | U | V | W | X | Y | Z

Index: H

Index: I

Index: J

Index: K

Index: L

Index: M

Index: N

Index: O

Index: P

Index: R

Index: S

Index: T

Index: U

Index: V

Index: W

Index: X

Index: Y

Index: Z

Foreword

In any society, a small percentage of people are malicious. It is estimated that the Internet now has about 30 to 40 million users. Even if the percentage of malicious users is less than one percent of the overall society, the potential number of malicious users is large enough so that it should concern you.

The number of security incidents reported to the Computer Emergency Response Team Coordination Center (CERT-CC) increases every year - less than 200 in 1989, about 400 in 1991, 1400 in 1993, and 2,241 in 1994. Estimates are that we'll see more than 3000 reported incidents in 1995. Incidents occur at government and military sites, among Fortune 500 companies, at universities, and at small startups. Some incidents involve a single account on a single system. Some (for example, those involving packet sniffers) might involve as many as 100,000 systems. Of course, these numbers are only the tip of the iceberg. Many intrusions aren't reported to the CERT Coordination Center or to other computer security incident response organizations. In fact, many aren't reported at all - in some cases, because the victimized organization would rather avoid publicity or charges of carelessness, in other cases because the intrusions are not even detected.

Nobody knows the correct statistics on how many attacks are actually detected by the sites broken into, but most people in the security community agree that only a few percent are. Here's one of the few statistics I can cite: one incident response team offers a network intrusion service to its customers. With the customer's permission, they try to penetrate a system using the same tools that intruders use in their own attacks. This team found that only 4% of the sites probed detected the penetration attempts. An even more frightening estimate: Bill Cheswick of AT&T Bell Labs believes that of those attacks that do succeed, at least 40% of the attackers gain root access.[1]

[1] *Firewalls Digest*, March 31, 1995.

It isn't only the numbers of incidents that are growing; it's the sophistication of the methods of attack. When the CERT Coordination Center was founded in the wake of the Internet worm in the fall of 1988, the attacks we faced fell into two major categories: password guessing and the exploiting of security holes in operating systems and system programs. Although too many sites still fall victim to such attacks, we're now seeing increasing technical complexity in most of the newer incidents. To some extent, this is the result of increasing consciousness among system users and system administrators - users are choosing better passwords, and administrators are applying system patches more quickly. Unfortunately, the result of this increased security consciousness isn't to stamp out security attacks; it's simply to force the attackers to learn new tricks. Many of today's attacks are more sophisticated. They include the forging of Internet Protocol (IP) addresses (intruders are guessing the sequence numbers associated with network connections and the acknowledgments between machines), the exploiting of the source routing option on IP packets on certain types of UNIX

systems, and the hijacking of open terminal or login sessions.

This is not to say that all users and administrators have learned their lessons about old-style attacks. There is still more education that needs to be done among users and administrators, as well as among managers, who too often won't budget what's needed for security training. It's a dangerous world in cyberspace, and too few people realize it. This is one way in which the growth of the Internet may actually have hurt us. In the early days of the Internet, sites connected to the net usually had a whole staff of hardware and software gurus. Today, connecting to the Internet is so easy that sites forget it takes technical sophistication to connect safely and to stay secure.

Several years ago, I worked with a site in Europe that apparently had been broken into by someone who used a site in the United States to launch the attack. When I contacted the system administrator at the U.S. site, she assured me that they didn't even have computers at their location that were connected to the Internet. I told her the full domain name of the suspect system, and she replied, "Oh, you mean the Sun." It turned out that the Sun had been installed for use in a special application and had been running for years without anyone at the site realizing that it was connected to the Internet. The administrator assured me that they would disconnect the machine. Well, next morning, the European manager sent me another flaming email message - another break-in from the same U.S. system. I called the U.S. system administrator. Yes, she'd disconnected the modems. Yes, she'd disconnected the CRTs. But through it all, she'd managed to leave the system connected to the Internet. She didn't know it, but the attacker did, and he continued to take advantage of the fact.

I talk to system administrators all the time who are frustrated by break-ins, but who haven't done the basics that might prevent these break-ins from succeeding. One system administrator complained that he had reloaded his systems multiple times, and he was still being attacked. It turned out that, although he knew about CERT advisories and vendor security bulletins, he'd never bothered installing them. For example, CERT Advisory CA-93:16 was posted to the net in November of 1993; it advised the UNIX community about a problem with most versions of Sendmail. Vendors had cooperated by providing replacement programs, and the advisory contained a replacement for the */bin/sh* program used in the MProg line of the *sendmail.cf* file. A year and a half later, CERT still gets calls from sites that are broken into using this old Sendmail vulnerability.

Although the number of security incidents continues to increase and the types of attacks become ever more sophisticated, still there is good news for those who care about security. Overall, we've seen a huge growth in awareness of the dangers of connecting to the Internet, and there's a lot of activity in the security community. One manifestation of that is the growth of the Forum of Incident Response and Security Teams (FIRST), which brings together a variety of computer security incident response teams (more than 40 at the time I'm writing this) from government, commercial, and academic organizations. Also heartening is the existence of ever-better security tools that our community makes freely available. The Computer Operations, Audit, and Security Technology (COAST) archive at Purdue is a central point for the collection and testing of many of these tools. (Appendix A, Resources of this book tells you how to contact both organizations.) Finally, the publication of some excellent books and papers on Internet security makes the hard-won wisdom of those at the front available to others.

In these dangerous times, firewalls are the best way to keep your site secure. Although you've got to include other types of security in the mix, if you're serious about connecting to the Internet, firewalls should be at the very center of your security plans. Brent Chapman has been known as the firewalls guru since the early days of firewalls on the Internet; his Firewalls mailing list and his tutorials are witness to that. Elizabeth Zwicky, especially through her work at the System Administrators Guild

(SAGE) is the voice of safe and rational system administration. Together, they have written a book that will raise consciousness of, and competence in, Internet security to a new level.

Ed DeHart
CERT Technical Advisor at the
CERT Coordination Center (CERT-CC)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA
June 1995

Preface

Contents:

This book is a practical guide to building your own firewall. It provides step-by-step explanations of how to design and install a firewall at your site, and how to configure Internet services such as electronic mail, FTP, the World Wide Web, and others to work with a firewall. Firewalls are complex, though, and we can't boil everything down to simple rules. Too much depends on exactly what hardware, operating system, and networking you are using at your site, and what you want your users to be able to do, and not do. We've tried to give you enough rules, examples, and resources here so you'll be able to do the rest on your own.

What is a firewall, and what does it do for you? A firewall is a way to restrict access between the Internet and your internal network. You typically install a firewall at the point of maximum leverage, the point where your network connects to the Internet. The existence of a firewall at your site can greatly reduce the odds that outside attackers will penetrate your internal systems and networks. The firewall can also keep your own users from compromising your systems by sending dangerous information - unencrypted passwords and sensitive data - to the outside world.

The attacks on Internet-connected systems we are seeing today are more serious and more technically complex than those in the past. To keep these attacks from compromising our systems, we need all the help we can get. Firewalls are a highly effective way of protecting your site from these attacks. For that reason, we strongly recommend you include a firewall in your site's overall Internet security plan. However, a firewall should be only one component in that plan. It's also vital that you establish a security policy, that you implement strong host security, and that you consider the use of authentication and encryption devices that work with the firewalls you install. This book will touch on each of these topics while maintaining its focus on firewalls.

Scope of This Book

This book is divided into four parts:

Part I, Network Security, explores the problem of Internet security and focuses on firewalls as part of an effective strategy to solve that problem.

Chapter 1, Why Internet Firewalls?, introduces the major risks associated with using the Internet today; discusses what to protect, and what to protect it against; discusses various security models; and introduces firewalls in the context of what they can and can't do for your site's security.

Chapter 2, Internet Services, outlines the services users want and need from the Internet, and summarizes the security problems posed by those services.

Chapter 3, Security Strategies, outlines the basic security principles an organization needs to understand before it adopts a security policy and invests in specific security mechanisms.

Part II, Building Firewalls, describes how to build firewalls and configure services to run with them.

Chapter 4, Firewall Design, outlines the basic components and major architectures used in constructing firewalls: dual-homed hosts, screened hosts, screened subnets, and variations on these basic architectures.

Chapter 5, Bastion Hosts, presents step-by-step instructions on designing and building the bastion hosts used in many firewall configurations.

Chapter 6, Packet Filtering, describes how packet filtering systems work, and discusses what you can and can't accomplish with them in building a firewall.

Chapter 7, Proxy Systems, describes how proxy clients and servers work, and how to use these systems in building a firewall.

Chapter 8, Configuring Internet Services, describes how to configure each major Internet service to run with a firewall.

Chapter 9, Two Sample Firewalls, presents two sample configurations for basic firewalls.

Chapter 10, Authentication and Inbound Services, discusses the problem of allowing users to access your systems from the Internet, and describes a variety of authentication strategies and products.

Part III, Keeping Your Site Secure, describes how to establish a security policy for your site, maintain your firewall, and handle the security problems that may occur with even the most effective firewalls.

Chapter 11, Security Policies, discusses the importance of having a clear and well-understood security policy for your site, and what that policy should and should not contain. It also discusses ways of getting management and users to accept the policy.

Chapter 12, Maintaining Firewalls, describes how to maintain security at your firewall over time and how to keep yourself aware of new Internet security threats and technologies.

Chapter 13, Responding to Security Incidents, describes what to do when a break-in occurs, or when you suspect that your security is being breached.

Part IV, Apendixes, consists of the following summary appendixes:

Appendix A, Resources, contains a list of places you can go for further information and help with Internet security: World Wide Web pages, FTP sites, mailing lists, newsgroups, response teams, books, papers, and conferences.

Appendix B, Tools, summarizes the best freely available firewall tools and how to get them.

Appendix C, TCP/IP Fundamentals, contains background information on TCP/IP that is essential for anyone building or managing a firewall.

Previous: Foreword
Foreword

Building Internet Firewalls

Book Index

Next: Audience
Audience

Part I: Network Security

Part I explores the problem of Internet security and focuses on firewalls as one very effective piece of the solution to that problem

Chapter 1, Why Internet Firewalls?, introduces the major risks associated with using the Internet today, discusses what you're trying to protect and what you're trying to protect it against, discusses various security models, and introduces firewalls by discussing what they can and can't do for your site's security.

Chapter 2, Internet Services, outlines the services users want need from the Internet and the security problems posed by those services.

Chapter 3, Security Strategies, outlines the basic security principles that an organization needs to understand before it adopts a security policy and invests in specific security mechanisms.

1. Why Internet Firewalls?

Contents:

It is scarcely possible to enter a bookstore, read a magazine or a newspaper, or listen to a news broadcast without seeing or hearing something about the Internet in some guise. It's become so popular that it no longer requires explanations when mentioned in nontechnical publications, and it gets mentioned plenty, in magazines ranging from *The New Yorker* to *Bead and Button*. While nontechnical publications are obsessed with the Internet, the technical publications have moved on and are obsessed with security. It's a logical progression; once the first excitement of having a superhighway in your neighborhood wears off, you're bound to notice that not only does it let you travel, it lets a very large number of strangers show up where you are, and not all of them are people you would have invited.

Both views are true: The Internet is a marvelous technological advance that provides access to information, and the ability to publish information, in revolutionary ways. But it's also a major danger that provides the ability to pollute and destroy information in revolutionary ways. This book is about one way to balance the advantages and the risks - to take part in the Internet while still protecting yourself.

Later in this chapter, we describe different models of security people have used to protect their data and resources on the Internet. Our emphasis in this book is on the network security model and, in particular, the use of Internet firewalls. A firewall is a form of protection that allows a network to connect to the Internet while maintaining a degree of security. The section later in this chapter called "What is an Internet Firewall?" describes the basics of firewalls and summarize what they can - and cannot - do to help make your site secure. Before we discuss what you can do with a firewall, though, we want to describe briefly why you need one. What are you protecting on your systems? What types of attacks and attackers are we seeing today? What types of security can you use to protect your site?

1.1 What Are You Trying to Protect?

A firewall is basically a protective device. If you are building a firewall, the first thing you need to worry about is what you're trying to protect. When you connect to the Internet, you're putting three things at risk:

- Your data: the information you keep on the computers
- Your resources: the computers themselves
- Your reputation

1.1.1 Your Data

Your data has three separate characteristics that need to be protected:

- *Secrecy:* you might not want other people to know it.
- *Integrity:* you probably don't want other people to change it.
- *Availability:* you almost certainly want to be able to use it yourself.

People tend to focus on the risks associated with secrecy, and it's true that those are usually large risks. Many organizations have some of their most important secrets - the designs for their products, their financial records, or student records - on their computers. On the other hand, you may find that at your site it is relatively easy to separate the machines containing this kind of highly secret data from the machines that connect to the Internet.

Suppose that you *can* separate your data in this way, and that none of the information that is Internet accessible is secret. In that case, why should you worry about security? Because secrecy isn't the only thing you're trying to protect. You still need to worry about integrity and availability. After all, if your data isn't secret, and if you don't mind its being changed, and if you don't care whether or not anybody can get to it, why are you wasting disk space on it?

Even if your data isn't particularly secret, you'll suffer the consequences if it's destroyed or modified. Some of these consequences have readily calculable costs: if you lose data, you'll have to pay to have it reconstructed; if you were planning to sell that data in some form, you'll have lost sales regardless of whether the data is something you sell directly, the designs you use to build things from, or the code for a software product. There are also intangible costs associated with any security incident. The most serious is the loss of confidence (user confidence, customer confidence, investor confidence, staff confidence, student confidence, public confidence) in your systems and data and, consequently, a loss of confidence in your organization.

Has Your Data Been Modified?

Security incidents are different from many other types of crimes because detection is unusually difficult. Sometimes, it may take a long time to find out that someone has broken into your site. Sometimes, you'll never know. Even if somebody breaks in but doesn't actually *do* anything to your system or data, you'll probably lose time (hours or days) while you verify that they didn't do anything. In a lot of ways, a brute-force trash-everything attack is a lot easier to deal with than a break-in by somebody who doesn't appear to damage your system. If they trash everything, you bite the bullet, restore from backups, and get on with your life. But, if they don't appear to have done anything, you spend a lot of time second-guessing yourself, trying to make *sure* they haven't done anything to damage your system or data. Although this book is about preventing security incidents, Chapter 13, Responding to Security Incidents supplies some general guidelines for detecting, investigating, and recovering from security incidents.

1.1.2 Your Resources

Even if you have data you don't care about - even if you enjoy reinstalling your operating system every week because it exercises the disks, or something like that - if other people are going to use your computers, you probably would like to benefit from this use in some way. Most people want to use their own computers, or they want to charge other people for using them. Even people who give away computer time and disk space usually expect to get good publicity and thanks for it; they aren't going to get it from intruders. You spend good time and money on your computing resources, and it is your right to determine how they are used.

Intruders often argue that they are using only excess resources; as a consequence, their intrusions don't cost their victims anything. There are two problems with this argument.

First, it's impossible for an intruder to determine successfully what resources are excess and use only those. It may look as if your system has oceans of empty disk space and hours of unused computing time; in fact, though, you might be just about to start computing animation sequences that are going to use every bit and every microsecond. An intruder can't give back your resources when you want them. (Along the same lines, I don't usually use my car between midnight and 6 A.M., but that doesn't mean I'm willing to lend it to you without being asked. What if I have an early-morning flight the next day, or what if I'm called out to deal with an emergency?)

Second, it's your right to use your resources the way you want to, even if you merely feel some sort of Zen joy at the sight of empty disk space, or if you like the way the blinky lights look when nothing's happening on your computer. Computing resources are not natural resources, nor are they limited resources that are wasted or destroyed if they're not used.

1.1.3 Your Reputation

An intruder appears on the Internet with your identity. Anything he does appears to come from you. What are the consequences?

Most of the time, the consequences are simply that other sites - or law enforcement agencies - start calling you to ask why you're trying to break into their systems. (This isn't as rare an occurrence as it may seem. One site got serious about security when its system administration staff added a line item to their time cards for conversations with the FBI about break-in attempts originating from their site.)

Sometimes, such impostors cost you a lot more than lost time. An intruder who actively dislikes you, or simply takes pleasure in making life difficult for strangers, may send electronic mail or post news messages that purport to come from you. Generally, people who choose to do this aim for maximum hatefulness, rather than believability, but even if only a few people believe these messages, the cleanup can be long and humiliating. Anything even remotely believable can do permanent damage to your reputation.

A few years ago, an impostor posing as a Texas A&M professor sent out hate email containing racist comments to thousands of recipients. The impostor was never found, and the professor is still dealing with the repercussions of the forged messages. In another case, a student at Dartmouth sent out email over the signature of a professor late one night during exam period. Claiming a family emergency, the forged mail canceled the next day's exam, and only a few students showed up.

It's possible to forge electronic mail or news without gaining access to a site, but it's much easier to show that a message is a forgery if it's generated from outside the forged site. The messages coming

from an intruder who has gained access to your site will look exactly like yours because they *are* yours. An intruder will also have access to all kinds of details that an external forger won't. For example, an intruder has all of your mailing lists available and knows exactly who you send mail to.

Even if an intruder doesn't use your identity, a break-in at your site isn't good for your reputation. It shakes people's confidence in your organization. In addition, most intruders will attempt to go from your machines to others, which is going to make their next victims think of your site as a platform for computer criminals. Many intruders will also use compromised sites as distribution sites for pirated software and/or pornography, which is not going to endear you to many folks either. Whether or not it's your fault, having your name linked to other intrusions, software piracy, and pornography is hard to recover from.

Previous: I. Network Security | Building Internet Firewalls | Next: 1.2 What Are You Trying To Protect Against?

I. Network Security | Book Index | 1.2 What Are You Trying To Protect Against?

[Library Home | DNS & BIND | TCP/IP | sendmail | sendmail Reference | Firewalls | Practical Security]

2. Internet Services

Contents:

In Chapter 1, Why Internet Firewalls? we discussed, in general terms, what you're trying to protect when you connect to the Internet: your data, your resources, and your reputation. In designing an Internet firewall, your concerns are more specific; what you need to protect are those services you're going to use or provide over the Internet.

There are a number of standard Internet services that users want and that most sites try to support. There are important reasons to use these services; indeed, without them, there is little reason to be connected to the Internet at all. But there are also potential security problems with each of them.

What services do you want to support at your site? Which ones can you support securely? Every site is different. Every site has its own security policy and its own working environment. For example, do all your users need electronic mail? Do they all need to transfer files outside your organization? How about downloading files from sites outside the organization's own network? Who should be able to log in remotely from another location over the Internet?

This chapter briefly summarizes the major Internet services your users may be interested in using. It provides only a high-level summary (details are given in Chapter 8, Configuring Internet Services). None of these services are really secure; each one has its own security weaknesses, and each has been exploited in various ways by attackers. Before you decide to support a service at your site, you will

have to assess how important it is to your users and whether you will be able to protect them from its dangers. There are various ways of doing this: running the services only on certain protected machines, using especially secure variants of the standard services; or, in some cases, blocking the services completely to or from some or all outside systems.

This chapter doesn't list every Internet service - it can't. Such a list would be incomplete as soon as it was finished, and would include services of interest only to a few sites in the world. Instead, we attempt to list the major services, and we hope this book will give you the background you need to make decisions about new services as you encounter them.

Managers and system administrators together need to decide which services to support at your site and to what extent. Chapter 8 describes what is necessary to support these services securely once you've decided to provide them, and the decisions you may need to make about them when building your site's firewall.

Getting Started with Internet Services

Are you just getting connected? Or, have you been connected for a while but are getting concerned about Internet security? Where should you start? Many system administrators try to be too ambitious. If you attempt to develop and deploy the be-all and end-all of firewall systems right from day one, you probably aren't going to succeed. The field is just too complex, and the technology is changing so fast that it will change out from under you before you get such an endeavor "finished."

Start small. At many sites, it boils down to six basic services. If you can provide these services securely, most of your users will be satisfied, at least for a while.

- Electronic mail (SMTP)
- File transfer (FTP)
- Usenet news (NNTP)
- Remote terminal access (Telnet)
- World Wide Web access (HTTP)
- Hostname/address lookup (DNS): users generally don't use this service directly, but it underlies the other five services by translating Internet hostnames to IP addresses and vice versa.

All six of these services can be safely provided in a number of different ways, including packet filtering and proxies - firewall approaches discussed in Part II of this book. Providing these services lets your users access most Internet resources, and it buys you time to figure out how to provide the rest of the services they'll be asking for soon, such as Archie, Gopher, WAIS, and other information services.

2.1 Electronic Mail

Electronic mail is one of the most popular and basic network services. It's relatively low risk, but that doesn't mean it's risk free. Forging electronic mail is trivial (just as is forging regular postal mail), and forgeries facilitate two different types of attacks: attacks against your reputation and social manipulation attacks (e.g., attacks in which users are sent mail purporting to come from an administrator and advising them to change to a specific password). Accepting electronic mail ties up

computer time and disk space, opening you up for denial of service attacks, although with proper configuration, only email service will be denied. Particularly with modern multimedia mail systems, people can send electronic mail containing programs that run with insufficient supervision and may turn out to be *Trojan horses*.

Although people worry most about the last risk mentioned above, in practice the most common problems with electronic mail are inadvertent floods (including chain letters) and people who put entirely inappropriate confidence in the confidentiality of electronic mail and send proprietary data via electronic mail across the Internet. However, as long as users are educated, and the mail service is isolated from other services so that inadvertent or purposeful denial of service attacks shut down as little as possible, electronic mail is reasonably safe.

Simple Mail Transfer Protocol (SMTP) is the Internet standard protocol for sending and receiving electronic mail. SMTP itself is not usually a security problem, but SMTP servers can be. A program that delivers mail to users often needs to be able to run as any user that might receive mail. This gives it broad power and makes it a tempting target for attackers.

The most common SMTP server on UNIX is Sendmail. Sendmail has been exploited in a number of break-ins, including the Internet worm, which makes people nervous about using it. Many of the available replacements, however, are not clearly preferable to Sendmail; the evidence suggests they are less exploited because they are less popular, not because they are less vulnerable. There are exceptions in programs designed explicitly for security, but these don't support all the functions necessary to send and receive arbitrary mail messages; some things are still best handled by Sendmail running in a secured space.

Previous: 1.4 What Is an Internet Firewall?
1.4 What Is an Internet Firewall?

Building Internet Firewalls

Book Index

Next: 2.2 File Transfer
2.2 File Transfer

[Library Home | DNS & BIND | TCP/IP | sendmail | sendmail Reference | Firewalls | Practical Security]

3. Security Strategies

Contents:

Before we discuss the details of firewalls, it's important to understand some of the basic strategies employed in building firewalls and in enforcing security at your site. These are not staggering revelations; they are straightforward approaches. They're presented here so that you can keep them in mind as you put together a firewall solution for your site.

3.1 Least Privilege

Perhaps the most fundamental principle of security (any kind of security, not just computer and network security) is that of *least privilege*. Basically, the principle of least privilege means that any object (user, administrator, program, system, whatever) should have only the privileges the object needs to perform its assigned tasks - and no more. Least privilege is an important principle for limiting your exposure to attacks and for limiting the damage caused by particular attacks.

Some car manufacturers set up their locks so that one key works the doors and the ignition, and a different key works the glove compartment and the trunk; that way, you can enforce least privilege by giving a parking lot attendant the ability to park the car without the ability to get at things stored in the trunk. Many people use splittable key chains, for the same reason. You can enforce least privilege by giving someone the key to your car, but not the key to your house as well.

In the Internet context, the examples are endless. Every user probably doesn't need to access every Internet service. Every user probably doesn't need to modify (or even read) every file on your system. Every user probably doesn't need to know the machine's root password. Every system administrator probably doesn't need to know the root passwords for all systems. Every system probably doesn't need to access every other system's files.

Applying the principle of least privilege suggests that you should explore ways to reduce the

privileges required for various operations. For example:

- Don't give a user the root password for a system if all she needs to do is reset the print system. Instead, write a privileged program the user can run that resets the print system.
- Don't make a program run *setuid* to root if all it needs to do is write to one protected file. Instead, make the file group-writable to some group and make the program run *setgid* to that group rather than *setuid* to root.
- Don't have your internal systems trust one of your firewall machines just so it can do backups. Instead, make the firewall machine trust the internal system, or, better yet, put a local tape drive on the firewall machine so that it can do its own backups.

