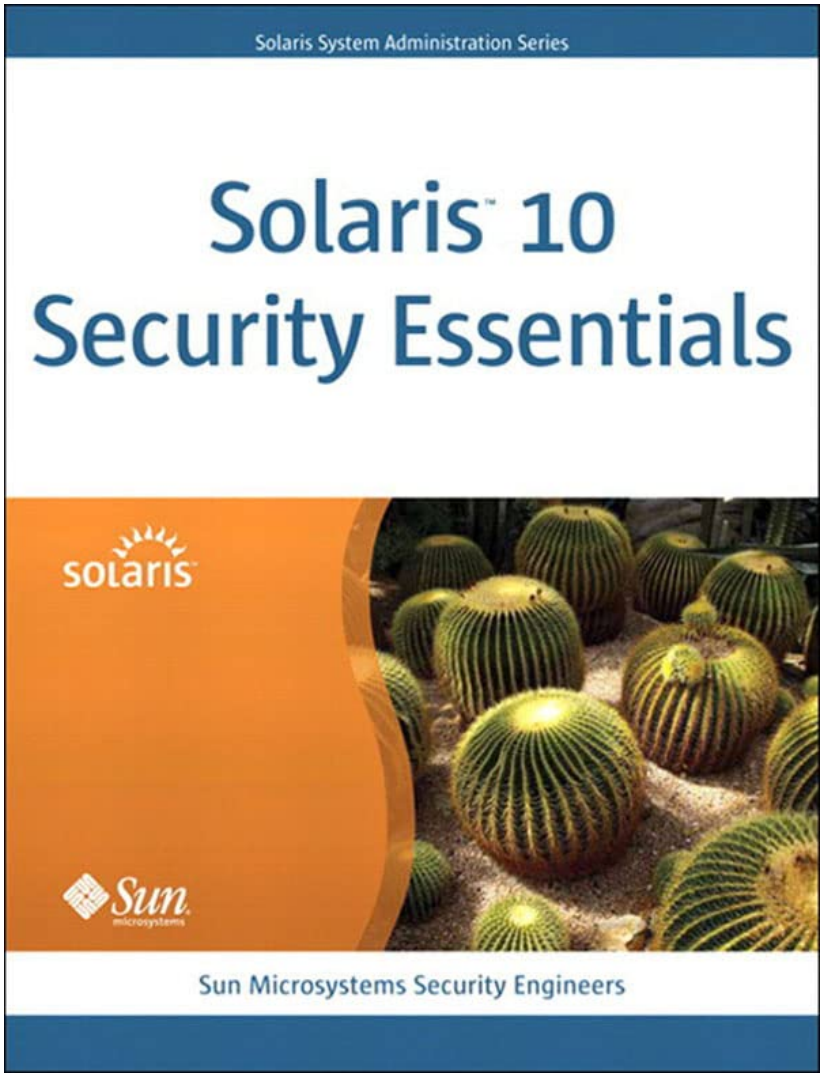
Solaris 10 Security Essentials (Oracle Solaris System Administration Series)

by

Margaret Miller



Synopsis

Solaris™ 10 Security Essentials describes the various security technologies contained in the Solaris operating system. The book describes how to make installations secure and how to configure the OS to the particular needs of your environment, whether your systems are on the edge of the Internet or running a data center. The authors present the material in a straightforward way that makes a seemingly arcane subject accessible to system administrators at all levels. The strengths of the Solaris operating system's security model are its scalability and its adaptability. It can protect a single user with login authentication or multiple users with Internet and intranet configurations requiring user-rights management, authentication, encryption, IP security, key management, and more. This book is written for users who need to secure their laptops, network administrators who must secure an entire company, and everyone in between. The book's topics includeZones virtualization securitySystem hardeningTrusted Extensions (Multi-layered Security)Privileges and role-based access control (RBAC)Cryptographic services and key managementAuditingNetwork securityPluggable Authentication Modules (PAM)Solaris™ 10 Security Essentials is the first in a new series on Solaris system administration. It is a superb guide to deploying and managing secure computer environments.

