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…

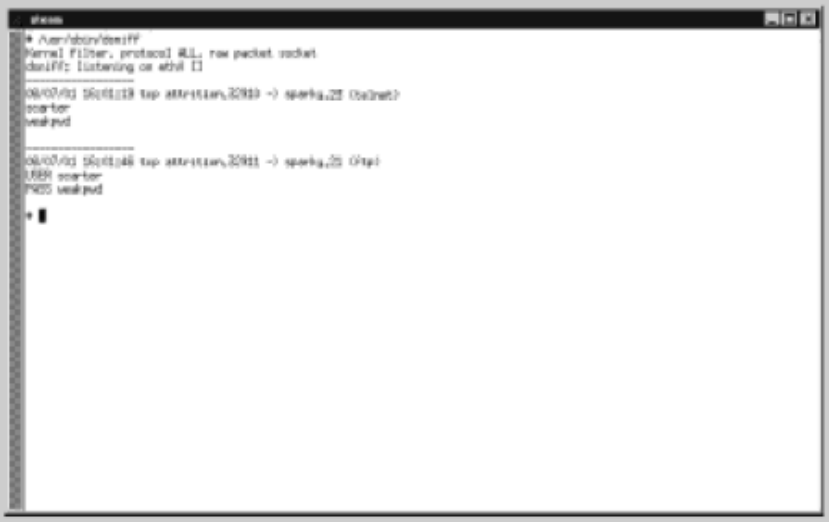
Using dsniff to Capture Passwords

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

Continued

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



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.

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.

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.