Many of the common security problems on the Internet can be viewed as failures to follow the principle of least privilege. For example, there have been and continue to be any number of security problems discovered in Sendmail, which is a big, complex program; any such program is going to have bugs in it. The problem is that Sendmail runs (at least some of the time) *setuid* to root; many of the attacks against Sendmail take advantage of this. Because it runs as root, Sendmail is a high-value target that gets a lot of attention from attackers; the fact that it's a complex program just makes their jobs easier. This implies both that privileged programs should be as simple as possible and that, if a complex program requires privileges, you should look for ways to separate and isolate the pieces that need privileges from the complex parts.[1]

> [1] It's important to realize that Sendmail is far from the only example we could cite; you can find similar problems in almost any large, complex, privileged piece of software.

Many of the solutions you'll employ in protecting your site are tactics for enforcing the strategy of least privilege. For example, a packet filtering system is designed to allow in packets for the services you want. Running insecure programs in an environment where only the privileges the programs absolutely need are available to them (e.g., a machine that's been stripped down in one way or another) is another example; this is the essence of a bastion host.

There are two problems with trying to enforce least privilege. First, it can be complex to implement when it isn't already a design feature of the programs and protocols you're using. Trying to add it on may be very difficult to get right. Some of the cars that try to implement least privilege with separate keys for the trunk and the ignition have remote trunk release buttons that are accessible without the keys, or fold-down rear seats that allow you to access the trunk without opening it the traditional way at all. You need to be very careful to be sure that you've actually succeeded in implementing least privilege.

Second, you may end up implementing something less than least privilege. Some cars have the gas cap release in the glove compartment. That's intended to keep parking lot attendants from siphoning off your gas, but if you lend a friend your car, you probably want them to be able to fill it up with gas. If you give your friend only the ignition key, you're giving them less than the minimum privilege you want them to have (because they won't be able to fill up the gas tank), but adding the key to the trunk and the glove compartment may give them more privilege than you want them to have.

You may find similar effects with computer implementations of least privilege. Trying to enforce least privilege on people, rather than programs, can be particularly dangerous. You can predict fairly well what permissions Sendmail is going to need to do its job; human beings are less predictable, and more likely to become annoyed and dangerous if they can't do what they want to. Be very careful to avoid turning your users into your enemies.

Part II: Building Firewalls

Part II describes how to build firewalls, configure services to run with them, and maintain firewalls over time.

Chapter 4, Firewall Design, outlines the basic components and major architectures used in constructing firewalls -- dual-homed hosts, screened hosts, screened subnets, and variations on these basic architectures.

Chapter 5, Bastion Hosts, presents step-by-step instructions for how to design and build the bastion hosts used in many firewall configurations.

Chapter 6, Packet Filtering, describes how proxy clients and servers work, and how to use these systems in building a firewall.

Chapter 7, Proxy Systems, describes how packet filtering systems work and discusses what you can and can't accomplish with them in building a firewall.

Chapter 8, Configuring Internet Services, describes how to configure each major Internet service to run with a firewall.

Chapter 9, Two Sample Firewalls, presents two sample configurations for basic firewalls.

Chapter 10, Authentication and Inbound Services, discusses the problem of allowing users to access your systems from the Internet, and describes a variety of authentication strategies and products.

4. Firewall Design

Contents:

In Chapter 1, Why Internet Firewalls?, we introduced Internet firewalls and summarized what they can and cannot do to improve network security. In this chapter, we present major firewalls concepts. What are the terms you will hear in discussions of Internet firewalls? What types of firewall architectures are used at sites today? What are the components that can be put together to build these common firewall architectures? In the remaining chapters of this book, we'll describe these components and architectures in detail.

4.1 Some Firewall Definitions

You may be familiar with some of the firewall terms listed below, and some may be new to you. Some may seem familiar, but they may be used in a way that is slightly different from what you're accustomed to (though we try to use terms that are as standard as possible). Unfortunately, there is no completely consistent terminology for firewall architectures and components. Different people use terms in different - or, worse still, conflicting - ways. Also, these same terms sometimes have other meanings in other networking fields; the definitions below are for a firewalls context.

These are very basic definitions; we describe these terms in greater detail elsewhere.

Firewall

A component or set of components that restricts access between a protected network and the Internet, or between other sets of networks.

Host

A computer system attached to a network.

Bastion host

A computer system that must be highly secured because it is vulnerable to attack, usually because it is exposed to the Internet and is a main point of contact for users of internal networks. It gets its name from the highly fortified projections on the outer walls of medieval

castles.[1]

> [1] Marcus Ranum, who is generally held responsible for the popularity of this term in the firewalls professional community, says, "Bastions...overlook critical areas of defense, usually having stronger walls, room for extra troops, and the occasional useful tub of boiling hot oil for discouraging attackers."

Dual-homed host

A general-purpose computer system that has at least two network interfaces (or homes)

Packet

The fundamental unit of communication on the Internet.

Packet filtering

The action a device takes to selectively control the flow of data to and from a network. Packet filters allow or block packets, usually while routing them from one network to another (most often from the Internet to an internal network, and vice versa). To accomplish packet filtering, you set up a set of rules that specify what types of packets (e.g., those to or from a particular IP address or port) are to be allowed and what types are to be blocked. Packet filtering may occur in a router, in a bridge, or on an individual host. It is sometimes known as *screening*.[2]

> [2] Some networking literature (in particular, the BSD UNIX release from Berkeley) uses the term "packet filtering" to refer to something else entirely (selecting certain packets off a network for analysis, as is done by the *etherfind* or *tcpdump* programs).

Perimeter network

A network added between a protected network and an external network, in order to provide an additional layer of security. A perimeter network is sometimes called a *DMZ*, which stands for *De-Militarized Zone* (named after the zone separating North and South Korea).

Proxy server

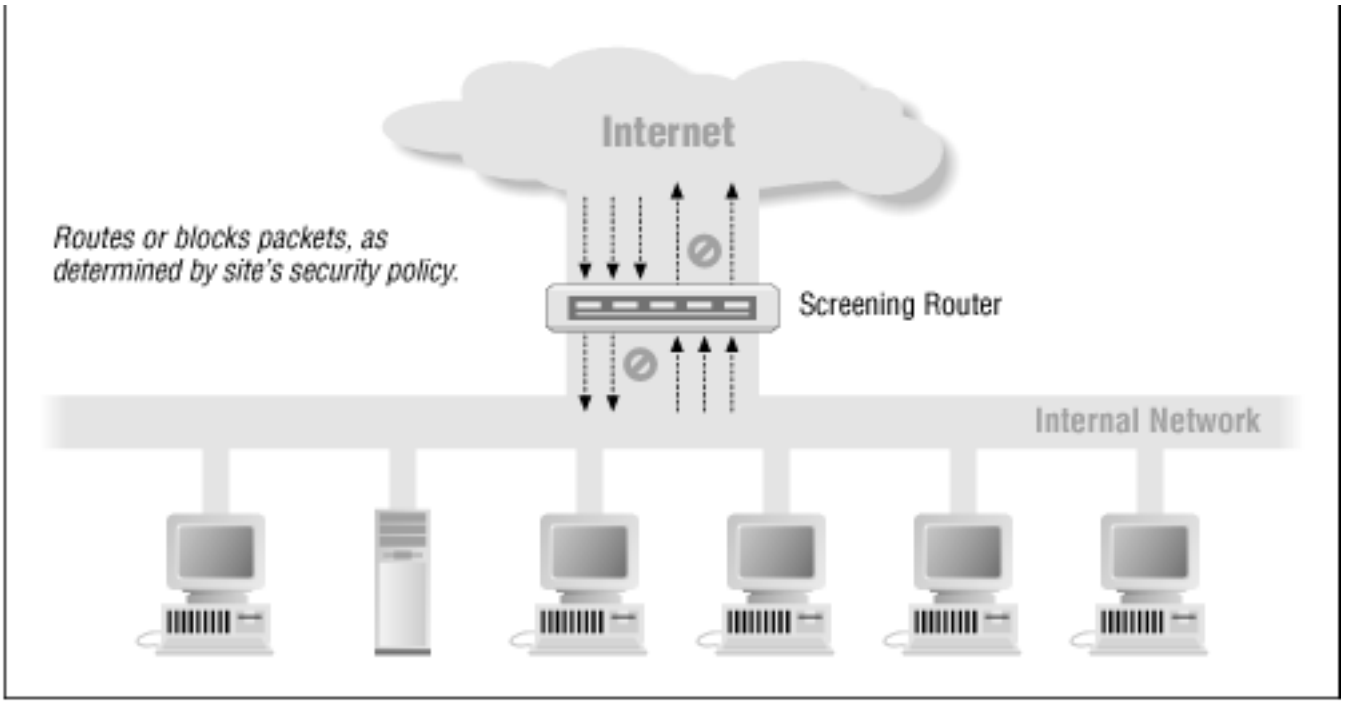
A program that deals with external servers on behalf of internal clients. Proxy clients talk to proxy servers, which relay approved client requests on to real servers, and relay answers back to clients.

The next few sections briefly describe packet filtering and proxy services, two major approaches used to build firewalls today.

4.1.1 Packet Filtering

Packet filtering systems route packets betweeen internal and external hosts, but they do it selectively. They allow or block certain types of packets in a way that reflects a site's own security policy as shown in Figure 4.1. The type of router used in a packet filtering firewall is known as a *screening router*.

Figure 4.1: Using a screening router to do packet filtering



As we discuss in Chapter 6, Packet Filtering, every packet has a set of headers containing certain information. The main information is:

- IP source address
- IP destination address
- Protocol (whether the packet is a TCP, UDP, or ICMP packet)
- TCP or UDP source port
- TCP or UDP destination port
- ICMP message type

In addition, the router knows things about the packet that aren't reflected in the packet headers, such as:

- The interface the packet arrives on
- The interface the packet will go out on

The fact that servers for particular Internet services reside at certain port numbers lets the router block or allow certain types of connections simply by specifying the appropriate port number (e.g., TCP port 23 for Telnet connections) in the set of rules specified for packet filtering. (Chapter 6 describes in detail how you construct these rules.)

Here are some examples of ways in which you might program a screening router to selectively route packets to or from your site:

- Block all incoming connections from systems outside the internal network, except for incoming SMTP connections (so that you can receive email).
- Block all connections to or from certain systems you distrust.
- Allow email and FTP services, but block dangerous services like TFTP, the X Window System, RPC, and the "r" services (*rlogin*, *rsh*, *rcp*, etc.).

To understand how packet filtering works, let's look at the difference between an ordinary router and a screening router.

An ordinary router simply looks at the destination address of each packet and picks the best way it knows to send that packet towards that destination. The decision about how to handle the packet is based solely on its destination. There are two possibilities: the router knows how to send the packet towards its destination, and it does so; or the router does not know how to send the packet towards its destination, and it returns the packet, via an ICMP "destination unreachable" message, to its source.

A screening router, on the other hand, looks at packets more closely. In addition to determining whether or not it *can* route a packet towards its destination, a screening router also determines whether or not it *should*. "Should" or "should not" are determined by the site's security policy, which the screening router has been configured to enforce.

Although it is possible for only a screening router to sit between an internal network and the Internet, as shown in Figure 4.1, this places an enormous responsibility on the screening router. Not only does it need to perform all routing and routing decision-making, but it is the only protecting system; if its security fails (or crumbles under attack), the internal network is exposed. Furthermore, a straightforward screening router can't modify services. A screening router can permit or deny a service, but it can't protect individual operations within a service. If a desirable service has insecure operations, or if the service is normally provided with an insecure server, packet filtering alone can't protect it.

A number of other architectures have evolved to provide additional security in packet filtering firewall implementations. Later in this chapter, we show the way that additional routers, bastion hosts, and perimeter networks may be added to the firewall implementations in the screened host and screened subnet architectures.

4.1.2 Proxy Services

Proxy services are specialized application or server programs that run on a firewall host: either a dual-homed host with an interface on the internal network and one on the external network, or some other bastion host that has access to the Internet and is accessible from the internal machines. These programs take users' requests for Internet services (such as FTP and Telnet) and forward them, as appropriate according to the site's security policy, to the actual services. The proxies provide replacement connections and act as gateways to the services. For this reason, proxies are sometimes known as *application-level gateways*.[3]

> [3] Firewall terminologies differ. Whereas we use the term *proxy service* to encompass the entire proxy approach, other authors refer to *application-level gateways* and *circuit-level gateways*. Although there are small differences between the meanings of these various terms, which we'll explore in Chapter 7, Proxy Systems, in general our discussion of proxies refers to the same type of technology other authors mean when they refer to these gateway systems.

Proxy services sit, more or less transparently, between a user on the inside (on the internal network) and a service on the outside (on the Internet). Instead of talking to each other directly, each talks to a proxy. Proxies handle all the communication between users and Internet services behind the scenes.

Transparency is the major benefit of proxy services. It's essentially smoke and mirrors. To the user, a

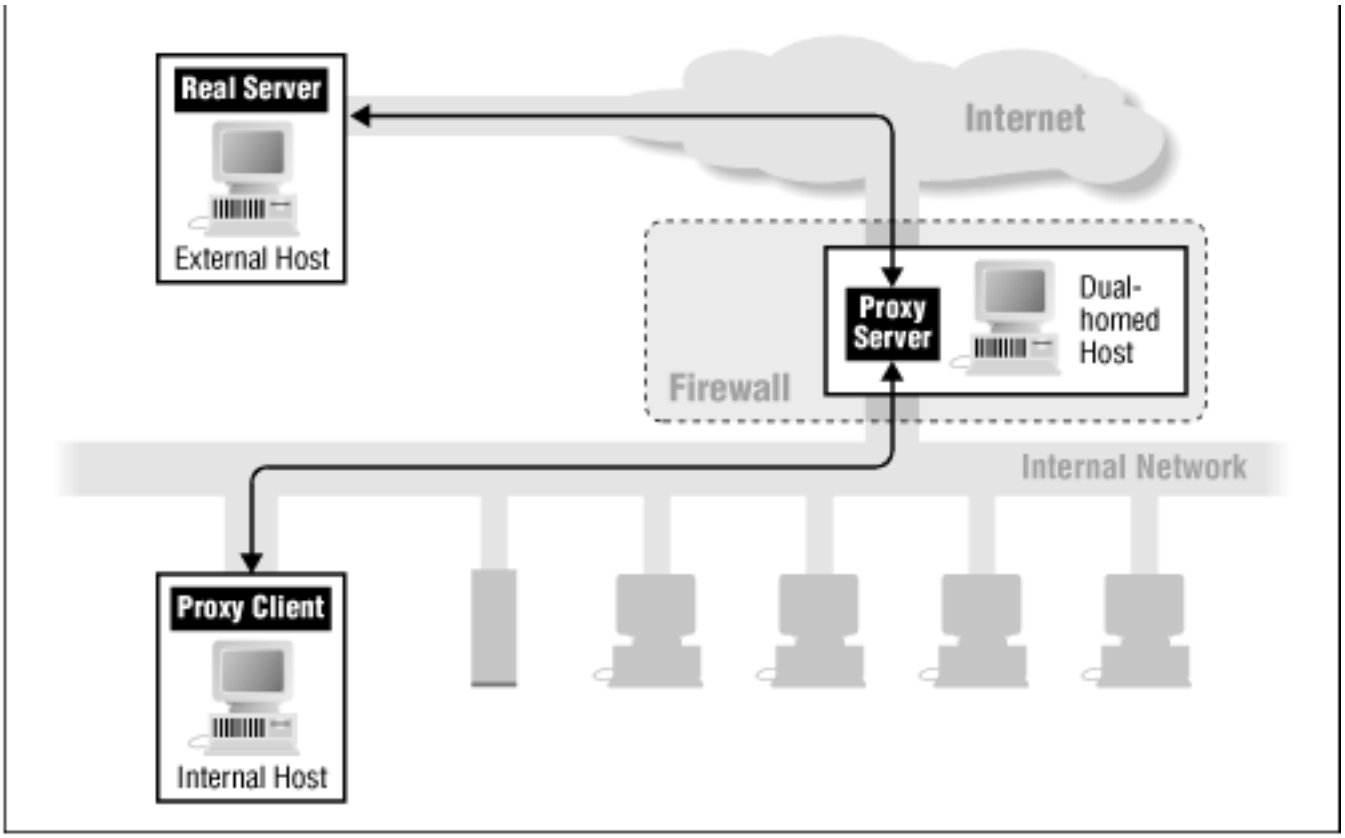
proxy server presents the illusion that the user is dealing directly with the real server. To the real server, the proxy server presents the illusion that the real server is dealing directly with a user on the proxy host (as opposed to the user's real host).

> **NOTE:** Proxy services are effective only when they're used in conjunction with a mechanism that restricts direct communications between the internal and external hosts. Dual-homed hosts and packet filtering are two such mechanisms. If internal hosts are able to communicate directly with external hosts, there's no need for users to use proxy services, and so (in general) they won't. Such a bypass probably isn't in accordance with your security policy.

How do proxy services work? Let's look at the simplest case, where we add proxy services to a dual-homed host. (We'll describe these hosts in some detail in "Dual-Homed Host Architectures" later in this chapter.)

As Figure 4.2 shows, a proxy service requires two components: a proxy server and a proxy client. In this situation, the *proxy server* runs on the dual-homed host. A *proxy client* is a special version of a normal client program (i.e., a Telnet or FTP client) that talks to the proxy server rather than to the "real" server out on the Internet; in addition, if users are taught special procedures to follow, normal client programs can often be used as proxy clients. The proxy server evaluates requests from the proxy client, and decides which to approve and which to deny. If a request is approved, the proxy server contacts the real server on behalf of the client (thus the term "proxy"), and proceeds to relay requests from the proxy client to the real server, and responses from the real server to the proxy client.

Figure 4.2: Using proxy services with a dual-homed host



In some proxy systems, instead of installing custom client proxy software, you'll use standard software, but set up custom user procedures for using it. (We'll describe how this works in Chapter 7.)

A proxy service is a software solution, not a firewall architecture per se. You can use proxy services

in conjunction with any of the firewall architectures described in the section called "Firewall Architectures" below.

The proxy server doesn't always just forward users' requests on to the real Internet services. The proxy server can control what users do, because it can make decisions about the requests it processes. Depending on your site's security policy, requests might be allowed or refused. For example, the FTP proxy might refuse to let users export files, or it might allow users to import files only from certain sites. More sophisticated proxy services might allow different capabilities to different hosts, rather than enforcing the same restrictions on all hosts.

There is some excellent software available for proxying. SOCKS is a proxy construction toolkit, designed to make it easy to convert existing client/server applications into proxy versions of those same applications. The Trusted Information Systems Internet Firewall Toolkit (TIS FWTK) includes proxy servers for a number of common Internet protocols, including Telnet, FTP, HTTP, *rlogin*, X11, and others; these proxy servers are designed to be used in conjunction with custom user procedures. See the discussion of these packages in Chapter 7.

Many standard client and server programs, both commercial and freely available, now come equipped with their own proxying capabilities, or with support for generic proxy systems like SOCKS. These capabilities can be enabled at run time or compile time.

4.1.3 Using a Combination of Techniques and Technologies

The "right solution" to building a firewall is seldom a single technique; it's usually a carefully crafted combination of techniques to solve different problems. Which problems you need to solve depend on what services you want to provide your users and what level of risk you're willing to accept. Which techniques you use to solve those problems depend on how much time, money, and expertise you have available.

Some protocols (e.g., Telnet and SMTP) can be more effectively handled with packet filtering. Others (e.g., FTP, Archie, Gopher, and WWW) are more effectively handled with proxies. (Chapter 8, Configuring Internet Services describes how to handle specific services in a firewall environment.) Most firewalls use a combination of proxying and packet filtering.

5. Bastion Hosts

Contents:

A bastion host is your public presence on the Internet. Think of it as the lobby of a building. Outsiders may not be able to go up the stairs and may not be able to get into the elevators, but they can walk freely into the lobby and ask for what they want. (Whether or not they will get what they ask for depends upon the building's security policy.) Like the lobby in your building, a bastion host is exposed to potentially hostile elements. The bastion host is the system that any outsiders - friends or possible foes - must ordinarily connect with to access a system or a service that's inside your firewall.

By design, a bastion host is highly exposed, because its existence is known to the Internet. For this reason, firewall builders and managers need to concentrate security efforts on the bastion host. You should pay special attention to the host's security during initial construction and ongoing operation. Because the bastion host is the most exposed host, it also needs to be the most fortified host.

Although we talk about a single bastion host in this chapter and elsewhere in this book, remember there may be multiple bastion hosts in a firewall configuration. The number depends on a site's particular requirements and resources, as discussed in Chapter 4, Firewall Design. Each is set up according to the same general principles, using the same general techniques.

Bastion hosts are used with many different firewall approaches and architectures; most of the information in this chapter should be relevant regardless of whether you're building a bastion host to use with a firewall based on packet filtering, proxying, or a hybrid approach. The principles and procedures for building a bastion host are extensions of those for securing any host. You want to use them, or variations of them, for any other host that's security critical, and possibly for hosts that are critical in other ways (e.g., major servers on your internal network).

5.1 General Principles

There are two basic principles for designing and building a bastion host: Keep it simple, and be prepared for the bastion host to be compromised.

Keep it simple

The simpler your bastion host is, the easier it is to secure.

Any service the bastion host offers could have software bugs or configuration errors in it, and any bugs or errors may lead to security problems. Therefore, you want the bastion host to do as little as possible. It should provide the smallest set of services with the least privileges it possibly can, while still fulfilling its role.

Be prepared for the bastion host to be compromised

Despite your best efforts to ensure the security of the bastion host, break-ins can occur. Don't be naive about it. Only by anticipating the worst, and planning for it, will you be most likely to avert it. Always keep the question, "What if the bastion host is compromised?" in the back of your mind as you go through the steps of securing the machine and the rest of the network.

Why do we emphasize this point? The reason is simple: the bastion host is the machine most likely to be attacked because it's the machine most accessible to the outside world. It's also the machine from which attacks against your internal systems are most likely to come because the outside world probably can't talk to your internal systems directly. Do your best to ensure that the bastion host *won't* get broken into, but keep in mind the question, "What if it does?"

In case the bastion host is broken into, you don't want that break-in to lead to a compromise of the entire firewall. You can prevent this by not letting internal machines trust the bastion host any more than is absolutely necessary for the bastion host to function. You will need to look carefully at each service the bastion host provides to internal machines, and determine, on a service-by-service basis, how much trust and privilege each service really needs to have.

Once you've made these decisions, you can use a number of mechanisms to enforce them. For example, you might install standard access control mechanisms (passwords, authentication devices, etc.) on the internal hosts, or you might set up packet filtering between the bastion host and the internal hosts.

6. Packet Filtering

Contents:

Packet filtering is a network security mechanism that works by controlling what data can flow to and from a network. We provide a very brief introduction to high-level IP networking concepts (a necessity for understanding packet filtering) here, but if you're not already familiar with the topic, then before continuing, you should refer to Appendix C, TCP/IP Fundamentals for a more detailed discussion.

To transfer information across a network, the information has to be broken up into small pieces, each of which is sent separately. Breaking the information into pieces allows many systems to share the network, each sending pieces in turn. In IP networking, those small pieces of data are called *packets*. All data transfer across IP networks happens in the form of packets.

The basic device that interconnects IP networks is called a *router*. A router may be a dedicated piece of hardware that has no other purpose, or it may be a piece of software that runs on a general-purpose UNIX or PC (MS-DOS, Windows, Macintosh, or other) system. Packets traversing an internetwork (a network of networks) travel from router to router until they reach their destination. The Internet itself is sort of the granddaddy of internetworks - the ultimate "network of networks."

A router has to make a routing decision about each packet it receives; it has to decide how to send that packet on towards its ultimate destination. In general, a packet carries no information to help the router in this decision, other than the IP address of the packet's ultimate destination. The packet tells the router where it wants to go, but not how to get there. Routers communicate with each other using "routing protocols" such as the Routing Information Protocol (RIP) and Open Shortest Path First (OSPF) to build *routing tables* in memory to determine how to get the packets to their destinations. When routing a packet, a router compares the packet's destination address to entries in the routing

table and sends the packet onward as directed by the routing table. Often, there won't be a specific route for a particular destination, and the router will use a "default route;" generally, such a route directs the packet towards smarter or better-connected routers. (The default routes at most sites point towards the Internet.)

In determining how to forward a packet towards its destination, a normal router looks only at a normal packet's destination address and asks only "*How* can I forward this packet?" A packet filtering router also considers the question "*Should* I forward this packet?" The packet filtering router answers that question according to the security policy programmed into the router via the packet filtering rules.

> **NOTE:** Some unusual packets do contain routing information about how they are to reach their destination, using the "source route" IP option. These packets, called *source-routed packets*, are discussed in the section called "IP Options" below.