Sort review

From the Inside FlapWork More Quickly and Easily—and Learn Some New TricksThe Mac command line offers a faster, easier way to accomplish many tasks. It's also the medium for many commands that aren't accessible using the GUI. The Mac OS X Command Line is a clear, concise, tutorial-style introduction to all the major functionality provided by the command line. It's also packed with information the experienced users need, including little-known shortcuts and several chapters devoted to advanced topics. This is a book to get you started, but also a book you won't soon outgrow.Coverage includes:Using Terminal, the gateway to the command lineMaking Terminal easier to use with shortcutsConfiguring your shell: bash and tcshNavigating your file system in TerminalMoving and copying files from the command lineAccessing hidden files with TerminalEditing configuration files with command-line text editorFinding any kind of file or content using simple commandsPrinting from the command lineArchiving, compressing, and decompressing filesManaging users, groups, and permissionsAccessing network volumes and the Internet using simple commandsManaging programs and processesUsing system maintenance commandsFrom the Back CoverWork More Quickly and Easily—and Learn Some New TricksThe Mac command line offers a faster, easier way to accomplish many tasks. It's also the medium for many commands that aren't accessible using the GUI. The Mac OS X Command Line is a clear, concise, tutorial-style introduction to all the major functionality provided by the command line. It's also packed with information the experienced users need, including little-

known shortcuts and several chapters devoted to advanced topics. This is a book to get you started, but also a book you won't soon outgrow.Coverage includes:Using Terminal, the gateway to the command lineMaking Terminal easier to use with shortcutsConfiguring your shell: bash and tcshNavigating your file system in TerminalMoving and copying files from the command lineAccessing hidden files with TerminalEditing configuration files with command-line text editorFinding any kind of file or content using simple commandsPrinting from the command lineArchiving, compressing, and decompressing filesManaging users, groups, and permissionsAccessing network volumes and the Internet using simple commandsManaging programs and processesUsing system maintenance commandsAbout the AuthorKirk McElhearn, co-author of the best-selling Mastering Mac OS X Third Edition, is a professional author, journalist and translator. He has written user manuals for many popular Macintosh programs, and articles for publications including MacWorld magazine and TidBITS.Read more

Look inside the book

Solaris™ 10 Security EssentialsSun Microsystems Security EngineersSun Microsystems PressUpper Saddle River, NJ • Boston • Indianapolis • San FranciscoNew York • Toronto • Montreal • London • Munich • Paris • MadridCapetown • Sydney • Tokyo • Singapore • Mexico CityMany of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.Sun Microsystems, Inc., has intellectual property rights relating to implementations of the technology described in this publication. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents, foreign patents, or pending applications. Sun, Sun Microsystems, the Sun logo, J2ME, J2EE, Solaris, Java, Javadoc, Java Card, NetBeans, and all Sun and Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.THIS PUBLICATION IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC., MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:U.S. Corporate and Government Sales(800) 382-3419corpsales@pearsontechgroup.comFor sales outside the United States please contact:International Salesinternational@pearsoned.comVisit us on the Web:Library of Congress Cataloging-in-Publication DataCopyright © 2010 Sun Microsystems, Inc.4150 Network Circle, Santa Clara, California 95054 U.S.A.All rights reserved.Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:Pearson Education, Inc.Rights and