6.1 Why Packet Filtering?

Packet filtering lets you control (allow or disallow) data transfer based on:

- The address the data is (supposedly) coming from
- The address the data is going to
- The session and application protocols being used to transfer the data

Most packet filtering systems don't do anything based on the data itself; they don't make content-based decisions.[1] Packet filtering will let you say:

> [1] Some packages, like CheckPoint's FireWall-1 product, are limited exceptions to this rule.

> Don't let anybody use Telnet (an application protocol) to log in from the outside.

or:

> Let everybody send us email via SMTP (another application protocol).

or even:

> That machine can send us news via NNTP (yet another application protocol), but no other machines can do so.

However, it won't let you say:

> This user can Telnet in from outside, but no other users can do so.

because "user" isn't something a packet filtering system can identify. And, it won't let you say:

> You can transfer these files but not those files.

because "file" also isn't something the packet filtering system can identify.

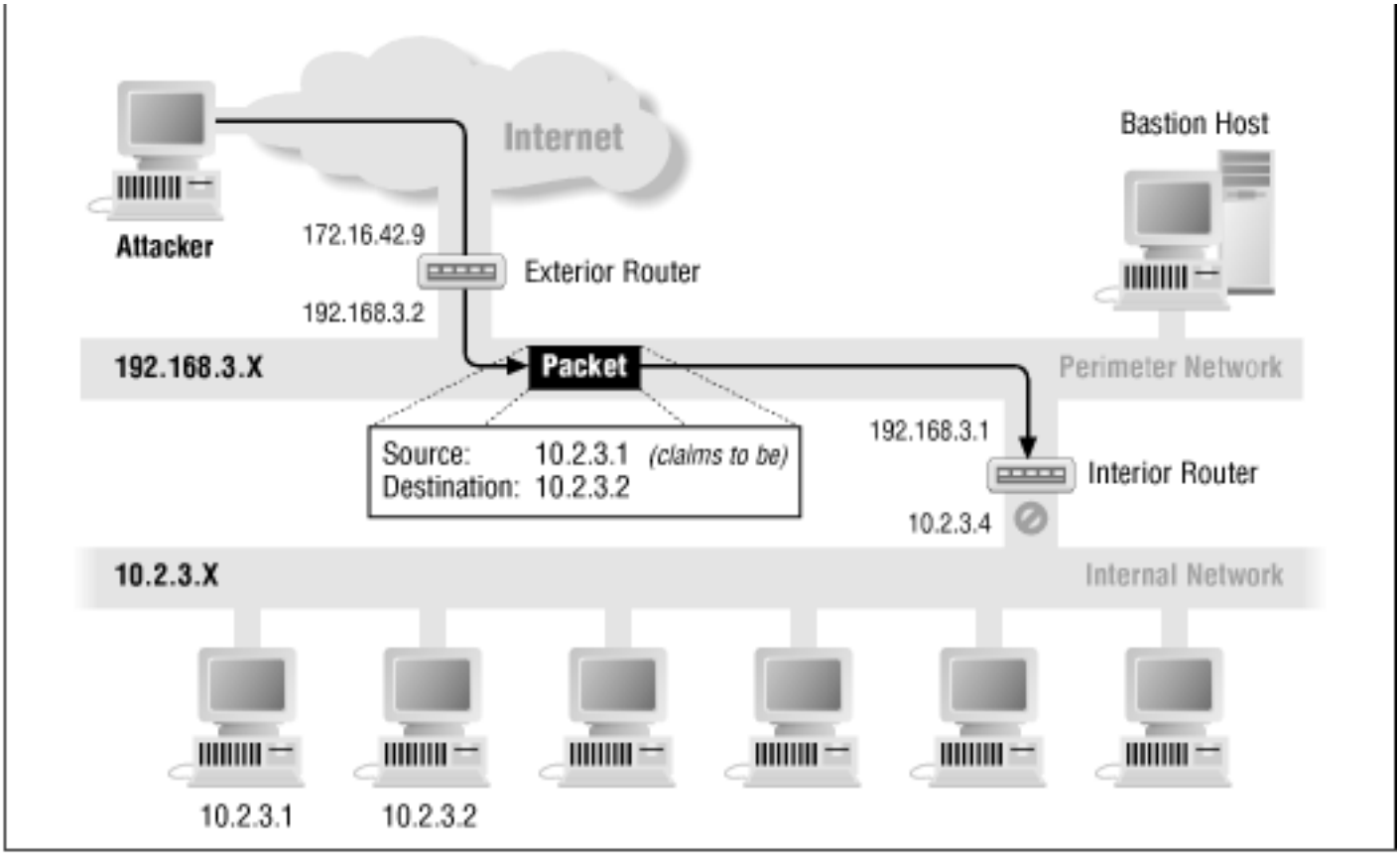
The main advantage of packet filtering is leverage: it allows you to provide, in a single place, particular protections for an entire network. Consider the Telnet service as an example. If you disallow Telnet by turning off the Telnet server on all your hosts, you still have to worry about someone in your organization installing a new machine (or reinstalling an old one) with the Telnet

server turned on. On the other hand, if Telnet is not allowed by your filtering router, such a new machine would be protected right from the start, regardless of whether or not its Telnet server was actually running. This is an example of the kind of "fail safe" stance we discussed in Chapter 3, Security Strategies.

Routers also present a useful choke point (also discussed in Chapter 3) for all of the traffic entering or leaving a network. Even if you have multiple routers for redundancy, you probably have far fewer routers, under much tighter control, than you have host machines.

Certain protections can be provided *only* by filtering routers, and then only if they are deployed in particular locations in your network. For example, it's a good idea to reject all packets that have internal source addresses - that is, packets that claim to be coming from internal machines but that are actually coming in from the outside - because such packets are usually part of address-spoofing attacks. In such attacks, an attacker is pretending to be coming from an internal machine. Decision-making of this kind can be done only in a filtering router at the perimeter of your network. Only a filtering router in that location (which is, by definition, the boundary between "inside" and "outside") is able to recognize such a packet, by looking at the source address and whether the packet came from the inside (the internal network connection) or the outside (the external network connection). Figure 6.1 illustrates this type of source address forgery.

Figure 6.1: Source address forgery



6.1.1 Advantages of Packet Filtering

Packet filtering has a number of advantages.

6.1.1.1 One screening router can help protect an entire network

One of the key advantages of packet filtering is that a single, strategically placed packet filtering router can help protect an entire network. If there is only one router that connects your site to the Internet, you gain tremendous leverage on network security, regardless of the size of your site, by doing packet filtering on that router.

6.1.1.2 Packet filtering doesn't require user knowledge or cooperation

Unlike proxying, described in Chapter 7, Proxy Systems, packet filtering doesn't require any custom software or configuration of client machines, nor does it require any special training or procedures for users. When a packet filtering router decides to let a packet through, the router is indistinguishable from a normal router. Ideally, users won't even realize it's there, unless they try to do something that is prohibited (presumably because it is a security problem) by the packet filtering router's filtering policy.

This "transparency" means that packet filtering can be done without the cooperation, and often without the knowledge, of users. The point is not that you can do this subversively, behind your users' backs (while actions like that are sometimes necessary - it all depends on the circumstances - they can be highly political). The point is that you can do packet filtering without their having to learn anything new to make it work, and without your having to depend on them to do (or not do) anything to make it work.

6.1.1.3 Packet filtering is widely available in many routers

Packet filtering capabilities are available in many hardware and software routing products, both commercial and freely available over the Internet. Most sites already have packet filtering capabilities available in the routers they use.

Most commercial router products, such as the routers from Livingston Enterprises and Cisco Systems, include packet filtering capabilities. Packet filtering capabilities are also available in a number of packages, such as Drawbridge, KarlBridge, and *screend*, that are freely distributed on the Internet; these are discussed in Appendix B, Tools.

> **NOTE:** In this book, it's impossible to give a complete list of commercial and publicly available packages, because new products are constantly being introduced and packet filtering capabilities are constantly being added to existing products. Instead, in this chapter we concentrate on discussing generic packet filtering features and capabilities, and the consequences of having - or not having - particular capabilities, so that you can make your own evaluation of the products currently available to you.

6.1.2 Disadvantages of Packet Filtering

Although packet filtering provides many advantages, there are some disadvantages to using packet filtering as well:

6.1.2.1 Current filtering tools are not perfect

Despite the widespread availability of packet filtering in various hardware and software packages, packet filtering is still not a perfect tool. The packet filtering capabilities of many of these products share, to a greater or lesser degree, common limitations:

- The packet filtering rules tend to be hard to configure. Although there is a range of difficulty, it

mostly runs from slightly mind-twisting to brain-numbingly impossible.

- Once configured, the packet filtering rules tend to be hard to test.
- The packet filtering capabilities of many of the products are incomplete, making implementation of certain types of highly desirable filters difficult or impossible.
- Like anything else, packet filtering packages may have bugs in them; these bugs are more likely than proxying bugs to result in security problems. Usually, a proxy that fails simply stops passing data, while a failed packet filtering implementation may allow packets it should have denied.

6.1.2.2 Some protocols are not well suited to packet filtering

Even with perfect packet filtering implementations, you will find that some protocols just aren't well suited to security via packet filtering, for reasons we'll discuss later in this book. Such protocols include the Berkeley "r" commands (*rcp*, *rlogin*, *rdist*, *rsh*, etc.) and RPC-based protocols such as NFS and NIS/YP. (The problems of using packet filtering to deal with these protocols are discussed in Chapter 8, Configuring Internet Services.)

6.1.2.3 Some policies can't readily be enforced by normal packet filtering routers

The information that a packet filtering router has available to it doesn't allow you to specify some rules you might like to have. For example, packets say what host they come from, but generally not what user. Therefore, you can't enforce restrictions on particular users. Similarly, packets say what port they're going to, but not what application; when you enforce restrictions on higher-level protocols, you do it by port number, hoping that nothing else is running on the port assigned to that protocol. Malicious insiders can easily subvert this kind of control.

---

7. Proxy Systems

Contents:

Proxying provides Internet access to a single host, or a very small number of hosts, while appearing to provide access to all of your hosts. The hosts that have access act as proxies for the machines that don't, doing what these machines want done.

A proxy server for a particular protocol or set of protocols runs on a dual-homed host or a bastion host: some host that the user can talk to, which can, in turn, talk to the outside world. The user's client program talks to this proxy server instead of directly to the "real" server out on the Internet. The proxy server evaluates requests from the client and decides which to pass on and which to disregard. If a request is approved, the proxy server talks to the real server on behalf of the client (thus the term "proxy"), and proceeds to relay requests from the client to the real server, and to relay the real server's answers back to the client.

As far as the user is concerned, talking to the proxy server is just like talking directly to the real server. As far as the real server is concerned, it's talking to a user on the host that is running the proxy server; it doesn't know that the user is really somewhere else.

Proxying doesn't require any special hardware, although it does require special software for most services.

> **NOTE:** Proxy systems are effective only when they are used in conjunction with some method of restricting IP-level traffic between the clients and the real servers, such as a screening router or a dual-homed host that doesn't route packets. If there is IP-level connectivity between the clients and the real servers, the clients can bypass the proxy system (and presumably so can someone from the outside).

7.1 Why Proxying?

There's no point in connecting to the Internet if your users can't access it. On the other hand, there's no safety in connecting to the Internet if there's free access between it and every host at your site. Some compromise has to be applied.

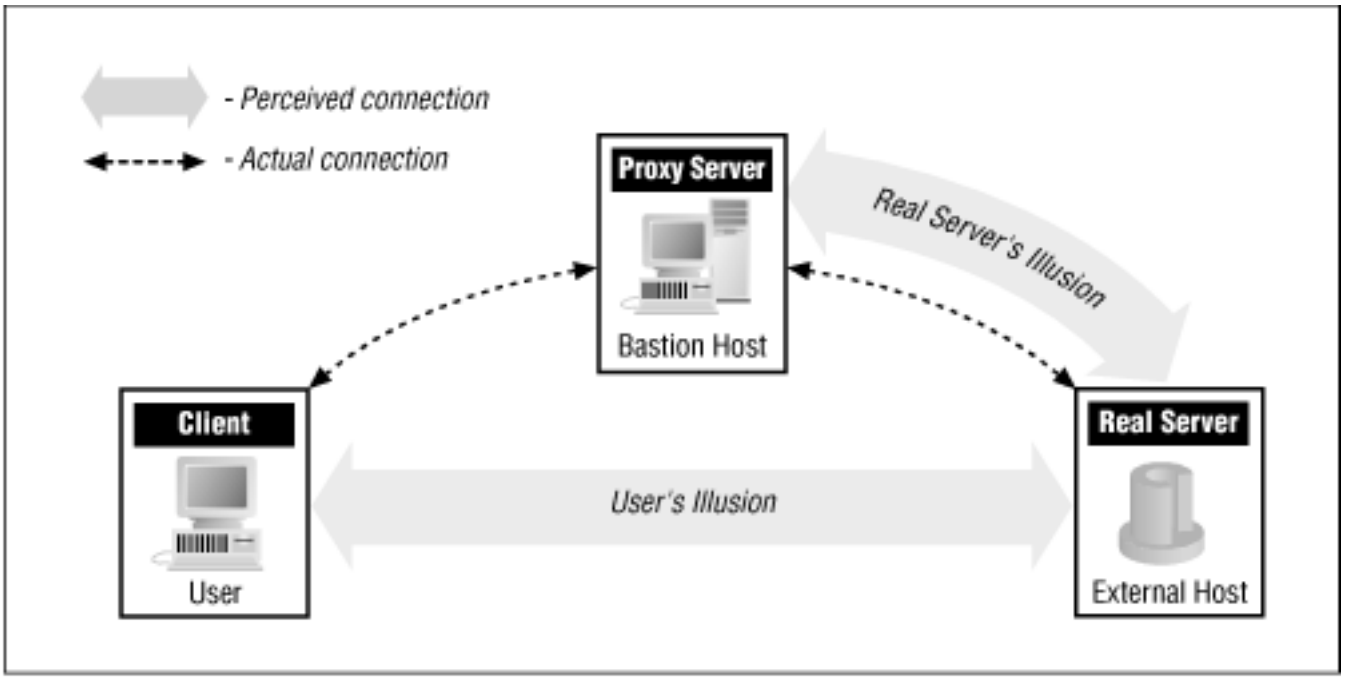
The most obvious compromise is to provide a single host with Internet access for all your users. However, this isn't a satisfactory solution because these hosts aren't transparent to users. Users who want to access network services can't do so directly. They have to log in to the dual-homed host, do all their work from there, and then somehow transfer the results of their work back to their own workstations. At best, this multiple-step process annoys users by forcing them to do multiple transfers and work without the customizations they're accustomed to.

The problem is worse at sites with multiple operating systems; if your native system is a Macintosh, and the dual-homed host is a UNIX system, the UNIX system will probably be completely foreign to you. You'll be limited to using whatever tools are available on the dual-homed host, and these tools may be completely unlike (and may seem inferior to) the tools you use on your own system.

Dual-homed hosts configured without proxies therefore tend to annoy their users and significantly reduce the benefit people get from the Internet connection. Worse, they usually don't provide adequate security; it's almost impossible to adequately secure a machine with many users, particularly when those users are explicitly trying to get to the external universe. You can't effectively limit the available tools, because your users can always transfer tools from internal machines that are the same type. For example, on a dual-homed host you can't guarantee that all file transfers will be logged because people can use their own file transfer agents that don't do logging.

Proxy systems avoid user frustration and the insecurities of a dual-homed host. They deal with user frustration by automating the interaction with the dual-homed host. Instead of requiring users to deal directly with the dual-homed host, proxy systems allow all interaction to take place behind the scenes. The user has the illusion he is dealing directly (or almost directly) with the server on the Internet that he really wants to access, with a minimum of direct interaction with the dual-homed host. Figure 7.1 illustrates the difference between reality and illusion with proxy systems.

Figure 7.1: Proxies - reality and illusion



Proxy systems deal with the insecurity problems by avoiding user logins on the dual-homed host and by forcing connections through controlled software. Because the proxy software works without requiring user logins, the host it runs on is safe from the randomness of having multiple logins. It's also impossible for anybody to install uncontrolled software to reach the Internet; the proxy acts as a control point.

7.1.1 Advantages of Proxying

There are a number of advantages to using proxy services.

7.1.1.1 Proxy services allow users to access Internet services `directly'

With the dual-homed host approach, a user needs to log into the host before using any Internet services. This is often inconvenient, and some users become so frustrated that they look for ways around the firewall. With proxy services, users think they're interacting directly with Internet services.

Of course, there's more going on behind the scenes but it's usually transparent to users. While proxy services allow users to access Internet services from their own systems, they do so without allowing packets to pass directly between the user's system and the Internet. The path is indirect, either through a dual-homed host, or through a bastion host and screening router combination.

7.1.1.2 Proxy services are good at logging

Because proxy servers understand the underlying protocol, they allow logging to be performed in a particularly effective way. For example, instead of logging all of the data transferred, an FTP proxy server logs only the commands issued and the server responses received; this results in a much smaller and more useful log.

7.1.2 Disadvantages of Proxying

There are also some disadvantages to using proxy services.

7.1.2.1 Proxy services lag behind nonproxied services

Although proxy software is widely available for the older and simpler services like FTP and Telnet, proven software for newer or less widely used services is harder to find. There's usually a distinct lag between the introduction of a service and the availability of proxying servers for it; the length of the lag depends primarily on how well the service is designed for proxying. This makes it difficult for a site to offer new services immediately as they become available. Until suitable proxy software is available, a system that needs new services may have to be placed outside the firewall, opening up potential security holes.

7.1.2.2 Proxy services may require different servers for each service

You may need a different proxy server for each protocol, because the proxy server has to understand the protocol in order to determine what to allow and disallow, and in order to masquerade as a client to the real server and as the real server to the proxy client. Collecting, installing, and configuring all these various servers can be a lot of work.

Products and packages differ greatly in the ease with which they can be configured, but making things easier in one place can make it harder in others. For example, servers that are particularly easy to configure are usually limited in flexibility; they're easy to configure because they make certain assumptions about how they're going to be used, which may or may not be correct or appropriate for your site.

7.1.2.3 Proxy services usually require modifications to clients, procedures, or both

Except for a few services designed for proxying, proxy servers require modifications to clients and/or procedures. Either kind of modification has drawbacks; people can't always use the readily available tools with their normal instructions.

Because of these modifications, proxied applications don't work as well as nonproxied applications. They tend to bend protocol specifications, and some clients and servers are less flexible than others.

7.1.2.4 Proxy services aren't workable for some services

Proxying relies on the ability to insert the proxy server between the client and the real server; that requires relatively straightforward interaction between the two. A service like *talk* that has complicated and messy interactions may never be possible to proxy (see the discussion of *talk* in Chapter 8, Configuring Internet Services).

7.1.2.5 Proxy services don't protect you from all protocol weaknesses

As a security solution, proxying relies on the ability to determine which operations in a protocol are safe. Not all protocols provide easy ways to do this. The X Window System protocol, for example, provides a large number of unsafe operations, and it's difficult to make it work while removing the unsafe operations. HTTP is designed to operate effectively with proxy servers, but it's also designed to be readily extensible, and it achieves that goal by passing data that's going to be executed. It's impossible for a proxy server to protect you from the data; it would have to understand the data being passed and determine whether it was dangerous or not.

---

8. Configuring Internet Services

Contents:

This chapter describes the major Internet services: how they work, what their packet filtering and proxying characteristics are, what their security implications are with respect to firewalls, and how to make them work with a firewall. The purpose of this chapter is to give you the information that will help you decide which services to offer at your site and to help you configure these services so they are as safe and as functional as possible in your firewall environment.

This chapter is intended primarily as a reference; it's not necessarily intended to be read in depth from start to finish, though you might learn a lot of interesting stuff by skimming the whole thing.

This chapter assumes that you are familiar with what the various Internet services are used for, and concentrates on explaining how to provide those services through a firewall. For introductory information about what particular services are used for, see Chapter 2, Internet Services.

Where we discuss the packet filtering characteristics of particular services, we use the same abstract tabular form we used to show filtering rules in Chapter 6, Packet Filtering. You'll need to translate

various abstractions like "internal," "external," and so on to appropriate values for your own configuration. See Chapter 6 for an explanation of how you can translate abstract rules to rules for particular products and packages, as well as more information on packet filtering in general.

Where we discuss the proxy characteristics of particular services, we rely on concepts and terminology discussed in Chapter 7, Proxy Systems.

Throughout this chapter, for each service, we'll show how its packets flow through a firewall. The following figures show the basic packet flow: when a service runs directly (Figure 8.1) and when a proxy service is used (Figure 8.2). The other figures in this chapter show variations of these figures for individual services. If there are no specific figures for a particular service, you can assume that these generic figures are appropriate for that service.

Figure 8.1: A generic direct service

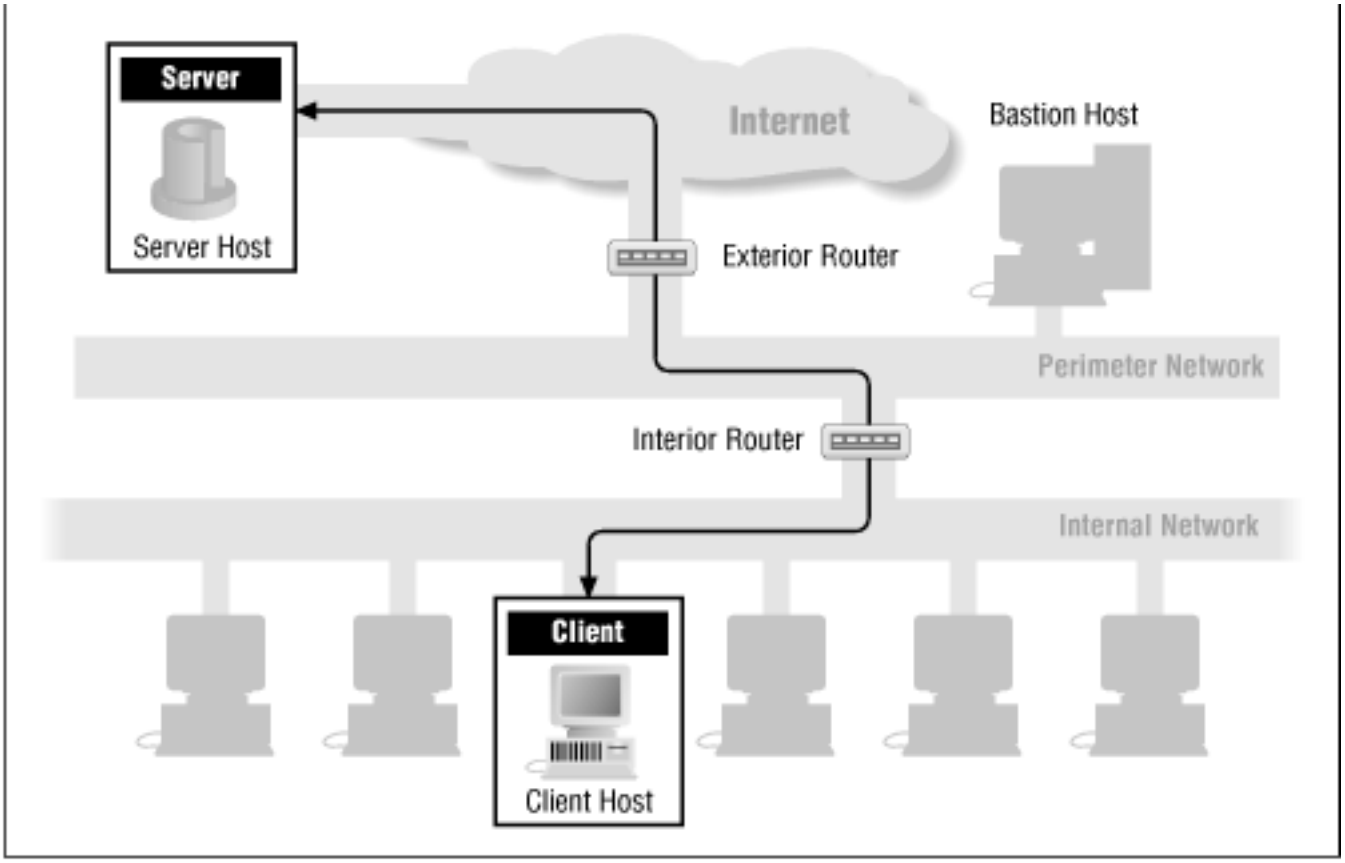
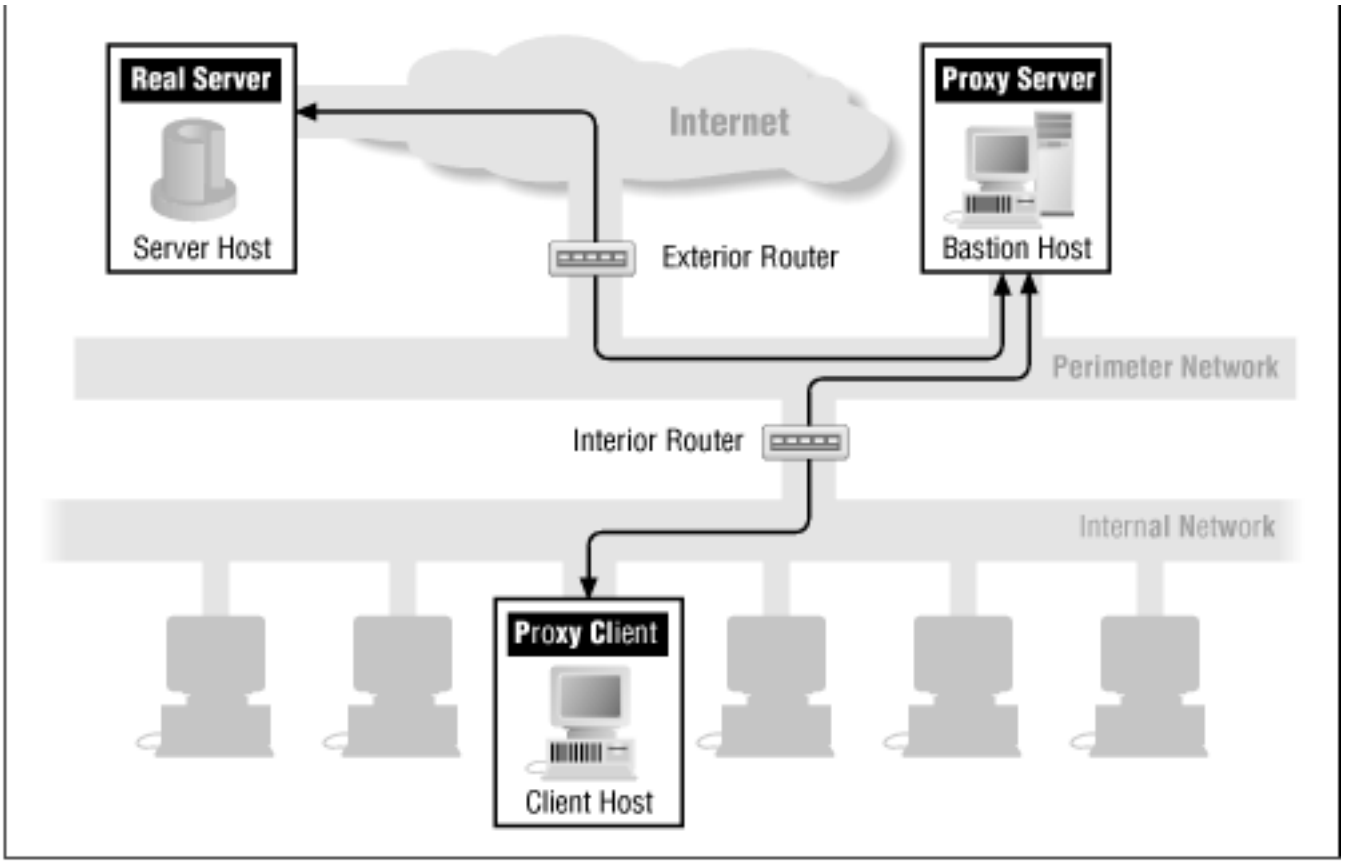


Figure 8.2: A generic proxy service



You can find information on particular resources and tools (including where to get them) in Appendix A, Resources and Appendix B, Tools.

> **NOTE:** We frequently characterize client port numbers as "a random port number above 1023." Some protocols specify this as a requirement, and on others it is merely a convention (spread to other platforms from UNIX, where ports below 1024 cannot be opened by regular users). Although it is theoretically allowable for clients to use ports below 1024 on non-UNIX platforms, it is extraordinarily rare: rare enough that many firewalls, including ones on major public sites that handle clients of all types, rely on this distinction and report never having rejected a connection because of it.

8.1 Electronic Mail

From a user's point of view, electronic mail is perhaps the most fundamental Internet service. Unfortunately, it's also one of the most vulnerable. Mail servers make extremely tempting targets, because they accept arbitrary data from arbitrary external hosts.

A mail system has three parts, which may be implemented by different programs or may be implemented by the same program, in any combination.

- A server accepts mail from external hosts or sends it to external hosts
- A delivery agent puts the mail in the correct mailbox on the local host
- A user agent lets the recipient read the mail and compose outgoing mail

Each of these parts is vulnerable for a different reason:

- The server directly accepts commands (related to delivering mail) from external hosts; for this reason, if the server isn't secure, it may end up immediately giving an attacker all the access it

has itself.

- The delivery agent needs special permissions because it needs to be able to write into every user's mailbox. Although the delivery agent doesn't need to talk to the external world, if it can be subverted somehow, the intruder obtains very broad access.
- The user agent runs as one user, and it doesn't talk to the external world, which limits its power and accessibility; however, it can often run arbitrary other programs in response to the data received.

Because it talks to the external world, the server is vulnerable to attacks in the commands it gets from the outside world; these are called *command channel attacks*. The delivery agent and the user agent don't receive the commands directly, but they may be vulnerable to dangers in the content of the mail message; these are called *data-driven attacks*. In addition, any program may be vulnerable to misuse by somebody who can control how it is executed (what arguments are used, what user is running it, what its data files are), through *command-line bugs*.

The Morris worm used a command channel attack against Sendmail; it issued a special Sendmail debugging command that allowed it to tell Sendmail to execute anything the intruder liked.

Data-driven attacks can exploit the delivery agent, the user agent, or the user. (The server generally pays no attention to the data.) For example, on UNIX machines, most versions of Sendmail use */bin/mail* as a local delivery agent. */bin/mail* is also a user agent and therefore has extensions to allow it to execute arbitrary commands. (If you have only one terminal, you're typing a mail message, and you want to see the output of some other command, it's very useful to be able to get your user agent to run the command for you.) Unfortunately, these extensions are not always disabled when */bin/mail* is running as a delivery agent, and the data in the mail message is sometimes capable of triggering them, allowing an outsider to run arbitrary commands on your system merely by sending appropriately formatted mail to you.

User agents may also be subverted even when they're acting as designed, as user agents. Today's multimedia mail readers are particularly subject to attacks of this kind, because they need to execute other programs to "display" a message for you (for example, a graphics program to display a picture or an audio program to play an audio attachment), but older mail readers were vulnerable as well. One Christmas, most of IBM's corporate network was put out of commission by a data-driven attack. Users received a message, usually from someone they knew and trusted, that said simply "run me". Because it appeared to be coming from a friend, most people did run it; when they did, it displayed a Christmas tree, and then resent itself to all of the entries in the reader's address book (personal alias file). It propagated very fast, and the network melted down under the force of trying to send millions of copies.

Finally, even if the server, the delivery agent, and the user agent are bug-free, some data-driven attacks work by subverting the user, and getting the user to perform some action the attacker wants. Sometimes that action involves running the program the attacker sent in the mail. Sometimes it's something more direct; for example, it's common for attackers to send mail to users that appears to come from their system's managers and that directs them to change their password to something specific. If the users do so, the attacker can now access their account.

How well does a firewall protect against these different types of attacks?

Command channel attacks

A firewall can protect against command channel attacks by restricting the number of machines

to which attackers can open command channels and by providing a secured server on those machines.

Data-driven attacks

There isn't much a firewall can do about data-driven attacks; the data has to be allowed through, or you won't actually be able to receive mail. In some cases it's possible to filter out "dangerous" characters in the mail addresses if you can somehow recognize them. Your best bet, though, is to run up-to-date delivery and user agents and to educate your users. Doing so will protect against most data-driven attacks. In any case, because data-driven attacks tend to be complicated and difficult to get information back from, they are relatively rare.

Command-line bugs

Command-line bugs are outside the scope of a firewall, because they can be exploited only by someone who is already able to execute commands on your system. One purpose of a firewall is to keep attackers from getting that ability.

The following sections describe the two protocols commonly used for electronic mail: SMTP and POP, as well as the MIME extension.

8.1.1 Simple Mail Transfer Protocol (SMTP)

On the Internet, electronic mail exchange between mail servers is handled with SMTP. A host's SMTP server accepts mail and examines the destination address to decide whether to deliver the mail locally or to forward it on to some other machine. If it decides to deliver the mail locally, it recodes the mail headers and delivery address into the proper form for the local delivery program, and it then hands the mail to that program. If it decides to forward the mail to another machine, it modifies the headers, and contacts that machine (usually via SMTP, but sometimes via UUCP or another protocol), and forwards the mail.

SMTP is a store-and-forward system, and such systems are well-suited to firewall applications, particularly those using proxy services. In Chapter 7, Figure 7.2 shows how mail sent from an individual user's workstation is directed initially to a gateway system before leaving the user's own network. Mail entering a network goes to a gateway system on that network before being distributed to individual users on other hosts.

8.1.1.1 SMTP for UNIX: Sendmail

The mailer most commonly used on UNIX systems is Sendmail. Sendmail is very powerful, but it also has a long and troubling history of major and minor security problems. Other mailers are available, including smail 3, MMDF, and Z-Mail, but none of them appears to be particularly more secure than a modern version of Sendmail.

Sendmail's security problems have been widely discussed, while the problems of these other mailers have not. However, the lack of public discussion about other mailers should not lead you to assume these mailers are any more secure than Sendmail. While some of them are intended to be more secure than Sendmail, there is nothing to guarantee they are. Most of these alternative mailers are apparently intended to address Sendmail's incredibly baroque configuration files and its occasional failure to comply with standards, or to provide features Sendmail doesn't have. Additional security is often an afterthought, not a design feature. These mailers are simply not as widely used as Sendmail, and therefore, they have fewer people - with both good intentions and bad - who are examining them for

security problems. This fact, more than any inherent superiority, is probably the reason for the apparently greater security of these alternative mailers; it's not that the problems aren't there, it's just that not as many people know what they are.

Sendmail is the devil that everybody knows; this is both an advantage and a disadvantage. On the one hand, problems are going to be found in Sendmail because that's where lots of people are looking for them (because lots of people use Sendmail). On the other hand, what problems are found are likely to be fixed very quickly (again, because lots of people use Sendmail). Sendmail is very actively supported on security issues.

8.1.1.2 Why does Sendmail have security problems?

One of the reasons Sendmail has security problems is that it's a very complex program. It performs several different functions, and it requires the collection of permissions necessary to perform *all* of those functions. Sendmail needs root privileges for a number of reasons; for example, these privileges allow Sendmail to:

- Listen on port 25 (a privileged port) for incoming SMTP connections.
- Operate as a particular user to read *.forward* files and *:include:* alias files owned by that user, and to run programs specified by those files.
- Execute certain kernel system calls that (in some versions of UNIX) are restricted to programs running as root, for example, to determine the amount of free disk space available to accept incoming messages.
- Protect files in the mail queue (i.e., messages in transit) from snooping by unprivileged users

These root permissions can be a liability, though, when Sendmail acts as an SMTP server; an attacker who manages to exploit a bug over an SMTP connection is now talking to a process that is running as root. The process can do essentially anything on the target machine at the attacker's bidding. Sendmail tries to be careful to give up its privileges whenever it doesn't really need them, but there have still been quite a number of privilege-related bugs over the years.

On a bastion host, it should be possible to make Sendmail run *setuid* to something other than root. You can use an alternative SMTP server (the *smap* package, discussed below) for incoming SMTP connections, so that Sendmail doesn't need to listen on port 25. You shouldn't have any users on the bastion host, so you shouldn't need the ability to operate as particular users to read protected *.forward* and *:include:* files. There probably aren't any privileged system calls on your system that are critical to Sendmail's operation (though you may need to recompile Sendmail from source to prevent it from attempting to use those calls). All you're left with is the need to keep ownership of files in the mail queue consistent, and to keep nonprivileged users (which the bastion host shouldn't have anyway) from snooping on messages in transit. Creating a *uid* just for Sendmail, and making that *uid* the owner of the queue directory should solve that problem.

Each of these tasks could probably be done in more secure ways, but this would require a major redesign and reimplementation of Sendmail, and nobody has yet stepped up to accept this challenge: among other reasons, out of fear that doing so would probably introduce new problems. Instead, we keep getting patch after patch for problem after problem, so that "the current Sendmail patch" has become something of a running joke in the network security community.

Sendmail has exhibited all of the types of general mailer vulnerabilities we discussed above. Patching

has eliminated or reduced most of them; for example, it used to be easy to exploit command-line bugs in Sendmail as an unprivileged user, but modern versions strictly limit the options available to unprivileged users. Given the program's past history, however, there are sure to be more problems yet to be discovered. Also, patches for old problems have sometimes introduced new problems.

8.1.1.3 Improving SMTP security with smap and smapd

An important step a firewall can take to improve security is to prevent attackers from speaking SMTP directly to Sendmail and, instead, to use a substitute server. Fortunately, this is feasible. SMTP stands for "Simple Mail Transport Protocol," and it really is simple. There are only about a half-dozen or so commands in the protocol that an SMTP server needs to implement in order to accept incoming mail.

You should consider adopting the *smap* package that is part of the TIS FWTK as a "wrapper" for your SMTP server (whether it is Sendmail or something else). The package includes a pair of programs called *smap* and *smapd*.

smap is a very short, simple program intended solely to handle incoming SMTP connections; unlike Sendmail, which contains about 30,000 lines of code, *smap* contains only about 700 lines. The relative simplicity of *smap* means that, unlike Sendmail, it can be easily be examined and considered in its entirety for security problems. Furthermore, it's designed with least privilege and compartmentalization in mind. The *smap* program runs without root privileges. It is started by *inetd*, which takes care of binding it to port 25 before starting it, so that *smap* doesn't need to run as root to do that. It runs *chroot*'d to a particular queue directory, and thus can't access anything outside that directory. All it does is accept incoming messages from the Internet via SMTP. It speaks the very minimum necessary set of SMTP commands, and it stores each message it receives in a separate file in the queue directory.

The second program, *smapd*, comes along regularly (typically once a minute) to process the files queued in this directory, normally by handing them to Sendmail for delivery.

The result of using this substitute SMTP server is that an attacker never has a direct SMTP connection to Sendmail or any other complex SMTP server. Such a system does not protect against data-driven security holes, but such holes would be extremely hard for any firewall system to guard against. Fortunately, data-driven holes in Sendmail seem to be very rare anyway; there has only been one instance to date.[1]

> [1] This is covered in CERT Advisory 93:16. For information on obtaining CERT Advisories, see Appendix A.

You do give up certain capabilities by using the *smap* package, because *smap* quite intentionally handles only the minimum possible set of SMTP commands. In particular, you give up the ability to do Extended SMTP (ESMTP). ESMTP supports a number of enhancements to basic SMTP, such as better handling of MIME messages (discussed below) and messages containing eight-bit data. The basic SMTP service supports only seven-bit data, and requires that eight-bit data be converted using something like *uuencode* before being transmitted, which leaves the recipient with the problem of unconverting the data. This isn't a big problem right now, because only a few clients and servers currently support ESMTP and those that do have compatibility modes that let them talk regular SMTP. ESMTP is becoming more and more common, however, and this will become more of a problem as time goes by. Of course, it's always possible that the *smap* package will be updated to support ESMTP at some point, if it becomes a critical issue. Like many other situations involving security, by using *smap* you're trading off functionality for security.

8.1.1.4 Packet filtering characteristics of SMTP

SMTP is a TCP-based service. SMTP receivers use port 25. SMTP senders use a randomly selected port above 1023.

Direc-tion	Source Addr.	Dest. Addr.	Pro-tocol	Source Port	Dest. Port	ACK Set	Notes
In	Ext	Int	TCP	>1023	25	[2]	Incoming mail, sender to recipient
Out	Int	Ext	TCP	25	>1023	Yes	Incoming mail, recipient to sender
Out	Int	Ext	TCP	>1023	25	[2]	Outgoing mail, sender to recipient
In	Ext	Int	TCP	25	>1023	Yes	Outgoing mail, recipient to sender

[2] ACK is not set on the first packet of this type (establishing connection) but will be set on the rest.

Normally, you want to configure your packet filters to allow incoming and outgoing SMTP only between external hosts and the bastion host, and between the bastion host and your internal mail servers.

Do not allow external hosts to contact random internal hosts via SMTP. As we've discussed above, only specially configured hosts can safely accept SMTP connections.

If you cannot filter on the ACK bit, you cannot safely allow outgoing SMTP connections directly from random internal hosts, as we demonstrate in the final example in Chapter 6. If you can filter on the ACK bit, you allow internal hosts to send mail to external hosts, but there isn't much advantage in doing so. Although it shouldn't increase your vulnerability, it increases the likelihood that you're going to send misformatted mail, because the mail (mis)configurations of all your machines would be visible to the external world, and the chances that all your internal machines do all the right things with mail headers (particularly in adding fully qualified domain names to addresses and "Message-ID:" lines) are low. Sending outgoing mail via the bastion host allows the bastion host the opportunity to clean up the headers before the mail is loosed upon the world.

Figure 8.3 (outbound SMTP) and Figure 8.4 (inbound SMTP) show how packet filtering works with SMTP.

Figure 8.3: Outbound SMTP

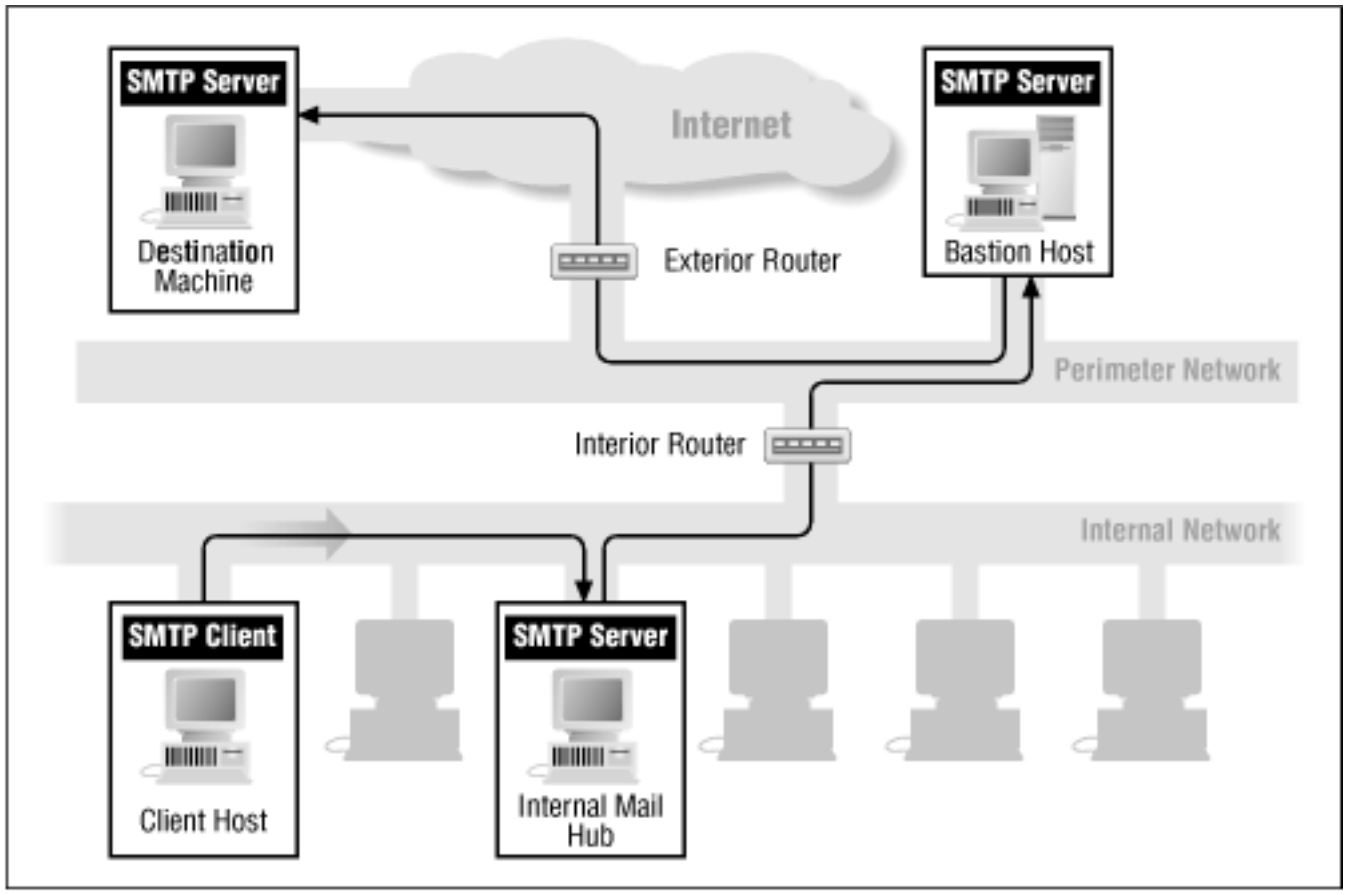
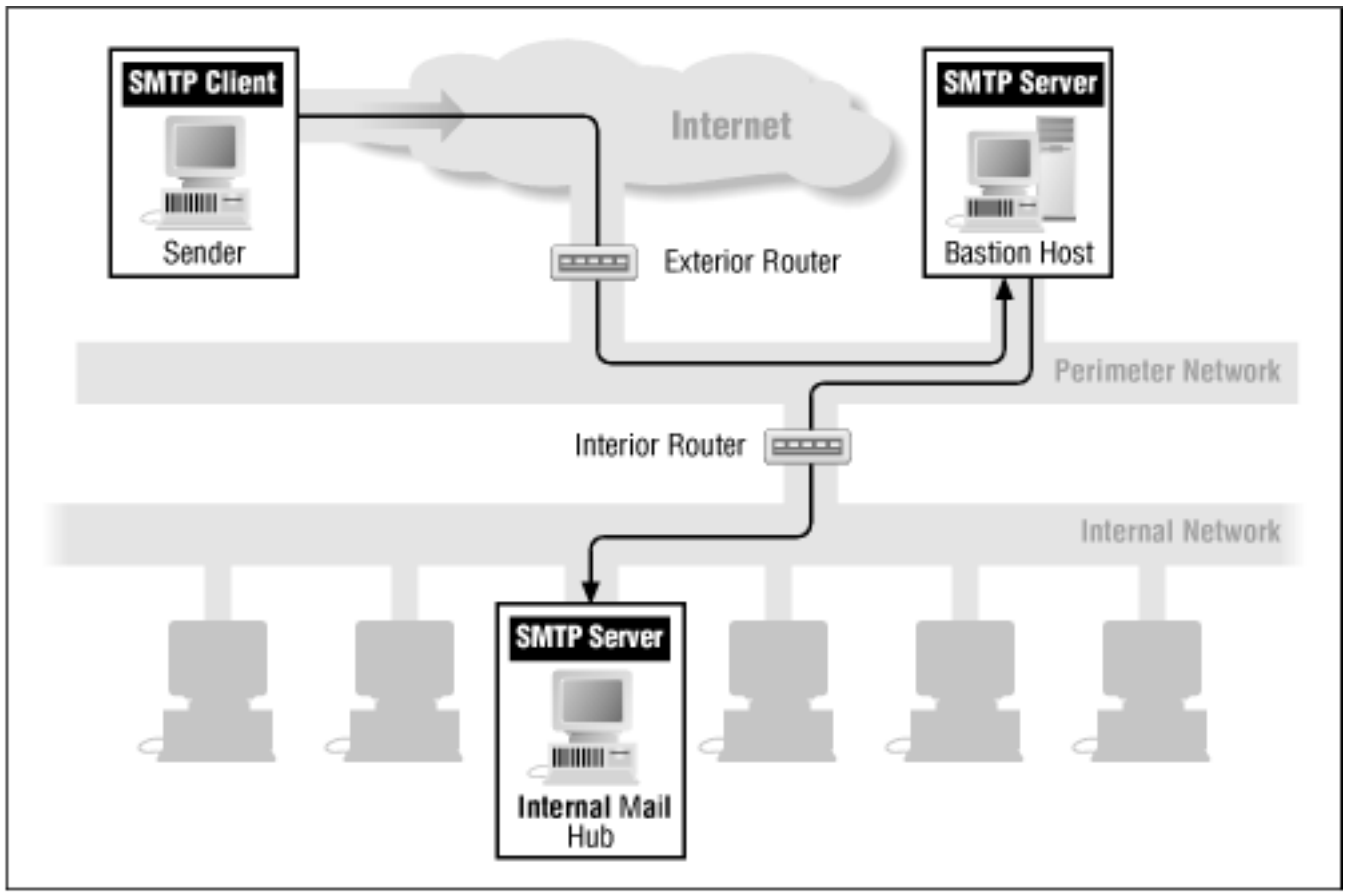


Figure 8.4: Inbound SMTP



8.1.1.5 Proxying characteristics of SMTP

Because SMTP is a store-and-forward protocol, it's inherently suited to proxying. Because any SMTP

server can be a proxy, it's rare to set up separate proxying for it. Instead, most sites direct SMTP connections to a bastion host running a secure SMTP server that is the proxy.