Contracts Department501 Boylston Street, Suite 900Boston, MA 02116Fax: (617) 671-3447ContentsPrefaceAbout the AuthorsChapter 1 Solaris Security Services1.1 A Solaris Security Story1.2 Security Services in the Solaris OS1.3 Configurable Security Services in the Solaris OSChapter 2 Hardening Solaris Systems2.1 Securing Network Services2.1.1 Minimizing Installed Software2.1.2 Using SMF to Control Network Services2.1.3 Selecting Profiles at Install Time and Afterwards2.1.4 Results of Applying Network Profiles2.2 Configuration Hardening2.2.1 Protect the OpenBoot PROM2.2.2 Enable Solaris Non-Executable Stacks2.2.3 Log Core Dumps2.2.4 Lock Out Accounts2.2.5 Other Security Configuration Ideas2.3 Basic Audit and Reporting Tool2.3.1 Collecting Filesystem Object Attributes (Create Mode)2.3.2 Comparing Filesystem Object Attributes (Compare Mode)2.3.3 Determining Whether a Filesystem Object Is Genuine2.4 Signed ELF Filesystem Objects2.5 Solaris Fingerprint Database (sfpDB)2.5.1 How the sfpDB Tool Works2.5.2 Creating an MD5 File Fingerprint2.5.3 Testing an MD5 File FingerprintChapter 3 System Protection with SMF3.1 Service Management Facility (SMF)3.2 How SMF Configuration Works3.3 Modifying Solaris Services Defaults3.3.1 Configuring the NFS Service3.3.2 Configuring the IP Filter Service3.3.3 Configuring the ftp Service3.3.4 Configuring the Apache2 Web ServiceChapter 4 File System Security4.1 Traditional UNIX File System Security4.1.1 setuid, setgid, and Sticky Bit4.1.2 chown/chgrp4.1.3 Default umask4.1.4 setfacl/getfacl4.1.5 Extended Attributes4.1.6 Promiscuous Execution4.2 ZFS/NFSv4 ACLs4.2.1 Access Permissions4.2.2 Inheritance Flags4.2.3 ZFS per Filesystem ACL Properties4.3 Maintaining File System Integrity4.3.1 Solaris Fingerprint Database (SFD)4.3.2 Solaris Fingerprint Database Command Line Tool4.3.3 Basic Audit Reporting Tool (BART)4.3.4 Integrating BART and the Solaris Fingerprint Database4.3.5 Signed ELF Objects4.4 UFS and NFSv4 Mount Options4.5 ZFS Mount Options4.6 ZFS Delegated AdministrationChapter 5 Privileges and Role-Based Access Control5.1 Traditional UNIX Security Model5.1.1 Shortcomings of the Traditional UNIX Security Model5.1.2 Real and Effective IDs in the Traditional UNIX Model5.2 Solaris Fine-Grained Privileges5.2.1 Solaris Privilege Sets5.2.2 Privilege Bracketing5.2.3 Preventing Privilege Escalation5.2.4 Restricting Process Privileges5.2.5 Solaris Privileges Implementation Details5.3 Solaris Role-Based Access Control5.3.1 Solaris Roles5.3.2 Solaris Rights Profiles5.3.3 Managing Solaris Rights Profiles with the Solaris Management Console5.3.4 Solaris Predefined Rights Profiles5.3.5 Assigning Rights Profiles to Roles5.3.6 How Profiles Are Implemented5.3.7 Rights Profiles and Normal Users5.3.8 Solaris RBAC Components Put Together5.3.9 Using Solaris RBAC Locally5.3.10 Using Solaris RBAC on a Network5.4 Privileges for System Services5.4.1 Authorizations for System ServicesChapter 6 Pluggable Authentication Modules (PAM)6.1 The PAM Framework6.2 The PAM Modules6.2.1 Introduction to PAM Modules6.2.2 The PAM Stacks for Solaris Login6.2.3 Available PAM Modules6.2.4 Standard Solaris PAM Modules6.3 The PAM Configuration File6.3.1 Overview of the pam.conf(4) File6.3.2 PAM Stacks and Module Flags6.3.3 The Module Flags for the PAM Stack for Solaris Login6.4 PAM Consumers6.4.1 PAM Configuration for the Remote Shell Daemon, in.rshd6.4.2 PAM Configuration for sshd6.5 The