Dedicated firewall products that provide proxying may proxy SMTP (they can't reasonably be expected to run a full SMTP server). This is straightforward to configure, because SMTP uses a single connection. In this configuration, it's not unreasonable to continue to direct the proxied SMTP connections to a single secured SMTP server on a bastion host that acts as a second proxy. Proxying protects you from unwanted connections, but not from misuses of connections; you don't want to let external hosts talk to a standard unsecured SMTP server, even through a proxy.

8.1.1.6 Configuring SMTP to work with a firewall

Because you want to send all your mail through your bastion host, you need to configure your mail system in a special way. Here are the important steps to follow:

1. Use DNS Mail Exchange (MX) records to specify that all your incoming mail should be directed to your bastion host(s).[3]

   [3] For a detailed discussion of MX records, how they work, and how to use them, see the books *TCP/IP Network Administration* by Craig Hunt (O'Reilly & Associates, 1992) and *DNS and BIND* by Paul Albitz and Cricket Liu (O'Reilly & Associates, 1992).

2. Configure the mailer on the bastion host to check the destination address on mail it receives. If the mail is being sent to an external host, the bastion host should process the mail as usual; if the mail is to an internal host, the bastion host should simply pass the mail to an internal mail server for processing, rather than attempt to deliver the mail itself. By passing the incoming mail to a single internal server for processing, the bastion host is relieved of having to keep track of internal aliases and internal mail configuration; this means you don't have to update the mailer configuration on the bastion host nearly as often. If the bastion host passes the incoming mail to a single internal server or small list of internal servers, the filtering system can restrict SMTP connections from the bastion host to just that host or hosts, reducing the number of internal systems that can be attacked via SMTP from the bastion host if the bastion host itself is compromised.

3. Configure your internal systems to send all outgoing mail to the bastion host.

You may also want to configure your mail system so that mail is sent out with a central address, instead of with the name of an individual host, as its return address. For example, you might want mail from your users to appear as *person@bigcompany.com* not as *person@littlemachine.bigcompany.com*. Because all of the incoming mail (replies to the above addresses in outgoing mail) will be going to the bastion host in any case, this doesn't remove any necessary information. It helps to guarantee that mail will go to the bastion host correctly, even if there are problems with the MX records for individual machines, and it gives more consistent information to the recipients of the mail.

8.1.1.7 Summary recommendations for SMTP

- Use the normal store-and-forward features of SMTP to send all incoming and outgoing mail through the bastion host.
- Use packet filtering to restrict SMTP connections from external hosts to just the bastion host.

- Use packet filtering to restrict SMTP connections from the bastion host to a specific internal server or set of servers.
- Allow any internal system to send outgoing mail to the bastion host.
- Use *smap* instead of Sendmail as the SMTP server on your bastion host and probably on your internal mail server as well.
- Keep up to date with patches on delivery agents and user agents.
- Educate your users concerning mail-based scams, such as instructions to run particular programs or to change their passwords to some specified string.

8.1.2 Post Office Protocol (POP)

SMTP is used to exchange mail between servers. Normally, users access their mail as a file (directly, or using NFS or something similar) on the machine where it is delivered; however, there are sometimes reasons to use a separate protocol to distribute mail from a server to an individual user.

POP is a client-server protocol for handling user electronic mailboxes. With POP, a user's mailbox (the actual file where that user's incoming email is held for his later access) is kept on a server, rather than on the user's personal machine. The server is probably available to accept incoming mail more consistently than the user's personal machine is (particularly if the user's "personal machine" is a portable that he carries with him and that is only sometimes connected to the network). When the user wants his email, he accesses his mailbox using a client program (such as Eudora or Z-Mail) on his own machine using the POP protocol.

There are two major security issues involved in using POP across the Internet. First, be aware that standard POP clients and servers send the user's POP password over the Internet in the clear, so that anyone snooping on the connection can capture and reuse it later. In most cases, the POP password is the same as the user's login password, so that someone who snoops on it can get all of the user's privileges - not just the user's electronic mail. There are more secure variants of POP that support Kerberos (often called KPOP) and nonreusable passwords for authentication (often called APOP), but these secure variants are not widely available or widely supported. You may have trouble finding a combination of clients and servers that support these variants and that works for your site.

Second, regardless of the authentication issues, be sure to also consider the sensitivity of the email your users will be accessing over the Internet via POP. Whatever email your users access will be visible to anyone snooping on their POP sessions; you need to think about how sensitive email might be in your own environment. Many sites decide that, regardless of the authentication issues, their users' internal email is often too sensitive to risk being snooped on by someone monitoring their POP sessions. These sites decide to provide alternative access methods, such as dial-ups, that aren't as susceptible to snooping. If you provide your users with the ability to reach your network on the inside of the firewall (for example, with modems and PPP or SLIP), you can give them POP access while they're traveling without allowing it across the Internet.

8.1.2.1 Packet filtering characteristics of POP

POP is a TCP-based service. POP servers for the current version of the POP protocol (which is known as POP3 and is by far the most common version in use) use port 110. Servers for the older POP2 protocol use port 109. (POP1 was never in widespread use.) POP clients use ports above 1023.

Direction	Source Addr.	Dest. Addr.	Protocol	Source Port	Dest. Port	ACK Set	Notes
In	Ext	Int	TCP	>1023	110	[5]	Incoming POP connection, client to server
Out	Int	Ext	TCP	110	>1023	Yes	Incoming POP connection, server to client
Out	Int	Ext	TCP	>1023	110	[5]	Outgoing POP connection, client to server
In	Ext	Int	TCP	110	>1023	Yes	Outgoing POP connection, server to client

109[4]

109[4]

109[4]

109[4]

[4] Modern POP (POP3) servers use port 110; older POP2 servers use port 109.

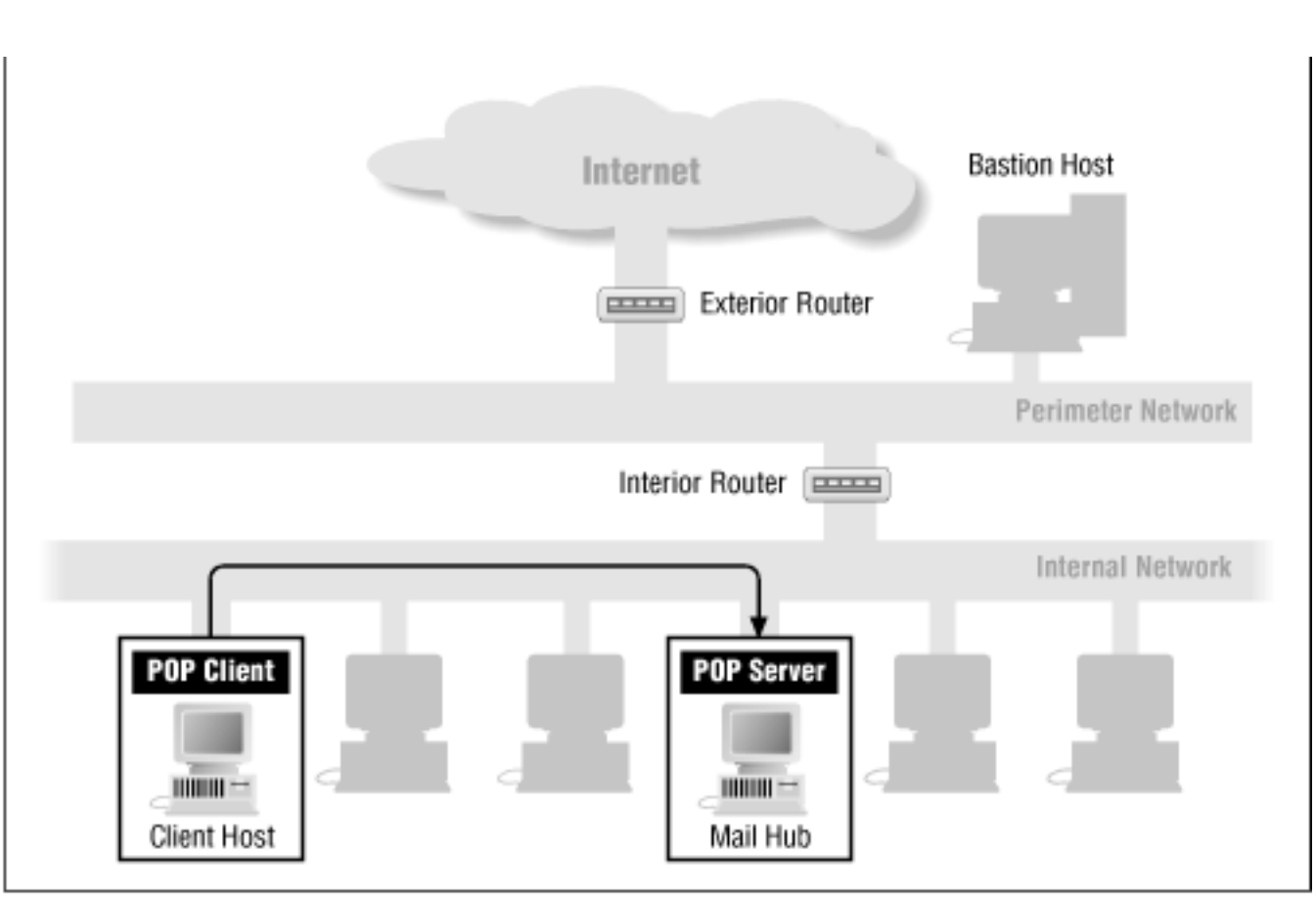
[5] ACK is not set on the first packet of this type (establishing connection) but will be set on the rest.

An outgoing POP connection would allow your users to download their mail from other sites. This is no more dangerous than allowing outgoing Telnet, and you will probably want to allow such a POP connection if there is any demand. On the other hand, POP over the Internet is rare enough that there is unlikely to be anyone interested in outgoing POP at your site, and there is no point in its use.

Incoming POP connections are those that allow people at other sites to read mail delivered for them at your site. As discussed in the previous section, you probably don't want to allow incoming POP. If you do, you should certainly limit POP connections to a POP server running on a single host. That will limit the number of vulnerable accounts and the amount of information that's being passed across the Internet. Although there are no known vulnerabilities in POP servers, if any are found, you should patch them on just one host, without worrying about all your internal hosts. Because POP requires user accounts, you don't want to run it on your normal bastion host. Although you can prevent users from logging in on POP accounts, they represent a maintenance hassle.

Figure 8.5 shows how POP works with a firewall.

Figure 8.5: POP via packet filtering



8.1.2.2 Proxying characteristics of POP

POP is straightforward to proxy, because it uses a single connection. Precompiled proxy-enabled POP clients (those that work with SOCKS, for example) are not widely available. This is mostly because, although POP is used extensively within LANs, POP across the Internet is rare. UNIX POP clients are available in source form and trivial to adapt for modified-client proxying systems. It is generally more difficult to locate the source for personal-computer implementations but no more difficult to modify them.

There is no simple way to do modified-procedure proxying for connections between internal clients and external servers, or external clients and internal servers, unless all your clients are connecting to the same server. If that's the case, then you could run a generic TCP proxy program (such as the *plug-gw* program from the TIS FWTK) on the POP3 port on your bastion host, configured to relay all connections to the single POP server; you would then configure your clients to access the "POP server" (really the proxy program) on the bastion host.

If you do need to provide access to multiple POP servers and can ensure that all client connections from a given IP address or domain should be directed to a particular server, you could set up a more complex configuration with the *plug-gw* program to direct connections to the appropriate server based on where the connection request is coming from. If multiple users on the same client machine or machines need to access different POP servers across the firewall, however, there's no simple way to do it with the code that's currently available. It is possible to write a custom POP proxy server to run on the bastion host that authenticated the user, decided which server that user needed to talk to, opened a connection and authenticated the user to that server, and then sat back and played the traditional proxy server "pass-through" role; it would be difficult to make this work with nonreusable passwords, though.

8.1.2.3 Summary recommendations for POP

- Do not allow your users to transfer your site's mail over the Internet via POP, unless you can do so without revealing reusable passwords, and unless either you aren't concerned about the sensitivity of the mail itself or you have an encrypted channel to transfer it over.
- If you have users who wish to transfer mail from other sites via POP, allow it via packet filtering, perhaps restricted to connections from specific sites or to specific hosts on your end.
- Proxying would be an effective solution for some POP problems, but you might need to do at least minor code modifications.

8.1.3 Multipurpose Internet Mail Extensions (MIME)

MIME is a set of extensions to the basic Internet electronic mail message format supporting things like:

- Non-ASCII character sets
- Nontext data such as pictures and audio segments
- So-called "rich text" messages (messages containing formatted text, with different fonts and so on, rather than simple single-font unformatted text)
- Multipart messages (messages containing multiple pieces, each piece in its own format).

MIME support is mostly a client issue; to mail servers and transport systems, MIME messages are generally just another message. The question is whether or not a given client can generate outgoing MIME messages, and whether or not it can recognize and cope with incoming MIME messages.

The MIME standards define certain basic data types, such as plain text, formatted text, standard audio, and so on. MIME is designed to be extensible, so that new data types can be added as necessary. MIME-capable mail clients generally understand certain data types (often only multipart messages and plain text), and rely on other programs to handle other data types (for example, graphics programs to display images, and sound programs to play audio clips). The clients generally have a list of external programs to run for particular types of data; this list can be extended or modified by the user.

Ironically, the biggest use of MIME hasn't been in mail systems, but in World Wide Web browsers. Most mail systems have little or no MIME support, but MIME is a key service upon which the World Wide Web is built. Every WWW server uses MIME to describe the format of every WWW page it hands to a client; every WWW client uses MIME to determine how to display or otherwise process the data it receives.

Because MIME is used more extensively for WWW support than for electronic mail, we discuss it below in the section on WWW, even though it is theoretically email-related. If you do have email clients that support MIME, they will be subject to the same vulnerabilities discussed in "What can a malicious server do to your clients?" in the section on the World Wide Web later in this chapter.

One difference between MIME support in email clients and WWW clients is how data is obtained. With a WWW client, the user chooses what data to access; with email, the user accesses whatever anybody sends them. In theory, email clients are more vulnerable, because you can't control what other people send you by email. In practice, however, the difference isn't that important, because it's fairly easy to lure a WWW user into accessing whatever you want them to access. Either way, you need to carefully control what data types your clients understand and how they process that data. See the full discussion of this issue in the WWW section.

9. Two Sample Firewalls

Contents:

In this chapter, we describe two sample configurations for basic firewalls. Almost any real firewall is going to be more complex than those described in this chapter, but this presentation should give you some idea of the tasks involved in building a firewall and how the various pieces fit together.

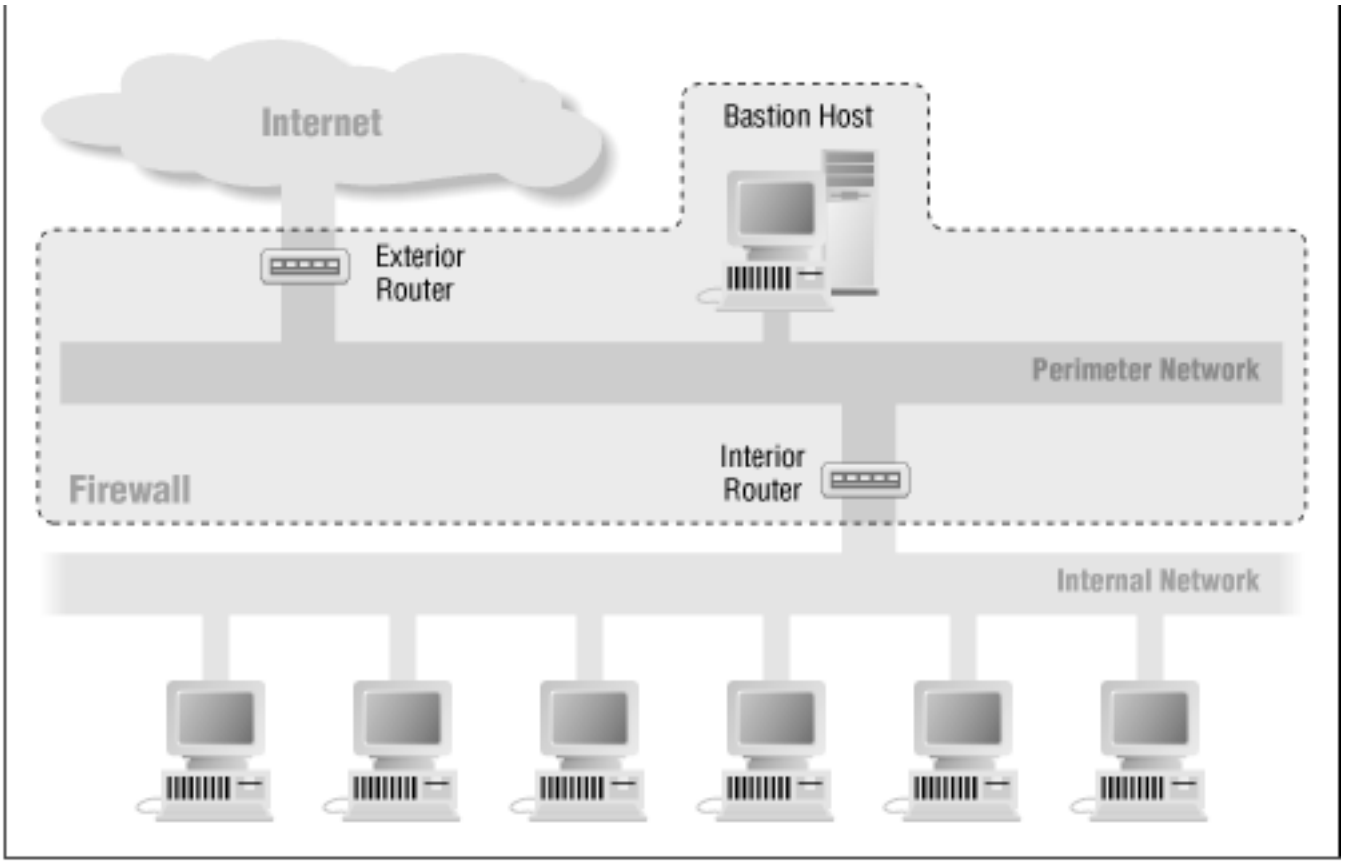
> **NOTE:** We want to emphasize that these examples are just that: examples. You shouldn't blindly implement one of these examples, without first taking the time to understand your own needs, your environment, and the implications and complications of the services you want your firewall to provide.

The services that we're going to provide through these sample firewalls are just the basics: terminal access, file transfer, electronic mail, Usenet news, the World Wide Web, and DNS. See Chapter 8, Configuring Internet Services for a full discussion of these and other services.

9.1 Screened Subnet Architecture

The screened subnet architecture, described in Chapter 4, Firewall Design and shown in Figure 9.1, is probably the most common do-it-yourself firewall architecture. This architecture provides good security (including multiple layers of redundancy) at what most sites feel is a reasonable cost.

Figure 9.1: Screened subnet architecture



There are two-router and single-router variations of the screened subnet architecture. Basically, you can use either a pair of two-interface routers or a single three-interface router. The single-router screened subnet architecture works about as well as the two-router screened subnet architecture, and is often somewhat cheaper. However, you need to use a router that can handle both inbound and outbound packet filtering on each interface. (See the discussion of this point in Chapter 6, Packet Filtering.) We're going to use a two-router architecture as our example in this section because it is conceptually simpler.

The components of this type of firewall include the following:

- Perimeter network
- Exterior router
- Interior router
- Bastion host

Let's review briefly the purposes of these various components, presented originally in Chapter 4.

Perimeter network

Isolates your bastion host from your internal network, so a security breach on the bastion host won't immediately affect your internal network.

Exterior router

Connects your site to the outside world. If possible, the exterior router also provides at least some protection for the bastion host, interior router, and internal network. (This isn't always possible, because some sites use exterior routers that are managed by their network service providers and are therefore beyond the site's control.)

Interior router

Protects the internal network from the world and from the site's own bastion host.

Bastion host

Serves as the site's main point of contact with the outside world. (It should be set up according to the guidelines in Chapter 5, Bastion Hosts.)

In addition to the machines that make up the firewall itself, assume there are machines on the internal network (internal hosts) that fulfill the following roles. (Note that any given internal machine might fill any, or even all, of these roles.)

- Mail server
- Usenet news server
- DNS server
- Client for various Internet services

Each of these internal services is provided directly (via packet filtering) or indirectly (via proxy servers running on the bastion host).

We're going to assume (at least for the purposes of this example) that internal users in our system are trusted not to actively try to circumvent the firewall, and that there is no particular need to monitor or log their Internet activities.

We're also going to assume that you are using properly assigned and routed IP addresses (that is, addresses assigned to your site, and properly routed and advertised to the rest of the Internet by your service provider) for your internal and perimeter networks. If you aren't, you have no choice but to use proxies, because you can't allow packets with those unassigned IP addresses out onto the Internet; even if you did, replies would have no way to come back to you.

Finally, we're going to assume you're using separate network numbers for your perimeter net and internal net, so that you can detect forged packets easily. (See the discussion in "Risks of Filtering by Source Address" in Chapter 6.)

9.1.1 Service Configuration

Given the architecture we've just described, how do we provide the basic Internet services?

9.1.1.1 Telnet

Outgoing Telnet could be provided through packet filtering or through proxies. Which approach should we use?

Proxying will require either modified clients or modified user procedures; either one will be tedious to implement for somebody. Proxying would allow us to restrict or monitor Telnet usage by user by forcing our users to authenticate to a proxy server before completing their requests. However, remember that we have decided to assume internal users are trustworthy, so there is no need for authentication. Proxying would be necessary if we were using unassigned or unadvertised IP addresses internally, but that's not the case here, either. Because we have trustworthy users and proper IP addresses, proxying for Telnet doesn't provide any advantage over packet filtering, and it's more difficult to set up and maintain. Therefore, for this example we're going to provide outgoing Telnet via

packet filtering.

Incoming Telnet is considerably more difficult to provide safely and conveniently. If it were necessary, incoming Telnet could be provided on the bastion host using extra authentication. However, for a site that is looking for a simple configuration, the reasonable thing to do is to disallow incoming Telnet altogether. That's what we'll do for this example.

9.1.1.2 FTP

Unlike Telnet, FTP doesn't lend itself to a pure packet filtering solution. Because normal mode FTP requires an incoming connection to an arbitrary port over 1023, trying to allow it without doing anything else gives attackers access to all kinds of services running on our internal systems. That leaves us two choices:

- Support passive mode via packet filtering
- Support normal mode via proxies

In either case, the standard FTP clients shipped with UNIX operating systems, and most of the popular, publicly available clients for personal computers, won't work the normal way. If we use the TIS FWTK *ftp-gw* proxy gateway (described in Chapter 7, Proxy Systems), we can use unmodified clients, but at the cost of teaching the users to follow special procedures. Some popular FTP clients - the ones built in to Web browsers like Netscape Navigator or Mosaic - do use passive mode without being modified, but then again, not all servers support it well.

A reasonable compromise would be to use packet filtering and proxies, using a proxying gateway like *ftp-gw* from the TIS FWTK that doesn't require modification of the clients. Clients that support passive mode will work via the packet filters. On platforms where we can easily replace the provided clients, we can provide passive mode clients; on platforms where we can't easily replace the clients, we can modify user procedures.

As we've said, if we wanted to monitor FTP usage, or if we were using an unassigned or unrouted network number, we'd have to do exclusively proxying, but that's not the case here. We also might want to use proxying exclusively if we decided to hide DNS data - it would save us the trouble of faking data for double-reverse lookups - but hiding DNS data is more trouble than it's worth. In a situation where proxying is used exclusively, we'd have to reconsider which proxy server to use. Requiring users to modify their procedures makes sense here, because there are no other options. The balance might come out differently if everybody were proxying.

Be aware that in order to use the TIS FWTK *ftp-gw* proxy server on your bastion host, your packet filtering will have to allow TCP connections from ports above 1023 on your bastion host to ports above 1023 on internal hosts, and from port 20 on external hosts to ports above 1023 on your bastion host, for FTP data channels. (See the discussion of FTP in Chapter 8.) This means that someone who breaks in to the bastion host could easily connect to any server on any internal host that uses a TCP port above 1023 (for example, an X11 server on port 6000). Further, any servers that are using such ports (that is, TCP ports above 1023) on the bastion host itself are also vulnerable. For this reason, if you allow these FTP data connections at all for proxying, you probably want to explicitly block access to internal systems to TCP ports above 1023 where you know, or have good cause to suspect, that servers might be listening. At the very least, such ports probably include 6000 through around 6003 (assuming four or fewer real or virtual X11 servers per machine; see the discussion of X11 in Chapter 8).

Keep in mind, however, that blocking specific ports, rather than blocking all ports by default and then allowing specific ports, is generally a dangerous strategy. It's hard to develop and maintain a complete list of ports that need to be blocked at your site. It would be better to block everything by default and then allow only specific ports, but you can't do that with standard (nonpassive) FTP, because of the way it works.

Allowing passive-mode FTP with packet filtering is a fairly liberal approach, because it allows pretty much any connection to go through as long as it is initiated by the inside. The number of servers above port 1023 is very large. This has some advantages (it lets users access nonstandard HTTP servers, for example), but it may also allow users to access all kinds of services that are unsafe. In allowing it, we're assuming that our users are not only well-intentioned, they're also capable of telling the difference between a safe and an unsafe connection, or at least avoiding the temptation to do unexpected things.

Because we've denied incoming Telnet, it makes sense to deny incoming user FTP as well. Both services require approximately the same security measures, and making one of them available would make configuring the other trivial.

Incoming anonymous FTP is a different matter, and we'll provide it. Because we're not a major Internet service provider, and because we're using the TIS FWTK already anyway, we can go for security over features and use the TIS FWTK anonymous FTP server. If we were going to provide major anonymous FTP, we'd probably want to use the more feature-filled *wuarchive* FTP server (see Chapter 8 for a description of its capabilities), and run it on a machine that was not the main bastion host but that was still a bastion host on the perimeter network.

9.1.1.3 SMTP

There aren't many options for SMTP in any configuration. We want all external SMTP connections to go to a single machine with a secured SMTP server, and we don't trust random internal machines to have safe SMTP servers. That means we'll put a secured SMTP server on the bastion host and use DNS MX records to direct all incoming mail to the bastion host, which will then pass all the incoming mail to a single internal secured SMTP server.

What other options do we have? We could put a secure SMTP server on the internal mail server and direct incoming mail to it, but if that SMTP server were compromised, the entire internal net would then be at risk. Alternatively, we could have the bastion host send mail directly to machines on the internal network, but once again, we'd be increasing our vulnerability if the SMTP server on the bastion host were compromised. The compromised bastion host would be speaking to untrustworthy internal SMTP servers, and compromise of the internal net would quickly follow. If the bastion host can speak only to the internal mail server, that narrows the possible attacks it can make; the internal mail server can run a secured SMTP server. Furthermore, by making this choice, messy maintenance tasks are transferred from the security-critical bastion host to the less vulnerable internal mail server.

How about outgoing mail? It would probably be safe to allow internal machines to send mail directly to the outside world, but doing so creates a maintenance headache. (You have to watch the mail configuration on all the internal machines.) Besides, doing this opens another direct connection between the internal and external networks. There aren't any known ways for an SMTP server to attack an SMTP client, but stranger things have happened.

Allowing internal machines to send mail directly to the outside world doesn't seem to bring much advantage either. The only difference it makes is that we'd be able to send mail (but not receive it)

when the bastion host is down.

No matter what decision we make, we'll have to configure our mail clients. (Unlike FTP and Telnet, SMTP does not work as installed without modifying configuration files.) In addition, the work to direct the mail to a server is less than the work to correctly send it to the external universe.

The only real question is whether to direct the outgoing mail from the internal machines to the internal mail server or to the bastion host. Directing it to the internal mail server has the same advantages for incoming mail, but the advantages are much smaller. Sending outgoing mail presents little risk to start with, so reducing the risk isn't worth much. Sending outgoing mail also doesn't involve configurations that change often enough to make it worrisome to maintain the bastion host. On the other hand, the cost of going through the internal mail server - in complexity, possible points of failure, and delay - is just as high for outgoing mail as for incoming mail. It no longer looks like an attractive bargain, so we'll direct the outgoing mail to the bastion host without using the internal mail server as an intermediary.

We'll set up SMTP, as outlined in Chapter 8, with the bastion host acting as a middleman for incoming and outgoing mail. Here's what to do:

- Publish DNS MX records that direct incoming mail for the site to the bastion host.
- Configure internal machines to send all outgoing mail to the bastion host.
- Configure the bastion host to send all incoming mail to a single internal mail server and to send outgoing mail directly to destination machines.

For this example, we're going to assume there is a single internal mail server for incoming mail, and that internal machines send their outgoing mail directly to the bastion host (rather than indirectly via the mail server).

9.1.1.4 NNTP

As we've discussed in Chapter 8, the most practical way to set up NNTP across a firewall is to allow your NNTP service provider(s) to talk directly to your internal Usenet news host, and vice versa. We'd need an overwhelming reason to do something else, and it's hard to imagine one. Even if we didn't have an existing internal news host, building one isn't any harder than building a news server on the bastion host, and it's safer by a vast margin. News servers fail with dreary regularity; while the problems usually aren't security-related, you still don't want to install anything high-maintenance on a bastion host.

For this example, we're going to assume a single external NNTP newsfeed.

9.1.1.5 HTTP

As with the other services, for HTTP we can use either packet filtering or proxy servers to provide service to internal clients. Packet filtering will allow our users to access HTTP servers only on standard ports; proxying will allow them to reach all HTTP servers. Which approach should we take?

For Telnet and FTP, the major drawback of proxying was that it required special clients; for HTTP, though, standard browsers support HTTP proxying. Proxying will increase configuration overhead, but that price seems fair for the increased abilities it offers. On the other hand, we've already decided to allow internal hosts to create outgoing connections to any port at or above 1024 in order to allow

passive-mode FTP directly from internal hosts to external hosts. That will allow access to almost any HTTP server (any one using a port at or above 1024, anyway). Using pure packet filtering would lose us only HTTP servers at nonstandard ports below 1024, and those ports are all supposed to be reserved anyway.

If we use the CERN HTTP server as our proxy server, the server can also cache Web pages. Doing so can significantly improve performance for all of the following:

HTTP clients

They obtain pages from the cache over the internal network, rather than from the original server over our Internet connection, which is probably much slower.

Non-HTTP clients

The HTTP clients won't be using so much of the bandwidth of our Internet connection.

HTTP servers at other sites

They will only get one request from our site for a given page, rather than multiple requests.

The CERN server will also allow us to provide HTTP service to external sites, so we'll take advantage of that to publish our site's own public WWW pages.

HTTP proxying is also trivial to add once you have other things proxied via SOCKS or TIS FWTK. If we had SOCKS running, it would be very tempting to simply proxy through it, because SOCKS is one of the systems that many of the browsers support. All of the proxy servers we've already decided to run are part of the TIS FWTK, however, and we'd have to install SOCKS just for HTTP if we wanted to use it. If we used the TIS FWTK HTTP proxy, we'd have to get the users to modify URLs they use. In this case, the proxy servers wouldn't be able to cut and paste out of their electronic mail when people tell them about cool new things. Neither SOCKS nor TIS FWTK makes an attractive option in this situation.

However, both proxying (through the CERN server) and packet filtering appear to be attractive and reasonable choices. Proxying would definitely be preferable if we weren't already providing passive-mode FTP directly (which gives our users the ability to talk to servers on any TCP port above 1023). On the other hand, packet filtering would definitely be preferable if we wanted to use clients that didn't come with built-in support for proxying, or if we wanted not to provide an HTTP server and had no other services being proxied.

For this example, we're going to assume that we're providing HTTP service to internal clients via a CERN proxy server running on the bastion host, and that we're using the same server to publish our public WWW pages to the world.

9.1.1.6 DNS

As discussed in Chapter 8, DNS across a firewall is best provided with a pair of servers: one on the bastion host, the other on an internal host. Like NNTP, DNS presents a situation in which the number of rational solutions is clearly limited. We need to decide whether to use separate internal and external servers to do information hiding, or whether we should allow the external world to see all of our host data. By deciding to allow direct passive-mode FTP, we've already made that decision indirectly. Direct passive-mode FTP would require intricate DNS setup to support information hiding and still provide valid data for the internal hosts that are FTP clients.

For this example, we're going to assume the DNS server on the bastion host is a secondary server for our domain, and the primary server is on an internal host. We're not going to do any DNS information hiding.

9.1.2 Packet Filtering Rules

Based on the configuration decisions we've made in the previous sections, let's look at the packet filtering rules necessary to support this configuration. We assume an "ideal" router (as discussed in "Choosing a Packet Filtering Router" in Chapter 6). If our router were less than ideal in terms of capabilities, we'd need to modify these rules accordingly, probably at the cost of security. We might have to rethink several crucial decisions entirely. For example, if we couldn't filter on the ACK bit, direct outbound passive-mode FTP would no longer be acceptably safe, and a lot of our other decisions have used that as a major factor. (See Chapter 6 for a full discussion of packet filtering capabilities, and the implications of not having particular capabilities.)

In the packet filtering rules presented below, we assume that the filtering system:

- Can distinguish between incoming and outgoing packets
- Can filter on source address, destination address, packet type (TCP or UDP), source port, and destination port
- Can filter on whether or not the ACK bit is set (for TCP packets)
- Applies rules in the order listed

9.1.2.1 Interior Router

The purpose of the interior router is to protect the internal network from the Internet and from your own bastion host. The interior router needs the following rules to support the outlined configuration. Explanations of each rule follow the table.

Rule	Direction	Source Address	Dest. Address	Protocol	Source Port	Dest. Port	ACK Set	Action
Spoof	In	Internal	Any	Any	Any	Any	Any	Deny
Telnet-1	Out	Internal	Any	TCP	>1023	23	Any	Permit
Telnet-2	In	Any	Internal	TCP	23	>1023	Yes	Permit
FTP-1	Out	Internal	Any	TCP	>1023	21	Any	Permit
FTP-2	In	Any	Internal	TCP	21	>1023	Yes	Permit
FTP-3	Out	Internal	Any	TCP	>1023	>1023	Any	Permit
FTP-4	In	Any	Internal	TCP	>1023	>1023	Yes	Permit
FTP-5	Out	Internal	Bastion	TCP	>1023	21	Any	Permit
FTP-6	In	Bastion	Internal	TCP	21	>1023	Yes	Permit
FTP-7	In	Bastion	Internal	TCP	Any	6000-6003	Any	Deny
FTP-8	In	Bastion	Internal	TCP	>1023	>1023	Any	Permit
FTP-9	Out	Internal	Bastion	TCP	>1023	>1023	Yes	Permit
SMTP-1	Out	Internal	Bastion	TCP	>1023	25	Any	Permit
SMTP-2	In	Bastion	Internal	TCP	25	>1023	Yes	Permit

SMTP-3	In	Bastion	Internal SMTP server	TCP	>1023	25	Any	Permit
SMTP-4	Out	Internal SMTP server	Bastion	TCP	25	>1023	Yes	Permit
NNTP-1	Out	Internal NNTP server	NNTP feed server	TCP	>1023	119	Any	Permit
NNTP-2	In	NNTP feed server	Internal NNTP server	TCP	119	>1023	Yes	Permit
NNTP-3	In	NNTP feed server	Internal NNTP server	TCP	>1023	119	Any	Permit
NNTP-4	Out	Internal NNTP server	NNTP feed server	TCP	119	>1023	Yes	Permit
HTTP-1	Out	Internal	Bastion	TCP	>1023	80	Any	Permit
HTTP-2	In	Bastion	Internal	TCP	80	>1023	Yes	Permit
DNS-1	Out	Internal DNS server	Bastion	UDP	53	53	[1]	Permit
DNS-2	In	Bastion	Internal DNS server	UDP	53	53	[1]	Permit
DNS-3	Out	Internal DNS server	Bastion	TCP	>1023	53	Any	Permit
DNS-4	In	Bastion	Internal DNS server	TCP	53	>1023	Yes	Permit
DNS-5	In	Bastion	Internal DNS server	TCP	>1023	53	Any	Permit
DNS-6	Out	Internal DNS server	Bastion	TCP	53	>1023	Yes	Permit
Default-1	Out	Any	Any	Any	Any	Any	Any	Deny
Default-2	In	Any	Any	Any	Any	Any	Any	Deny

[1] UDP packets do not have ACK bits.

Here is some additional information about each set of rules in this table:

Spoof

Blocks incoming packets that claim to have internal IP addresses (that is, forged packets presumably sent by an attacker).

Telnet-1 and Telnet-2

Allow outgoing Telnet connections.

FTP-1 and FTP-2

Allow outgoing connections to FTP servers, for use by passive-mode internal clients that are interacting with those servers directly.

FTP-3 and FTP-4

Allow the FTP data channel connections from passive-mode internal clients to external FTP servers. Note that these rules actually allow all connections from internal TCP ports above 1023 to external TCP ports above 1023. That may be more than you want to allow, but there's no way to cover passive-mode FTP with anything less broad, and connections are at least restricted to those opened from the inside.

FTP-5 and FTP-6

Allow normal (nonpassive-mode) internal FTP clients to open an FTP command channel to the proxy FTP server on the bastion host. Note that these rules are actually redundant if you have rules FTP-1 and FTP-2 in place earlier in the list, because "Bastion" as a source or destination (covered by rules FTP-5 and FTP-6) is a subset of "All" (covered by rules FTP-1 and FTP-2). Having these redundant rules is going to impose a slight performance cost, but makes the rule set easier to understand. It also makes it possible to change rules FTP-1 and FTP-2 (e.g., if you decide you don't want to support passive-mode clients) without accidentally breaking

normal-mode client access to the proxy server.

FTP-7 through FTP-9

Allow FTP data connections from the proxy server on the bastion host to nonpassive internal clients. The FTP-7 rule prevents an attacker who has gained access to the bastion host from attacking internal X11 servers via the hole created by rules FTP-8 and FTP-9. If you have other servers internally listening for connections on TCP ports above 1023, you should add similar rules for them. Note that trying to list things that should be denied (as in rule FTP-7) is generally a losing proposition, because your list will almost always be incomplete somehow (e.g., because you overlooked or didn't know about some internal service, or because the service was added after the filters were established). However, it's the best you can do in this situation to support normal-mode FTP clients.

SMTP-1 and SMTP-2

Allow outgoing mail from internal hosts to the bastion host.

SMTP-3 and SMTP-4

Allow incoming mail from the bastion host to your internal mail server.

NNTP-1 and NNTP-2

Allow outgoing Usenet news from your news server to your service provider's news server.

NNTP-3 and NNTP-4

Allow incoming Usenet news from your service provider's news server to your news server.

HTTP-1 and HTTP-2

Allow internal HTTP clients to connect to the HTTP proxy server on your bastion host.

DNS-1

Allows UDP-based DNS queries and answers from the internal DNS server to the bastion host DNS server.

DNS-2

Allows UDP-based DNS queries and answers from the bastion host DNS server to the internal DNS server.

DNS-3 and DNS-4

Allow TCP-based DNS queries from the bastion host DNS server to the internal DNS server, as well as answers to those queries. Also allow zone transfers in which the bastion host DNS server is the secondary server and the internal DNS server is the primary server.

DNS-5 and DNS-6

Allow TCP-based DNS queries from the internal DNS server to the bastion host DNS servers, as well as answers to those queries. Also allow zone transfers in which the bastion host DNS server is the primary server and the internal DNS server is the secondary server.

Default-1 and Default-2

Block all packets not specifically allowed by one of the preceding rules.

How you translate these abstract rules into specific rules for your particular filtering system depends

on the syntax used by your system. Some systems allow you to enter the rules as a single table, much as we show here in this table. Other systems require you to specify rules for incoming and outgoing packets in separate rule sets. Splitting these rules between incoming and outgoing packets is not a problem, as long as you preserve the order for rules of each type; that is, as long as all the incoming rules stay in the same order relative to each other, and all the outgoing rules stay in the same order relative to each other.

9.1.2.2 Exterior Router

The purpose of the exterior router is twofold:

- To connect the perimeter net (and thus your site) to the outside world
- To protect the perimeter net and the internal net against the outside world

In many circumstances, only the former purpose is possible, because the exterior router is often provided and managed by your network service provider. That provider may be unable or unwilling to set up and maintain packet filtering rules on the exterior router (and unable or unwilling to let you do it yourself).

If you can set up filtering on the exterior router, it's a good idea to do so. If nothing else, this can serve as a backup to some of the filtering on the interior router. For this example, you would need to establish the following rules:

Rule	Direction	Source Address	Dest. Address	Protocol	Source Port	Dest. Port	ACK Set	Action
Spoof-1	In	Internal	Any	Any	Any	Any	Any	Deny
Spoof-2	In	Perim.	Any	Any	Any	Any	Any	Deny
Telnet-1	Out	Internal	Any	TCP	>1023	23	Any	Permit
Telnet-2	In	Any	Internal	TCP	23	>1023	Yes	Permit
FTP-1	Out	Internal	Any	TCP	>1023	21	Any	Permit
FTP-2	In	Any	Internal	TCP	21	>1023	Yes	Permit
FTP-3	Out	Internal	Any	TCP	>1023	>1023	Any	Permit
FTP-4	In	Any	Internal	TCP	>1023	>1023	Yes	Permit
FTP-5	Out	Bastion	Any	TCP	>1023	21	Any	Permit
FTP-6	In	Any	Bastion	TCP	21	>1023	Yes	Permit
FTP-7	In	Any	Bastion	TCP	20	6000-6003	Any	Deny
FTP-8	In	Any	Bastion	TCP	20	>1023	Any	Permit
FTP-9	Out	Bastion	Any	TCP	>1023	20	Yes	Permit
FTP-10	In	Any	Bastion	TCP	>1023	21	Any	Permit
FTP-11	Out	Bastion	Any	TCP	21	>1023	Yes	Permit
FTP-12	Out	Bastion	Any	TCP	20	>1023	Any	Permit
FTP-13	In	Any	Bastion	TCP	>1023	20	Yes	Permit
FTP-14	In	Any	Bastion	TCP	>1023	>1023	Any	Permit
FTP-15	Out	Bastion	Any	TCP	>1023	>1023	Any	Permit
SMTP-1	Out	Bastion	Any	TCP	>1023	25	Any	Permit
SMTP-2	In	Any	Bastion	TCP	25	>1023	Yes	Permit

SMTP-3	In	Any	Bastion	TCP	>1023	25	Any	Permit
SMTP-4	Out	Bastion	Any	TCP	25	>1023	Yes	Permit
NNTP-1	Out	Internal NNTP server	NNTP feed server	TCP	>1023	119	Any	Permit
NNTP-2	In	NNTP feed server	Internal NNTP server	TCP	119	>1023	Yes	Permit
NNTP-3	In	NNTP feed server	Internal NNTP server	TCP	>1023	119	Any	Permit
NNTP-4	Out	Internal NNTP server	NNTP feed server	TCP	119	>1023	Yes	Permit
HTTP-1	Out	Bastion	Any	TCP	>1023	Any	Any	Permit
HTTP-2	In	Any	Bastion	TCP	Any	>1023	Yes	Permit
HTTP-3	In	Any	Bastion	TCP	>1023	80	Any	Permit
HTTP-4	Out	Bastion	Any	TCP	80	>1023	Yes	Permit
DNS-1	Out	Bastion	Any	UDP	53	53	[2]	Permit
DNS-2	In	Any	Bastion	UDP	53	53	[2]	Permit
DNS-3	In	Any	Bastion	UDP	Any	53	[2]	Permit
DNS-4	Out	Bastion	Any	UDP	53	Any	[2]	Permit
DNS-5	Out	Bastion	Any	TCP	>1023	53	Any	Permit
DNS-6	In	Any	Bastion	TCP	53	>1023	Yes	Permit
DNS-7	In	Any	Bastion	TCP	>1023	53	Any	Permit
DNS-8	Out	Bastion	Any	TCP	53	>1023	Yes	Permit
Default-1	Out	Any	Any	Any	Any	Any	Any	Deny
Default-2	In	Any	Any	Any	Any	Any	Any	Deny

[2] UDP packets do not have ACK bits.

Here is some additional information about each set of rules in this table.

Spoof-1 and Spoof-2

Block incoming packets that claim to have internal or perimeter net IP addresses - that is, forged packets presumably sent by an attacker. Rule Spoof-1 is the same as the Spoof rule on the interior router; rule Spoof-2 is unique to the exterior router.

Telnet-1 and Telnet-2

Allow outgoing Telnet connections. These are identical to the corresponding rules on the interior router (as are all rules on the exterior router that involve internal and external hosts, but nothing on the perimeter net).

FTP-1 through FTP-4

Allow outgoing passive-mode FTP connections and are identical to the corresponding rules on the interior router.

FTP-5 and FTP-6

Allow the FTP proxy server on the bastion host to open an FTP command channel to FTP servers on the Internet. Note that, unlike the corresponding rules on the interior router, these rules are *not* redundant if you have rules FTP-1 and FTP-2 in place earlier in the list. Why? Because "Bastion" as a source or destination (covered by rules FTP-5 and FTP-6) is not a subset of "Internal" (covered by rules FTP-1 and FTP-2).

FTP-7 through FTP-9

Allow FTP data connections from external FTP servers to the proxy server on the bastion host. The FTP-7 rule prevents an attacker from attacking X11 servers on the bastion host via the hole created by rules FTP-8 and FTP-9. If you have other servers on the bastion host listening for connections on TCP ports above 1023, you should add similar rules for them. Note that trying to list things that should be denied (as in rule FTP-7) is a losing proposition, because your list will almost always be incomplete, e.g., because you overlooked or didn't know about some service, or because the service was added after the filters were established. However, it's the best you can do in this situation, if you must support normal-mode FTP.

FTP-10 through FTP-15

Allow passive- and normal-mode FTP from external clients to the anonymous FTP server on the bastion host. There are no equivalent rules on the internal router because there are no FTP servers on the internal network that external clients can access.

SMTP-1 and SMTP-2

Allow outgoing mail from the bastion host to the outside world.

SMTP-3 and SMTP-4

Allow incoming mail from the outside world to the bastion host.

NNTP-1 to NNTP-4

Allow Usenet news both ways between your Usenet news server and your service provider's news server. These rules are identical to the corresponding rules on the interior router.

HTTP-1 and HTTP-2

Allow the bastion host HTTP proxy server to connect to the HTTP servers on any machine on the Internet. Actually, these rules allow any TCP client program on the bastion host using a port above 1023 to contact any server program on any host on the Internet using any port. This is done so that the HTTP proxy server can contact HTTP servers on nonstandard port numbers (i.e., other than port 80). As broad as these rules are, it's important that they allow only outgoing connections, by examining the ACK bit.

HTTP-3 and HTTP-4

Allow external clients to contact the bastion host HTTP server. There are no equivalent rules on the internal router because there are no HTTP servers on the internal network that external clients can access.

DNS-1

Allows UDP-based DNS queries and answers from the bastion host DNS server to DNS servers in the outside world.

DNS-2

Allows UDP-based DNS queries and answers from Internet DNS servers to the bastion host DNS server. Note that rule DNS-2 (which allows server-to-server communication) is redundant if rule DNS-3 (which allows client-to-server communication) is present.

DNS-3 and DNS-4

Allow external UDP-based DNS clients to query the DNS server on the bastion host and it to answer them.

DNS-5 and DNS-6

Allow TCP-based DNS queries from the bastion host to DNS servers on the Internet, as well as answers to those queries. Also allow zone transfers in which the bastion host DNS server is the secondary server and an external DNS server is the primary server.

DNS-7 and DNS-8

Allow TCP-based DNS queries from the outside world to the bastion host DNS server, as well as answers to those queries. Also allow zone transfers in which the bastion host DNS server is the primary server and an external DNS server is the secondary server.

Default-1 and Default-2

Block all packets not specifically allowed by one of the preceding rules, just as the corresponding rules do on the interior router.

9.1.3 Other Configuration Work

In addition to setting up the packet filtering rules, we need to do various other kinds of configuration work, as described below.

On all of the internal machines

Configure electronic mail so that it gets sent to the bastion host. We're also going to need to install passive mode FTP clients if they're available.

On the internal mail server

Install the *smap* and *smapd* programs from the TIS FWTK and an up-to-date mailer release so that we have a trusted SMTP server.

On the internal (primary) name server

Put in an MX record for every A record, pointing incoming mail to the bastion host; further MX records may be necessary for the internal mail server to direct the traffic internally. We also need to configure the bastion host as a recognized secondary name server, and remove any TXT or HINFO records we don't want the external world to see (i.e., pretty much any records the external world could possibly make any sense of).