Example 2: Encrypted NFS10.10 Interoperability with Microsoft Active Directory10.10.1 Example: Generate a Keytab That Contains a Host Principal's Key on a Windows 2003 ServerChapter 11 Zones Virtualization Security11.1 The Concept of OS Virtualization: Introduction and Motivation11.2 The Architecture of Solaris Zones11.2.1 Branded Zones (BrandZ)11.2.2 Labeled Zones11.2.3 Zones and Networking11.2.4 Zone Identity, CPU Visibility, and Packaging11.2.5 Zones and Devices11.2.6 Resource Control in Zones11.3 Getting Started with Zones11.3.1 Zone Administration11.3.2 Creating, Installing, and Booting a Zone for an Apache Web Server11.4 The Security Advantages of OS Virtualization11.4.1 Isolation and Encapsulation11.4.2 Offering Replicated or Redundant Services Using Zones11.4.3 Hardening Web-Facing Web Server Using Zones11.4.4 A Reduced Set of Privileges for Non-Global Zones11.4.5 Benefits of Exclusive IP Stack Instances11.5 Monitoring Events in Zones11.5.1 Auditing Events in Non-Global ZonesChapter 12 Configuring and Using Trusted Extensions12.1 Why Use Trusted Extensions?12.2 Enabling Trusted Extensions12.3 Getting Started12.3.1 How Labels Are Used12.4 Configuring Your Trusted Network12.4.1 Using the Solaris Management Console12.4.2 Construct Remote Host Templates12.4.3 Adding Hosts to the System's Known Network12.4.4 Assigning a Security Template to a Host or a Group of Hosts12.4.5 Limiting the Hosts That Can Be Contacted on the Trusted Network12.4.6 Configuring Multilevel Network Ports12.5 Creating Users and Roles12.5.1 Creating a Role12.5.2 Creating a User12.6 Creating Labeled Zones12.6.1 Sharing Network Interfaces12.6.2 Creating Your First Zone12.6.3 Creating Your Second Zone12.7 Using the Multilevel Desktop12.7.1 Changing Your Workspace Label12.7.2 Moving Windows into Other Workspaces12.7.3 Copying and Pasting Between Labels12.7.4 File System Protection12.7.5 Observing Processes12.7.6 Assuming a Role12.7.7 Accessing Devices12.7.8 Accessing Your NetworkIndexPrefaceSolaris™ 10 Security EssentialsSolaris™ 10 Security Essentials is the first book in the new series on Solaris system administration. It covers all of the features of the Solaris 10 Operating System that make it the best choice for meeting the present-day challenges to robust and secure computing. Other books in the series are Solaris™ 10 System Administration Essentials and Solaris™ 10 ZFS Essentials. The former covers all of the breakthrough features of the Solaris 10 Operating System in one place. Solaris™ 10 ZFS Essentials provides a hands-on look at the revolutionary new ZFS file system introduced in the Solaris 10 OS.The Solaris OS has a long history of innovation. The Solaris 10 OS is a watershed release that includes features such as: Zones, which provide application isolation and facilitate server consolidation ZFS, the file system that provides a new approach to managing your data with an easy administration interface The Fault Management Architecture, which automates fault detection and resolution The Service Management Facility, a unified model for services and service management on every Solaris system Dynamic Tracing (DTrace), for troubleshooting OS and application problems on production systems in real timeSecurity has long been a core strength of the Solaris OS and it has been significantly enhanced in the Solaris 10 version in areas such as: Zones virtualization security System hardening Trusted Extensions Privileges and Role-Based Access Control

(RBAC) Cryptographic services and key management Auditing Network security Pluggable Authentication Modules (PAM)The strength of Solaris operating system security is its scalability and adaptability. It can protect a single-user system with login authentication to Internet and intranet configurations.This book is the work of the engineers and architects who conceptualized the services, wrote the specifications, and coded the Solaris OS's security features. They bring a wide range of industry and academic experience to the business of creating and deploying secure operating systems. These are the people who know Solaris 10 security best. They have combined to write a book that speaks to readers who want to learn Solaris or who want to use Solaris for the first time in their company's or their own environment. Readers do not have to be experienced Solaris users or operating system developers to take advantage of this book.Books in the Solaris System Administration SeriesSolaris™ 10 System Administration EssentialsSolaris™ 10 System Administration Essentials covers all of the breakthrough features of the Solaris 10 Operating System in one place. It does so in a straightforward way that makes an enterprise-level operating system accessible to system administrators at all levels.Solaris™ 10 System Administration Essentials provides a comprehensive overview along with hands-on examples of the key features that have made Solaris the leading UNIX operating system for years and the significant new features of Solaris 10 that put it far ahead of its competitors. These features include Zones, the ZFS file system, Fault Management Architecture, Service Management Facility, and DTrace, the dynamic tracing tool for troubleshooting OS and application problems on production systems in real time.Solaris™ 10 ZFS EssentialsSolaris™ 10 ZFS Essentials presents the revolutionary Zettabyte File System introduced in Solaris 10. It is a file system that is elegant in its simplicity and the ease with which it allows system administrators to manage data and storage.ZFS is an all-purpose file system that is built on top of a pool of storage devices. File systems that are created from a storage pool share space with the other file systems in the pool. Administrators do not have to allocate storage space based on the intended size of a file system because file systems grow automatically within the space that is allocated to the storage pool. When new storage devices are added, all file systems in the pool can immediately use the additional space.Intended AudienceThe books in the Solaris System Administration Series can benefit anyone who wants to learn more about the Solaris 10 operating system. They are written to be particularly accessible to system administrators who are new to Solaris, and people who are perhaps already serving as administrators in companies running Linux, Windows, and/or other UNIX systems.If you are not presently a practicing system administrator but want to become one, then this series, starting with Solaris™ 10 System Administration Essentials, provides an excellent introduction. In fact, most of the examples used in the books are suited to or can be adapted to small learning environments like a home setup. Even before you venture into corporate system administration or deploy Solaris 10 in your existing IT installation, these books will help you experiment in a small test environment.OpenSolarisIn June 2005, Sun Microsystems introduced OpenSolaris, a fully functional Solaris operating system release built from open source. While the books in this series