On the bastion host

Do all the standard bastion host configuration (removing unused servers, adding logging, and so on), as discussed in Chapter 5. We need to install TIS FWTK and configure FTP proxying, *smap* and *smapd*, and anonymous FTP service from it. We also need to install the CERN HTTP server and configure it to do proxying, as well as to serve the HTTP pages we want to show the outside world.

9.1.4 Analysis

Just how good a firewall is this one we've configured? Let's consider it in relation to the strategies and principles discussed in Chapter 3, Security Strategies.

9.1.4.1 Least privilege

The principle of least privilege is that an object (a program, a person, a router, or whatever) should

have the minimum privileges necessary to perform its assigned task and no more. A corollary of this principle is that systems should be configured so they require as little privilege as possible. You can see this principle in action in several places in this setup. For example, configuring SMTP so that outgoing mail goes out via the bastion host (rather than directly to remote systems) is an application of least privilege, because it lets you control more tightly how internal systems connect to external systems. (In this case, it makes it unnecessary for internal systems to talk directly to external systems in order to provide this service.)

9.1.4.2 Defense in depth

The principle of defense in depth is something else that you can see in the setup we've described. For example, internal hosts are protected from the outside world by the exterior and interior routers. Similarly, the bastion host is protected against attack both by its own careful configuration and by the exterior router.

Several times, we've explicitly made decisions to increase the depth of defense. For example, that's one of the main purposes of using an internal mail server between the bastion host and the internal clients. The interior and exterior routers often deny the same packets. Defense in depth is almost the only reason for having the interior router deny packets that it supposedly can't receive (because they've already been denied by the exterior router).

9.1.4.3 Choke point

The principle of a choke point is clearly applied in our setup, because everything between internal clients and the Internet comes through the perimeter net. Further, much of it comes through the bastion host, via proxies. Only Telnet and FTP are provided in ways that leave them relatively open. These services could have been better choked by using proxies everywhere.

9.1.4.4 Weakest link

There is no single obvious weak link to attack in this configuration. Probably the weakest link is the bastion host, but even a completely compromised bastion host isn't going to help an attacker when it comes to attacking the internal systems; there just aren't that many connections allowed from the bastion host to internal systems. Some of the weakest links you can see remaining in this setup include proxy FTP and SMTP from the bastion host to the internal mail server.

The proxy FTP setup we've described would allow an attacker who has compromised the bastion host to attack servers on ports above 1023 (if there are any) on internal hosts. How can you address this vulnerability? Obtain and use only passive-mode FTP clients internally, don't run proxy FTP, and remove the rules allowing proxy FTP from the filters.

Similarly, the SMTP setup we've described would allow an attacker who has compromised the bastion host to attack your mail server via SMTP. How can you address this vulnerability? Improve the security of the SMTP server on your mail server, e.g., by using an up-to-date version of Sendmail, and by using the TIS FWTK *smap* package.

You can keep playing this game of thinking "If I were an attacker, what would I do?", and then addressing the problems you discover ad nauseam, or until you run out of time or money. At some point, though, you (or your management) will probably decide you've done enough (based on your own site's definition of "enough").

9.1.4.5 Fail-safe stance

You can see the principle of a fail-safe stance applied through the packet filtering rules. In general, the rules specify what you're going to allow, and deny everything else by default. This is a fail-safe approach, because if something unanticipated comes along (a new service, for example), it won't be allowed through your firewall; unless, of course, it mimics or is tunneled through some other service you do allow. The redundant router rules also provide a fail-safe against failure of one router or the other. If filtering accidentally or temporarily gets turned off on one router (causing it to pass all packets), the other still does most of the same filtering, at least as far as the outside world is concerned.

9.1.4.6 Universal participation

If this is our site's only connection to the Internet, we have involuntary universal participation; everybody has to go through the firewall to get to the Internet. Of course, we'd be much better off with voluntary universal participation, but that may require some user education about the goals of and the need for the security measures we're adopting.

To some extent, we're relying on voluntary universal participation. We've granted free Telnet and FTP access, and in the process we've allowed any outbound connection to ports at or above 1024 (which is plenty of rope for the users to hang us with). FTP is by no means the only service above 1024, and Telnet is a perfectly good client to use to get to many of them.

In particular, we've assumed that this is your sole connection to the Internet and that internal users aren't just going to bypass the firewall entirely by setting up their own Internet connections. All it takes is one joker with a modem, a PPP software package, and an outside phone line, and you too could have an unprotected back door into your network.

9.1.4.7 Diversity of defense

There are opportunities in this configuration to apply the principle of diversity of defense, e.g., using routers from different vendors for the interior and exterior packet filtering systems. Most sites will probably conclude that such an approach is not worth the hassle. However, even if you use similar or identical hardware, you still might get some diversity by having different people do at least the initial configuration of the different filtering systems, and then having them cross-check each other's work.

Using different SMTP servers on the internal mail server and the bastion host would be a fairly major advance in this configuration, because that's one of the main weak points of your setup. Even a less secure SMTP server on the internal mail server is arguably better than one that's going to yield to the exact same attack that just succeeded on the bastion host. The more vulnerable the SMTP server you're using, the more important an issue this is.

9.1.5 Conclusions

There are a lot of advantages offered by a scheme such as the one we've described in the sections above. The main potential disadvantages we can see are cost and complexity; but we don't think that the configuration we've presented is too expensive for most sites, and we think that it presents the minimum necessary level of complexity.

What if you really need to save some money? It would be feasible to construct a screened subnet architecture using a single three-interface router, instead of the pair of two-interface routers we've

described above. The solution would be a little more complex, because you'd have to merge the two separate filtering sets described above, but doing so shouldn't be too difficult.

It would also be relatively easy to construct a more secure configuration with the same basic architecture. A less trusting site would force all Telnet and all FTP through proxies, which would allow much better logging and remove the nagging holes created by allowing every outbound connection to ports above 1024. Once you'd forced Telnet and FTP through proxies, you'd also find that DNS information hiding would be both more practical and more reasonable. However, the price of this increased security would be a more complex and fragile configuration, and one that presents more annoyance to the users.

It would also be possible to increase the services offered without major architecture changes. For example, incoming Telnet and incoming user FTP could be supported relatively easily for a few users on the bastion host or on a dedicated host on the screened network. Serious anonymous FTP service or HTTP service could be provided by configuring extra machines on the screened network. Similarly, you could scale up the firewall to support a second Internet connection or redundant bastion hosts to provide more reliable service or service for a much larger internal network.

Previous: 8.18 Analyzing Other Protocols | Building Internet Firewalls | Next: 9.2 Screened Host Architecture

8.18 Analyzing Other Protocols | Book Index | 9.2 Screened Host Architecture

[Library Home | DNS & BIND | TCP/IP | sendmail | sendmail Reference | Firewalls | Practical Security]

10. Authentication and Inbound Services

Contents:

This book concentrates primarily on how to safely let your users go *out to* the Internet, but there's also another side to Internet security: how do you safely allow users to come *in from* the Internet?

For anonymous services, such as accessing an anonymous FTP server, HTTP server, or Gopher server that you provide, the solution is clear: you protect the servers as best you can to allow outsiders to access the information you want to provide and to prevent them from accessing anything else. In these anonymous services, all the information you release is intended to be readable by anybody. (See the discussion of these services in Chapter 8, Configuring Internet Services.)

For nonanonymous services, however, the situation is much more complex. For nonanonymous services (or "authenticated" services, as they're commonly called), the user who is attempting to access the service first needs to prove his identity to your server so that your server can decide whether the user is authorized to do what he is requesting. Examples of authenticated services you might want to provide include:

- Allowing your users to log in (via Telnet) from the Internet, e.g., while they're away at conferences or visiting other sites.
- Allowing researchers and collaborators from affiliated sites to log in to your systems.
- Allowing selected customers or clients to log in to your systems.

Authentication is basically verified, proven identification. How do users prove to a system that they're really who they say they are? Don't confuse authentication (figuring out who somebody is) and authorization (figuring out what they're allowed to do). Authentication is a prerequisite for authorization (unless everybody is authorized to do something, such as anonymous FTP), but they are separate and distinct concepts.

This chapter focuses on inbound services and how authentication can reduce the risks associated with using these services. It also touches on a few additional encryption and authentication issues that apply to both inbound and outbound services, such as network-level encryption and where to place modem pools.

Cryptography

What are the basic differences between private and public key cryptography?

Private key algorithms include the Data Encryption Standard (DES) (used by Kerberos), IDEA, and the Skipjack algorithm that underlies the Clipper Chip. With private key, a single key is shared by two parties and must be kept secret by both of them (this is the private key). The sender of a communication encrypts the message with this secret key; the recipient must decrypt it with the same key. To communicate with someone securely, you must tell that person the cryptographic key you are using; you must also keep anyone else from discovering or overhearing the key. This process, called key distribution, is difficult and cumbersome to do securely.

Public key algorithms include RSA, and Diffie-Hellman. With public key systems, a mathematical process generates two mathematically related keys for each individual. A message encrypted with one key (the public key) can be decrypted only with the other key (secret or private key). Public keys can be known to anyone, but secret keys must be kept so. To transmit a secret message, the sender encrypts his message with the public key of the intended recipient. The recipient decrypts that message with his own secret key; the only key that will decrypt the message is the secret key associated with the public key used to encrypt it. Public key cryptography also gives you the ability to "sign" messages. If the sender signs a message with his secret key, the recipient can validate the signature by applying the sender's public key to the message; if the public key successfully decrypts the message, it must have been signed with the corresponding secret key.

Public key algorithms are slow, often thousands of times slower than equivalently secure private key algorithms. For this reason, public and private key algorithms are often used in conjunction with each other. For example, the Pretty Good Privacy (PGP) encryption package works this way. To send an encrypted message to a recipient, the sending PGP program generates a random "session key." This session key is used with a private key algorithm to encrypt the message to be sent (this is fast). The session key itself is encrypted with a public key algorithm (this is slow, but the session key is small, especially compared to the whole message), using the recipient's public key, and is sent along with the encrypted message. The recipient first uses the public key algorithm and his secret key to decrypt the session key (this is slow, but the session key is small), and then uses the session key and private key algorithm to decrypt the whole message (this is fast). For a detailed discussion of PGP, see Simson Garfinkel's book, *PGP: Pretty Good Privacy* (referenced in Appendix A, Resources).

> **NOTE:** Although this chapter mentions aspects of cryptography - cryptography is the basis for many of the authentication mechanisms described in this book - we do not attempt to discuss cryptography itself here in any depth. There are many excellent books on this broad and complex topic, and we can't hope to do justice to the topic in a few pages here, where our focus is practical, rather than theoretical. You will be better served by referring to a book such as Bruce Schneier's *Applied Cryptography* for a definitive treatment. (See Appendix A for information.)

10.1 Risks of Using Inbound Services

Inbound services pose a number of security risks. In this section, we focus on Telnet as an example, but the same problems, principles, and solutions apply to other authenticated services (such as nonanonymous FTP) as well.

There are three principal risks associated with allowing inbound services:

Hijacking

Having someone steal a connection after the user has authenticated himself or herself to your system.

Packet sniffing

Having someone read confidential data as it passes across the network, without interfering with the connection itself.

False authentication

Having someone who is not a valid user convince your system he or she *is* a valid user.

10.1.1 Hijacking

Hijacking attacks allow an attacker to take over an open terminal or login session from a user who has been authenticated by the system. Hijacking attacks generally take place on a remote computer, although it is sometimes possible to hijack a connection from a computer on the route between the remote computer and your local computer.

How can you protect yourself from hijacking attacks on the remote computer? The only way is to allow connections only from remote computers whose security you trust; ideally, these computers should be at least as secure as your own. You can apply this kind of restriction by using either packet filters or modified servers. Packet filters are easier to apply to a collection of systems, but modified servers on individual systems allow you more flexibility. For example, a modified FTP server might allow anonymous FTP from any host, but authenticated FTP only from specified hosts. You can't get this kind of control from packet filtering. Connection control at the host level is available from wrappers in the TIS FWTK (the *netacl* program) or Wietse Venema's TCP Wrapper; these may be easier to configure than packet filters, but provide the same level of discrimination - by host only.

Hijacking by intermediate sites can be avoided using end-to-end encryption. (See the discussion of network-level encryption later in this chapter.) If you use end-to-end encryption, intermediate sites won't be able to encrypt the data stream properly (because they don't know the appropriate key), and therefore won't be able to hijack sessions traversing them.

Hijacking is a fairly technical attack. The overall risk to an organization from hijacking attacks is probably pretty small. Most sites choose to accept this small risk and allow some accounts to access systems from anywhere on the Internet. You may decide that hijacking is an acceptable risk for your own organization, particularly if you are able to minimize the number of accounts that have full access and the time they spend logged in remotely. However, you probably do not want to allow hundreds of people to log in from anywhere on the Internet. Similarly, you do not want to allow users to log in consistently from particular remote sites without taking special precautions.

10.1.2 Packet Sniffing

Attackers may not need to hijack a connection in order to get the information you want to keep secret. By simply watching packets pass - anywhere between the remote site and your site - they can see any information that is being transferred. *Packet sniffing* programs automate this watching of packets.

Sniffers may go after passwords or data. There are different risks associated with each type of attack. Protecting your passwords against sniffing is easy: use one of the several mechanisms described later in this chapter to use nonreusable passwords. With nonreusable passwords, it doesn't matter if the password is captured by a sniffer; it's of no use to them, because it cannot be reused.

Protecting your data against sniffers is more difficult. You could encrypt the data at your site if you always knew in advance which data to encrypt, and if you could rely on the remote site to have the appropriate decryption programs. It isn't safe for a user to ask for data to be encrypted while the user is logged in across the network; the sniffer will see the commands issued by the user (perhaps even the key used for encryption), and may be able to use that information to decrypt the data. If the user doesn't provide a key directly, the system has to somehow use a stored key, which might be compromised in other ways (such as a break-in to the system doing the encryption).

Unfortunately, encryption in advance is not practical. It may serve if you need to transfer files occasionally, but it isn't going to provide any kind of meaningful connection. In order to preserve data confidentiality for real interactive access, you'll need end-to-end encryption. Most end-to-end encryption systems require advance coordination between the two ends in order to set the system up. If you have ongoing sensitive interactions with particular sites, however, it may be worth the effort.

As we've described for hijacking, if only a small number of people from a site are doing occasional work from random hosts over the Internet, most organizations are willing to accept the relatively small risk associated with the sniffing of data. However, you need to make sure that nobody at your site purposefully accesses confidential information across the Internet without taking precautions. Moreover, you certainly do not want to set up situations in which confidential information consistently crosses the Internet unencrypted. For example, you would not want a human resources person to work from home on your unencrypted personnel files across the Internet.

10.1.3 False Authentication

The third main risk to inbound services is *false authentication*: the subversion of the authentication that you require of your users, so that an attacker can successfully masquerade as one of your users.

In most cases, if you have a secret you want to pass across the network, you can encrypt the secret and pass it that way. There is one case in which the encryption solution does not work, and that is the case in which information does not have to be understood to be used. For instance, encrypting passwords will not work, because an attacker who is using packet sniffing can simply intercept and resend the encrypted password without having to decrypt it. (This is called a *playback attack*, because the attacker records an interaction and plays it back later.) Therefore, dealing with authentication across the Internet requires something more complex than encrypting passwords. What you need is an authentication method where the data that passes across the network is nonreusable, so an attacker can't capture it and play it back.

The next section describes authentication and how it works. As we explain there, there are many types of authentication methods, some more secure than others.

Part III: Keeping Your Site Secure

Part III describes how to establish a security policy for your site, maintain your firewall, and handle the security problems that may occur with even the most effective firewalls.

Chapter 11, Security Policies, discusses the importance of having a comprehensible and well-understood security policy for your site, and what that policy should and should not contain. It also discusses ways of getting management and users to buy into the policy.

Chapter 12, Maintaining Firewalls, describes how to maintain security at your firewall over time and how to keep yourself aware of new Internet security threats and technologies.

Chapter 13, Responding to Security Incidents, describes what to do when a breakin occurs, or when you suspect that your security is being breached.

11. Security Policies

Contents:

The word "policy" makes many people flinch, because it suggests impenetrable documents put together by unknowledgeable committees, which are then promptly ignored by everyone involved (except when they make a good excuse or weapon). That's not the kind of policy we're discussing in this chapter.

The policy we're talking about here is like a nation's foreign policy. It might be discussed in documents - of varying amounts of legibility - but its primary purpose is to lay out a direction, a theory of what you're trying to achieve. People sometimes confuse the words "policy," "strategy," and "tactics." A *policy* is what determines what wars you're going to fight and why. A *strategy* is the plan for carrying out the war. A *tactic* is a method for carrying out a strategy. Presidents determine policy; generals determine strategies; and anybody down to a foot soldier might determine a tactic.

Most of this book is about tactics. The tactics involved in building a firewall, the nitty-gritty details of what needs to be done here, are complex and intricate. However, no matter how good your tactics are, if your strategy and policy are bad, you can't succeed. In the 1800s, an American named William Walker set out to conquer Nicaragua for the United States. His strategy and tactics were, if not impeccable, certainly successful: he conquered Nicaragua. Unfortunately, there was a literal fatal flaw in his plan. The United States did not at the time want Nicaragua, and when he announced that he had conquered it, the U.S. government was completely uninterested in doing anything about it. Walker ended up ruling Nicaragua very briefly, before he was killed in a popular uprising. This was the result of getting the strategy and the tactics right, but completely botching the policy.

11.1 Your Security Policy

Most technical computer people consider a single, unified, published security policy to be desirable in the abstract, but believe - with a strong basis in personal experience - that attempting to come up with one is going to be extremely painful. For example, walk up to any system administrator and ask about users and passwords, and you are almost guaranteed to be rewarded with a rant. Everybody has a story about the apparent insanity of people faced with passwords, one of the simplest and most comprehensible security issues: the professor who explained that he was too important to need a good

password; the mathematician who was told that he couldn't use a password because it was in an English dictionary (and who replied that he wasn't using the *English* word that was spelled that way, he was using the *Russian* word that was spelled that way, and nobody had told him not to use Russian words). This kind of experience is apt to convince system administrators that their user community is incapable of dealing intelligently with security issues.

There is no doubt that putting together a security policy is going to be a long, involved process, and that it's the exact opposite of the types of tasks most technical people enjoy. If you like to program, you are extremely unlikely to enjoy either the meetings or the bureaucracy involved in policy making. On the other hand, putting together a security policy is a great deal more amusing than dealing with the side effects of not having a policy. In the long run, you'll spend less time in meetings arguing about security if you get it out of the way ahead of time.

Developing a security policy also doesn't need to be as bad as you may be expecting. Many of the problems with security policies are caused by people who are trying to write a security policy that sounds like a security policy, which is to say that it's written in big legal and technical words and says threatening things about how users had better behave themselves. This doesn't work. It's also the most unpleasant way to do things, because it involves hostility and incomprehension all around. It's true that your organization may at some point need a security policy that's written in big legal words (to satisfy some big legal requirements). In that case, the security policy you write shouldn't contradict the legalistic document, but the policy you write doesn't need to be that legalistic one.

Another problem people have in trying to write security policies is that they have a strong feeling about what the policy ought to be, and they're uncomfortable that the actual policy they enforce does not meet that standard. There is a great deal of lip service paid to the notion that security should be absolute: you should have a site that nobody could ever break in to; where every user has exactly one account, and every account has exactly one user; and where all the passwords are excellent, and nobody ever uses anybody else's password for anything.

In the real world, nobody's site is like that, a fact that is well-known and well-accepted. That doesn't keep people from claiming that they want to make their site like that, sometimes in big words on many pieces of paper that they call a security policy. Invariably, every time, without exception, these policies are not followed by anybody.

It's unlikely that your policy is one that emphasizes security at all costs. Such a policy would be irrational. It is reasonable to value other things highly enough to be willing to compromise security.

Most houses would be more secure with bars over all the windows. Few people are willing to put bars over their windows, despite a desire to protect themselves. People have a number of reasons for compromising their security in this way. To start with, bars are expensive and they interfere with using the windows for many of their normal purposes (e.g., seeing out of, climbing out of in an emergency). But people are willing to go to equal expense and inconvenience to apply other security solutions, and they may avoid barring windows even when it's the cheapest and most convenient solution, because it looks bad and makes them feel oppressed.

This is entirely reasonable, and it's entirely reasonable to make the same type of decision about your computer security. You may not want the best security money can buy, or even the best security you can afford.

What do you want? You want the best security that meets your requirements for:

Affordability

How much money does the security cost?

Functionality

Can you still use your computers?

Cultural compatibility

Does it conflict with the way people at your site normally interact with each other and the outside world?

Legality

Does it meet the your site's legal requirements?

Don't pretend that you want to be absolutely secure, if only you could afford it. You don't live your life with the most perfect security money could buy. For the same reasons, it's extremely unlikely that your institution can maintain the characteristics that are important to it if it also installs the most perfect security money could buy. People don't like learning or working in a hostile environment; because they won't do it, you'll either lose the security or lose the organization.

Sometimes a small concession to insecurity can buy a large payoff in morale. For example, rulemakers reel at the idea of guest accounts, but a guest account for a spouse can make a big difference in how people feel about work. And there are sometimes unexpected results. One university computer center was asked why its student employees were allowed to hang around at all hours, even after the labs were closed, doing random activities of dubious value to the computer center; it seemed insecure at best. The answer was that several years before, an operator who was typing his girlfriend's term paper in a lab after hours had discovered and responded to a critical emergency. Because he had saved the facility from what seemed likely to be a million dollars worth of uninsured damage (insurance companies have a nasty tendency to consider floods in windowless third-floor computer rooms to be acts of God, and thus uninsurable), the computer facility management figured that all the computer time the operators wanted had already been paid for.

On the other hand, if you have too little security, you can lose the organization to lawyers or attackers, and what matters there is what you do, not what you write down. Writing down marvelous policies that don't get enforced certainly won't save you from people who are trying to break into your computer, and it generally won't save you from lawsuits, either. The law counts only policies that you make some attempt to enforce. Writing it down and brazenly not doing it proves that you aren't simply too stupid to know what to do: it demonstrates that you actually knew what you had to do, and didn't do it!

11.1.1 What Should a Security Policy Contain?

First and foremost, a security policy is a way of communicating with users and managers. It should tell them what they need to know to make the decisions they need to make about security.

11.1.1.1 Explanations

It's important that the policy be explicit and understandable about why certain decisions have been made. Most people will not follow rules unless they understand why they're important. A policy that specifies what's supposed to be done, but not why, is doomed. As soon as the people who wrote it leave, or forget why they made those decisions, it's going to stop having any effect.

11.1.1.2 Everybody's responsibilities

A policy sets explicit expectations and responsibilities among you, your users, and your management; it lets all of you know what to expect from each other. It's a mistake to distribute a policy that concentrates entirely on what users need to do to make the site secure (it seems hostile and unfair), or entirely on what system administrators need to do (it encourages the users to believe that somebody else will handle it, and they don't have to worry about it).

11.1.1.3 Regular language

Most people are not lawyers, and they're not security experts. They're comfortable with casual descriptions. You may be afraid to write a policy that way because it may seem uncomfortably casual and too personal. But it's more important to make your security policy friendly and understandable than to make it precise and official-looking. Write it as if you were explaining it to a reasonably bright but nontechnical friend. Keep it a communication between peers, not a memo from Mount Olympus. If that's not acceptable in your corporate culture, write two separate policy descriptions.

You will not get people to comply unless they understand the document and want to comply with it, and that means they have to at least be willing to read it. If they shut their brains off in paragraph two because the document sounds legal and threatening, you lose. You also lose if they decide that you think they're stupid, or if they decide that you don't care. Don't get so informal that you seem condescending or sloppy. If necessary, get a technical writer to clean up the punctuation and spelling.

11.1.1.4 Enforcement authority

Writing down the policy is not the point; living by it is. That means that when the policy isn't followed, something should happen to fix it. Somebody needs to be responsible for making those corrections happen, and the policy needs to specify who that's going to be and the general range of corrections. Here are some examples of what a security policy might specify:

- Managers of certain services have the authority to revoke access.
- Managers will be asked to take care of some kinds of transgressions.
- Facilities that don't meet certain standards may be cut off from the corporate network and external access by the people who run the corporate network.

The policy should specify who is going to decide and give some indication of what kinds of penalties are available to them. It should not specify exactly what will happen when; it's a policy, not a mandatory sentencing law.

11.1.1.5 Provision for reviews

You can't expect to set a policy up once and forget it. The needs of your site will change over time, and policies that were perfectly sensible may become either too restrictive or too lax. Sometimes change is obvious: if you work for a startup company that goes from six people to 6,000 people, it will probably occur to you that things are different in important ways (but you still may not get around to redoing the security policy if you didn't set up a mechanism for that in advance). If you work for a 200-year old university, however, you may not expect much change. However, even if the organization appears to be doing its best to fossilize, the computers change, the external networks change, and new people come in to replace ones who leave. You still need to review and change your policies on a regular basis.

11.1.1.6 Discussion of specific security issues

Because of the differences between organizations, it's hard to be specific about issues without writing an entire book just about security policies. However, here are some common issues to consider when you are writing a policy:

- Who is allowed to have an account at your site? Do you have guest accounts? What do you do about contractors, vendors, and clients?
- Can accounts be shared between multiple people? What about a secretary who uses an executive's account to process that person's electronic mail? What about joint projects? What about family members? Is it sharing an account if you let somebody else borrow a window on your machine really quickly?
- When do people lose the privilege of having an account, and what do you do about it? What happens if people leave or are denied access?
- Who can set up dial-in modems? Is it OK for other people to set up dial-out modems? Is there anything special about PPP, SLIP, or ISDN lines?
- What do people need to do before they connect a computer to the main network?
- How secure do computers need to be before they get services from centrally maintained machines?
- How secure do computers need to be in order to connect to a network with unprotected access to the Internet?
- How is financial data going to be protected?
- How is confidential information about people going to be protected?
- What do individual users need to do to protect themselves and the site? What kinds of passwords should they have, and when should they change them?
- What can people do on the Internet? Should they be transferring random executables in and running them?
- What precautions do you need to take against viruses on personal computers?
- Who can connect your site to external networks, and what's an external network? Is it OK for a project manager to connect your site to another specific site? How about putting in a second Internet connection?
- How are home computers going to be secured? How are they going to get secure access to your network?
- How are people who are traveling going to get access to the network?
- What information is considered company confidential? How is it going to be protected? Can it be sent outside the site via electronic mail?
- If you have remote sites, how are they going to get secure access to your main network?

11.1.2 What Should a Security Policy Not Contain?

Some pieces of information don't belong in your site's security policy, as we discuss in this section.

11.1.2.1 Technical details

The security policy needs to describe what you're trying to protect and why; it doesn't necessarily need to describe the details of how. It's much more useful to have a one-page document that describes "what" and "why" in terms that everyone in your organization can understand, than a 100-page document that describes "how," but that nobody except your most senior technical staff can understand.

For example, consider a policy that includes a requirement that says:

> Nonreusable passwords shall be used to authenticate all incoming connections from the outside world, in order to prevent potential attackers from being able to capture reusable passwords by monitoring such connections.

This requirement is much more useful than a policy that says:

> S/Key will be used for all incoming connections.

Why? Because the first policy describes *what* is to be protected and *why*, and it leaves *how* open so the technical staff can select the best implementation.

A policy that says the following is better yet:

> Regular passwords are often stolen and reused when they pass across networks. We won't use passwords that can be reused across networks our company doesn't control.

This policy communicates the same information without the legal-style language. It also clarifies some other points. For example, in the original language does the "outside world" include companies that have special relationships with yours? It may seem obvious to you that it does, but it probably doesn't seem obvious to the managers who are arranging to work with those companies. The reworded language makes it clear what the criterion is (although you may still end up arguing about what networks meet it).

Policy can guide you in selecting and implementing technology, but it shouldn't be used to specify it. It's often much easier to get management to buy into, and sign off on, an overall policy than on a specific technology.

11.1.2.2 Somebody else's problems

Every site's security policy is different. Different sites have different concerns, different constraints, different users, and different capabilities; all of these lead to different policies. Further, a site's policy may change over time, as the site grows and changes. Don't assume that you need to do things the way they've always been done, or that you can borrow somebody else's policy and simply change the names in it.

---

12. Maintaining Firewalls

Contents:

If you've done a good job of designing a firewall that fits the needs of your organization, maintaining that firewall should be fairly straightforward. What does it mean to maintain a firewall? Maintenance tasks fall into three major categories:

- Housekeeping
- Monitoring your system
- Keeping up to date

Once you've designed and built your firewall, it really shouldn't take a great deal of effort to keep it going, especially because much of the maintenance work can be automated.

12.1 Housekeeping

Housekeeping is the eternal round of small tasks that need to be done to keep your firewall clean and safe. There are three main tasks you'll need to deal with again and again:

- Backing up your firewall
- Managing your accounts
- Managing your disk space

12.1.1 Backing Up Your Firewall

Make sure to back up all parts of your firewall. That means not only the general-purpose computers you may be using as bastion hosts or internal servers, but also the routers or other special-purpose devices. Rebuilding router configurations usually isn't easy, and your security depends on having your routers configured correctly.

Put your general-purpose machines on a regular, automated backup system. Preferably, that system should produce confirmation mail when it is running normally and distinctly different messages when it sees errors.

Why not produce mail only when errors occur? If the system produces mail only on errors, you won't notice the system if it fails to run at all. (Silence is not necessarily golden, as any parent of small children knows. If they aren't making noise, they're probably making mischief.)

Why distinctly different messages? If the system produces even vaguely similar messages when it is running normally and when it fails, people who are accustomed to ignoring the success messages will also ignore the failure messages. Ideally, a separate program should check to make sure that the backups have run, and to produce messages when they haven't.

Special-purpose machines like routers change much less often, and probably don't need an automated backup system. (This is fortunate, because they rarely support them.) When you do make changes, take advantage of any means available to record the configuration. Most systems write their configuration to a local floppy disk; some of them also support FTP. Write two floppy disks, clearly label and date them, and store one of them separate from the machine. Make these backups even if you have downloaded the configuration with FTP; you don't want the router to be completely dependent on another machine. If you have written the configuration on the router, rather than FTP'ing it, make an FTP copy as well as the floppy disks. Why? Sometimes it's easier to find files than to find small physical objects like floppy disks, and sometimes the floppy drive dies when the rest of the router still works.

> **NOTE:** The design of backup systems is outside the scope of this book. This description, along with the section in Chapter 13, Responding to Security Incidents called "Backing Up Your Filesystems" provide only a summary. If you're uncertain about your backup system, you'll want to look at a general system administration reference. See Appendix A, Resources for complete information on additional resources.

12.1.2 Managing Your Accounts

Account management - adding new accounts, removing old ones, aging passwords, etc. - is one of the most often neglected housekeeping tasks. On firewall systems, it's absolutely crucial that new accounts be added correctly, old accounts removed promptly, and passwords changed appropriately. (See your own system's documentation for how to do all this.)

Establish a procedure for adding accounts; wherever you can, use a program to add them. Even though there shouldn't be many users on your firewall systems, every one of them is a possible danger, and it's worth the effort to ensure they're set up correctly every time. People have an unfortunate tendency to leave out steps, or to pause for a few days in the middle of a process. If that gap leaves an account that has no password, you're creating open invitations to intruders.

Make sure your account creation procedure includes dating the account, and that accounts are automatically reviewed every few months. You don't need to automatically turn them off, but you do need to automatically inform somebody that they've timed out. This is relatively easy to do on a UNIX system; it may be harder on other systems, particularly dedicated systems like routers. If possible, set things up so that the accounts can be watched by an automated system. This can be done by generating account files on the UNIX side and then transferring them to the other machine, or by generating the accounts on the machine itself, but automatically copying the account files to the UNIX side and examining them.

You should also arrange to get termination notices from the appropriate authorities whenever someone leaves your organization. Most companies are able to send around notices for full-time employees, and most universities can provide graduation notification for students. It may be much harder to keep track of contractors and students who drop out, so you shouldn't rely on official notifications to tell you about everybody who has left. You may also need to confirm notifications: for example, you may get termination notices for people who are actually converting to contract status, or graduation notices for people who are continuing as graduate students or junior faculty. These people are going to be annoyed if you get rid of all their files (although it's probably acceptable to temporarily disable their accounts if their status is in doubt).

If your operating system supports password aging, you may want to turn it on. Use a relatively long time period - perhaps three to six months. If you time out passwords more frequently (e.g., every month), users will be willing to go to great lengths to circumvent the timeout, and you probably won't see any real gain in security. Similarly, if your password aging doesn't guarantee that the user will see a notification before the account becomes unusable, don't turn it on. Otherwise, you will annoy your users, and you will run the risk of accidentally locking out administrators who have a critical need to use the machine.

If password aging on your system is going to require a user to change his password as he is logging in, you need a password program that strictly enforces good passwords. If you don't do this, people will choose simple passwords in the heat of the moment, honestly intending to change them to better ones later. All in all, you may find it more effective to simply send people notices on a regular basis, even though you'll get less compliance that way.

12.1.3 Managing Your Disk Space

Data always expands to fill all available space, even on machines that have almost no users. People dump things in odd corners of the filesystem, "just for now," and they build up there. This causes more problems than you may realize. Aside from the fact that you may want that disk space, this random junk complicates incident response. You'll end up asking yourself:

> Is that a program that you left lying around last time you needed to install a new version, or did an intruder put it in?
>
> Is that truly a random data file, or does it have some deep meaning to an intruder?

Unfortunately, there is no automatic way to find junk; human beings, particularly system administrators who can write anywhere on the disk, are too unpredictable. Another person needs to look around on a regular basis. It's particularly effective to send every new system administrator on a tour of the disks; they'll notice things the old hands have become accustomed to.

Auditing programs like Tripwire, discussed in Chapter 5, Bastion Hosts, will tell you about new files that appear in supposedly static areas, and this information will help you keep things straight. You will still need to check all the areas where you intentionally allow changes, and you should periodically go back and re-check the static areas. You will probably get the alert while you still know why you put that file in that place, and that knowledge may wear off over time.

On most firewall machines, your main disk space problem will be logs. These can and should be rotated automatically, and you may want to compress them as well; a program like *trimlog* (see Appendix B, Tools) helps automate the process. It's important to stop programs or cause them to suspend logging while you are trying to truncate or move logs. If a program is trying to write to a log

file while you're trying to move or truncate it, you're obviously going to have problems. In fact, though, you may run into difficulties even if a program is simply holding the file open in preparation for writing to it later, e.g., you may discover the program is still logging to the file you renamed.

---

13. Responding to Security Incidents

Contents:

The CERT Coordination Center (CERT-CC) reports that, despite increased awareness, the first time many organizations start thinking about how to handle a computer security incident is *after* an intrusion has occurred. Obviously, this isn't a great approach. You need a plan for how you're going to respond to a computer security incident at your site, and you need to develop that plan well before an incident occurs.

There isn't room here to detail everything you need to know to deal with a security incident: attacks are many and varied; they change constantly; and responding to them can involve a Byzantine assortment of legal and technical issues. This chapter is intended to give you an outline of the issues involved, and the practical steps you can take ahead of time to smooth the process. Appendix A, Resources provides a list of resources that may provide additional help.

13.1 Responding to an Incident

This section discusses a number of steps you'll need to take when you respond to a security incident. You won't necessarily need to follow these steps in the order they're given, and not all of these steps are appropriate for all incidents. But, we recommend that you at least contemplate each of them when you find yourself dealing with an incident.

In the section called "Planning Your Response," later in this chapter, we'll look again at each of these rules and help you figure out how to work them into the overall response plan that you should develop before an incident actually occurs.

Rules for Incident Response

In their book *Practical UNIX Security*, Simson Garfinkel and Gene Spafford provide two excellent, overriding rules for incident response. Keep these rules in mind as you read this chapter and during any real-life incident response.

- Rule 1: Don't Panic!
- Rule 2: Document!

13.1.1 Evaluate the Situation

The first step in responding to a security incident is to decide what response, if any, needs to be made immediately. Ask these questions:

- Has an attacker succeeded in getting into your systems? If so, you have a genuine emergency on your hands, whether or not the attacker is currently active.
- Is the attack currently in progress? If so, you need to decide how you're going to react right now. If the attack isn't currently in progress, you may not be in such a hurry.

If the incident looks like an aggressive attack on your system, you probably want to take strong steps quickly. These steps might include shutting down the system or your Internet connection until you figure out how to deal with the situation.

On the other hand, if the incident is a less aggressive one - perhaps someone has just opened a Telnet connection to your machine and is trying various login/password pairs - then you may want to move more slowly. If you're reasonably confident that the attack won't succeed (e.g., you can see that the attacker is trying passwords that consist of all lower-case letters, and you know for certain that no account on the system has such a password), you might want to leave things alone and just watch for a while to see what the attacker does. This may give you an opportunity to trace the attack. (However, see the section below called "Pursuing and Capturing the Intruder" for a discussion of the issues involved in tracing an attack.)

Whatever you do, remember Rule 1: Don't Panic!

13.1.2 Disconnect or Shut Down, as Appropriate

Once you've evaluated the situation, your next priority is to give yourself the time to respond without risking your systems further. The least disruptive alternative is usually to disconnect the affected machine from all networks; this will shut down any active connections. Shutting down active connections may make it harder to trace the intruder, but it will allow the rest of the people at your site to continue to do their work, and it will leave the intruder's programs running. This may help you to identify who the intruder might be.

If you're afraid that other machines have been compromised or are vulnerable to the same attack, you'll probably want to disconnect as many machines as you can as a unit. This may mean taking down your connection to the Internet, if possible. If your Internet connection is managed elsewhere in your organization, you may need to detach just your portion of the network, but you'll also need to talk to other parts of your organization as soon as possible to let them know what's happening.

In some situations, you may want to shut down the compromised system. However, this should be a last resort for a number of reasons:

- It destroys information you may need.
- You won't be able to analyze or fix the machine while it's down; you'll have to disconnect it from the network eventually anyway to bring it back up again.

- It's even more disruptive to legitimate users than removing the network connection.
- It protects only one machine at a time. (It's much easier to cleanly disconnect a set of systems than to cleanly shut them down.)

Even if you're responding to an incident that has already ended, you still might want to disconnect or shut down the system, or at least close it to users, while you analyze what happened and make any changes necessary to keep it from happening again. This will keep you from being confused by things users are doing, and it will prevent the intruder from returning before you're done.

13.1.3 Analyze and Respond

Your next priority is to start to fix what's gone wrong. The first step in actually correcting the problem is to relax, think for a while, and make sure you really understand what's happening and what you're dealing with. The last thing you want to do is make the situation worse by doing something rash and ill-considered. Whatever corrective actions you're contemplating, think them through carefully. Will they really solve the problem? Will they, in turn, cause other problems?

When you're working in an unusual, high-stress situation like this, the chances of making a major error go way up. Because you're probably going to be working with system privileges (for example, working as root on a UNIX system), the consequences of an error could be serious.

There are several ways you can reduce the chances of making an error. One good way is to work with a partner; each of you can check the other's commands after they're typed but before they're executed. Even if you're working alone, many people find that reading commands aloud and checking the arguments in reverse order before executing them helps avoid mistakes. Resist the temptation to try to work fast. You will go home sooner if you work slowly and carefully.

Try not to let your users get in the way of your response. You may want to give someone the specific job of dealing with user inquiries so the rest of your response team can concentrate on responding to the incident.

Also, try to keep your responders from tripping over each other. Make it clear which system managers and investigators are working on which task, so they won't step on each other's toes (or wind up unintentionally chasing each other as part of the investigation!).

13.1.4 Make `Incident in Progress' Notifications

You're not the only person who needs to know what's going on. A number of other people - in a number of different places - have to be kept informed.

13.1.4.1 Your own organization

Within your own organization, there are people who need to know that something is happening: management, users, and staff. At the very least, let them know that you are busy responding to an incident, and that you may not be available to them for other matters. Usually, they need to know why they're being inconvenienced and what they should do to speed recovery (even if the only thing they can do is to go away and leave you alone).

Depending on the nature of your site and the incident in question, you may also need to inform your legal, audit, public relations, and security departments.

If there are multiple computer facilities at your site, you'll need to inform the other facilities as soon as possible; they are likely sources and future targets for similar attacks.

13.1.4.2 CERT-CC or other incident response teams

If your organization is served by an incident response team such as CERT-CC, or has its own such team, let them know what's going on and try to enlist their aid. (For instructions on how to contact CERT-CC or another response team, see Appendix A.) What steps response teams can take to help you will depend on the charter and resources of the response team. Even if they can't help you directly, they can tell you whether the attack on your site looks as if it is part of a larger pattern of incidents. In that case, they may be able to coordinate your response with the responses of other sites.

13.1.4.3 Vendors and service providers

You might want to get in touch with your vendor support contacts or your Internet service provider(s), if you think they might be able to help or should be aware of the situation. For example, if the attackers appear to be exploiting an operating system bug, you should probably contact the vendor to see if they know about it and have a fix for it. At the very least, they'll be able to warn other sites about the bug. Similarly, your Internet provider is unlikely to be able to do much about your immediate problem, but they may be able to warn other customers. There is also a possibility that your Internet provider has itself been compromised, in which case they need to know immediately. Your vendors and service provider may have special contacts or procedures for security incidents that will yield much faster results than going through normal support channels.

13.1.4.4 Other sites

Finally, if the incident appears to involve other sites - that is, if the attack appears to be coming from a particular site, or if it looks as if the attackers have gone after that site after breaking into yours - you should inform those other sites. These sites are usually easy to identify as the sources or destinations of connections. It's often much harder to figure out how to find an actual human being with some responsibility for the computer in question, who is awake and reachable, and has a common language with you.

If you don't know who to inform, talk to your response team (or CERT-CC). They will probably either know or know how to find out, and they have experience in calling strangers to tell them they have security problems.

13.1.5 Snapshot the System

Another early step to take is to make a "snapshot" of each compromised system. You might do this by doing a full backup to tape, or by copying the whole system to another disk. In the latter case, if your site maintains its own spare parts inventory, you might consider using one of the spares for this purpose, instead of a disk that is already in use and might itself turn out to have been compromised.

The snapshot is important for several reasons:

- If you misdiagnose the problem or blow the recovery, you can always get back to the time of the snapshot.
- The snapshot may be vital for investigative and legal proceedings. It lets you get on with the work of recovering the system without fear of destroying evidence.

- You can examine the snapshot later, after you're back in operation, to determine what happened and why.

Because the snapshot may become important for legal proceedings, you need to secure the evidence trail. Here are some guidelines:[1]

- Uniquely identify (label) the snapshot media, and put the date, time, your name, and your signature on it.
- Write-protect the media - permanently, if possible.
- Safeguard the media against tampering (for example, put it in a locked container) so that if and when you hand it over to law-enforcement or other authorities, you can tell them whose custody the media has been in, and why you're certain it hasn't been tampered with since it was first created.

  [1] See *Computer Crime: A Crimefighter's Handbook*, by David Icove, Karl Seger, and William VonStorch (O'Reilly & Associates, 1995), for a detailed discussion of labeling and protecting evidence.

It's a good idea to set aside an adequate supply of fresh media just for snapshots, because you never know when you're going to need to produce one. It's very frustrating to respond to an incident, and be ready to do the snapshot, only to discover that the last blank tape got used for backups the day before, and the new order hasn't come in yet.

13.1.6 Restore and Recover

Finally, you're at the point of actually dealing with the incident. What do you do? It depends on the circumstances. Here are some possibilities:

- If the attacker didn't succeed in compromising your system, you may not need to do much. You may decide not to bother reacting to casual attempts. You may also find that your incident was actually something perfectly innocent, and you don't need to do anything at all.
- If the attack was a particularly determined one, you may want to increase your monitoring (at least temporarily), and you'll probably want to inform other people to watch out for future attempts.
- If the attacker became an intruder, that is, he actually managed to get into your computers, you're going to need to at least plug the hole the intruder used, and check to make certain he hasn't damaged anything, or left anything behind.

At worst, you may need to rebuild your system from scratch. Sometimes you end up doing this because the intruder damaged things, purposefully or accidentally. More often, you'll rebuild things because it's the only way to ensure you have a clean system that hasn't been booby-trapped. Most intruders start by making sure they'll be able to get back into your system, even if you close their initial entry point. As a result, your systems may be compromised even if the intruder was present only for a short time.

> **NOTE:** Always assume that an intruder has created back doors into your system so that he can get back in again easily. This is one of the first things many intruders do when they break in to a system.

If you need to rebuild your system, first ensure that your hardware is working properly. You want to make sure it passes all relevant self-tests and diagnostics; you don't want to restore onto a flaky system. A reinstall may reveal previously unnoticed hardware problems. For instance, a disk may have bad spots that are in unused files. When you reinstall the operating system, you will attempt to write over the bad parts and the problem will suddenly become apparent.

Next, make sure you are using trusted media and programs, not necessarily your last backup, to restore the system. Unless you are absolutely sure that you can accurately date the first time the intruder accessed your system, you don't know whether or not programs had already been modified at the time the backups happened. It's often best to rebuild your system from vendor distribution media (that is, the tapes or CD-ROM your operating system release came on), and then reload only user data (not programs that multiple users share) from your backup tapes.

If you need programs you didn't get from your vendor (for instance, packages from the Internet), then do one of the following:

- Rebuild and reinstall these programs from a trusted backup (one you're absolutely positive contains a clean copy).
- Obtain and install fresh copies from the site you got the packages from in the first place.

Do not recompile software until you've reinstalled the operating system, including the compiler; you don't know whether the compiler itself, and the libraries it depends on, have been compromised.

This implies that if you're heavily customizing your system or installing a lot of extra software beyond what your vendor gives you, you need to work out a way of archiving those customizations and packages that you're sure can't be tampered with by an attacker. This way, you can easily restore those customizations and packages if you need to. One good way is to make a special backup tape of new software immediately after it's installed and configured, before an attacker has a chance to modify it.

You may have programs that were locally written, and in these cases you may not be able to find even source code that's guaranteed to be uncontaminated. In this situation, someone - preferably the original author - will need to look through the source code. People rarely bother to modify source code, and when they do they aren't particularly subtle most of the time. That's because they don't need to be; almost nobody actually bothers to look at the source before recompiling it.

In one case, a programmer installed a back door into code he expected would run on only one machine, as a personal convenience. The program turned out to be fairly popular and was adopted in a number of different sites within his university. Years after he wrote it, and long after the original machine was running a version without the back door, he discovered that the back door was still present on all the other sites, despite the fact that it was clearly marked and commented and within the first page of code. You can't make a comprehensive search of a large program, but you can at least avoid humiliation by looking for obvious changes.

13.1.7 Document the Incident

Life gets very confusing when you're discovering, investigating, and recovering from a security incident. A good chain of communication is important in keeping people informed and preventing them from tripping over each other. Keeping a written (either hardcopy or electronic) record of your activities during the incident is also important. Such a record serves several purposes:

- It can help keep people informed (and thereby help them to resolve the incident more quickly).

- It tells you what you did, and when, in responding, so that you can analyze your response later on (and maybe do better next time).
- It will be vital if you intend to pursue any legal action.

From a legal standpoint, the best records are hardcopy records generated and identified at the time of occurrence. Just about anything else (particularly anything kept online) could be tampered with or falsified fairly easily - or at least a judge and jury could be convinced of that. You need to produce records on pieces of paper, label and date them, and sign them. Furthermore, unless the pages are actually bound together, so that pages can't be inserted or removed without indication, you'll need to date and sign every page. (And you thought continuous tractor-feed paper was useless these days!)

You need to have legal documentation even if you aren't completely certain you're going to need it. An incident that looks fairly simple to start with may turn out to be serious. Don't assume it isn't going to be worth bringing in the police.

Here are several useful documentation methods you might want to consider:

- Notebooks - Carbon copy lab notebooks are especially useful, because you can write a note, tear it out and give it to someone, and still have a copy of the note. Another benefit is that the pages are usually numbered, so you can determine later on whether any pages have been removed or added.
- Terminals running with attached printers or old-fashioned printing terminals.
- A shell running under the UNIX *script* command, with the resulting typescript immediately printed and identified.
- A personal-computer terminal program running in "capture" mode, with the resulting typescript immediately printed and identified.
- A microcassette recorder for verbal notes.

You will probably want to use multiple methods, one to record what's happening online and one to record what's happening outside of the computer. For example, you might have a typescript of the commands you were typing, but a handwritten log for phone calls.

It's easy to decide what to record online; you simply record everything you do. Remember to use the terminal or session that's being recorded. (With some methods, like *script*, you can record every session you've got going; just make sure you record each session in a separate file.) It's harder to decide what to record of the events that don't just get automatically captured. You certainly want to record at least this much:

- Who you called, when, and why.
- A summary of what you told them.
- A summary of what they told you. (That summary may end up being "see above" some of the time, but you still want to be able to figure out who you were talking to, and when, and why.)
- Meetings and important decisions and actions that aren't captured online (e.g., the time at which you disconnected the network).