focus on Solaris 10, they often incorporate aspects of OpenSolaris. Now that Solaris has been open-sourced, its evolution has accelerated even beyond its normally rapid pace. The authors of this series have often found it interesting to introduce features or nuances that are new in OpenSolaris. At the same, many of the enhancements introduced into OpenSolaris are finding their way into Solaris 10. Whether you are learning Solaris 10 or already have an eye on OpenSolaris, the books in this series are for you.About the AuthorsThis book benefits from the contributions of numerous experts in Solaris security technologies. Below are brief biographies of each of the contributing authors.Glenn Brunette is a Distinguished Engineer, Global Systems Engineering Director, and Chief Security Architect at Sun Microsystems, where he leads a global team focused on information security and assurance. Glenn is the founder of Sun's Systemic Security approach; an OpenSolaris Security Community Leader; the co-founder of the Solaris Security Toolkit software; and a frequent author, contributor, and speaker at both Sun and industry events. Externally, Glenn has served in leadership positions at the National Cyber Security Partnership, the Enterprise Grid Alliance, and the Center for Internet Security.Hai-May Chao is a Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. For the past six years, she has been participating in the design and development of the cryptographic framework and key management framework for the Solaris OS. Hai-May started at Sun in the Trusted Solaris group, developing security features for Trusted Solaris software and the Solaris Management Console. Before joining Sun, Hai-May was a firmware developer at Amdahl, working on IBM-compatible features such as the Multiple Domain Facility (MDF).Martin Englund is a Principal Security Engineer in the Web Engineering organization at Sun Microsystems. He has over sixteen years of experience in data and network security. Web Engineering itself operates the majority of Sun's external Web presence to the Internet and operationally comes under hourly attack. Martin is responsible for the operational security of over 500 Solaris systems that collectively provide the platform for Sun's external Web presence. Prior to this role, Martin worked in Java Security Engineering and the Sun IT Security Office, where he was responsible for creating the standards and tools used by Sun IT to secure its servers. He has also authored numerous technical papers on security, including Sun Blueprints. Before joining Sun, Martin completed studies in Computer Science and Mathematics and worked as a research engineer at the University of Umeå in Sweden.Glenn Faden is a Distinguished Engineer in the Solaris Security Technologies group and has worked at Sun for nineteen years. He is currently the architect for Solaris Trusted Extensions and was one of the architects for the Trusted Solaris OS and role-based access control (RBAC). He designed Sun's multilevel desktops based on Open Look, CDE, and GNOME and holds a patent for the underlying X11 security policy. Glenn has made extensive contributions to the Solaris security foundation, including access control lists, auditing, device allocation, and OS virtualization. He also developed the RBAC and process rights management tools for the Solaris Management Console. He has authored several articles for Sun's Blueprints Web site and the Solaris Developer Connection. Glenn previously worked for Qubix, OmniCad, and Gould Computer

Systems in Desktop Publishing and OS development. He has a Master of Science in Computer Science from the Florida Institute of Technology.Mark Fenwick is a Developer in the Solaris Security Technologies group. His current focus is IPsec and IKE. With over two decades of hardware and UNIX software experience, he has extensive knowledge of firewalls, VPN technology, encryption, and networking protocols. Mark has taught technical classes and presented technical papers to Sun support staff and customers. He closely follows key Sun technologies such as ZFS, Sun Ray, and OpenSolaris, both at work and in his UNIX-only home. In his spare time, he enjoys riding one of his many bicycles and cooking.Valerie Anne Fenwick is a Staff Engineer at Sun Microsystems with over a decade of experience in computer security. She helped to design and develop the cryptographic framework for the Solaris OS. In addition, she chaired the team responsible for code revisions to the Operating System and Networking consolidation for OpenSolaris. Valerie has a Bachelor of Science in Computer Science from Purdue University. In her spare time, she enjoys performing at community theaters, riding her bike, and skiing.Wyllys Ingersoll is a Senior Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. He has been working in the UNIX security field for thirteen years. Since 2000, he has worked on many of the security features in the Solaris OS: Kerberos, the cryptographic framework, and the key management framework, to name a few. Prior to working at Sun, he developed application firewall software for MCI Worldcom and AOL. He received a Master of Science in Computer Science from Virginia Tech in 1996 and a Bachelor of Science in Computer Engineering from Lehigh University in 1988. He lives and works from his home in northern Virginia today.Wolfgang Ley received his diploma in computer science at the German Technical University of Clausthal-Zellerfeld. From 1994 to 1999, he worked at the DFN-CERT (German Computer Emergency Response Team) to build up this new service. During that time, he was also active in the Forum of Incident Response and Security Teams (FIRST). He then joined Sun Microsystems, working as Backend Support Engineer and is dealing with network, security, and kernel internals such as crashdump analysis.Darren Moffat is a Senior Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. Darren is the architect for the Solaris cryptographic framework and Solaris encrypted storage projects. He is also involved in various other OpenSolaris security-related technologies and features. He is an OpenSolaris Security Community leader. Before joining Solaris engineering, Darren worked in SunServices supporting the Trusted Solaris OS, NFS, name services, and Kerberos. Before joining Sun, Darren worked as an analyst/programmer for the UK Ministry of Defence. He is a graduate of the Computing Science Department of Glasgow University.Pravas Kumar Panda received his engineering degree from Biju Patnayak University of Technology in Orissa, India, in 2003 and joined Sun Microsystems soon after that. He has been working on security technologies for the past three years. During this time, he has focused on Kerberos, SSH, PAM, and packet capturing. He has delivered talks on OpenSolaris, secure programing, and security technologies at various universities in India. Pravas enjoys playing the keyboard. He lives in Bangalore with his wife Bobby and their young daughter Avishi.Jan Pechanec works at Sun

Microsystems in the Solaris Security Technologies group. At the moment, he works mostly with technologies around the SSH protocol and the PKCS #11 standard. He graduated in 2001 from the Faculty of Mathematics and Physics at Charles University in Prague, where he majored in operating systems and computer networks. For the last few years, he has also been working at Charles University, lecturing on UNIX and UNIX C programming.Mark Phalan works at the Prague office of Sun Microsystems for the Solaris Security Technologies group. He works on Kerberos-related technologies for the Solaris OS and for OpenSolaris. He graduated in 2003 from the Computer Science department of Trinity College Dublin with a BA(Mod) in Computer Science.Scott Rotondo is a Principal Engineer at Sun Microsystems who works as a security architect for the Solaris operating system. He also serves as President of the Trusted Computing Group (TCG), an industry consortium that defines open standards for hardware-enabled trusted computing. In addition to his work on other security features, Scott leads Sun's development effort to support TCG technologies in Solaris. Scott has over twenty years of technical and management experience in UNIX operating system development. For more than a decade, his focus has been on security issues and features in the Solaris operating system.Christoph Schuba studied mathematics and management information systems at the Universität Heidelberg and at the Universität Mannheim in Germany. As a Fulbright scholar, he earned his Master of Science and Doctor of Philosophy degrees in Computer Science from Purdue University in 1993 and 1997, respectively, performing most of his dissertation research in the Computer Science Laboratory at the Xerox Palo Alto Research Center (PARC). Christoph has taught undergraduate and graduate courses in computer and network security, cryptography, operating systems, and distributed systems at San Jose State University, USA; at the Universtität Heidelberg, Germany; at the International University in Bruchsal, Germany; and at Linkopings universitet in Linköping, Sweden, where he held the chair in information security. Since 1997, Christoph has been working at Sun Labs and most recently in the Solaris Software Security Organization at Sun Microsystems, Inc. He holds 13 patents and is author and co-author of numerous scientific articles in computer and network security.Sharon Read Veach is a Technical Writer at Sun. She specializes in documenting inherent security features of the Solaris OS, such as auditing, cryptography, and IPsec. As part of a team of writers, she worked on the Trusted Solaris OS, the predecessor to the Trusted Extensions software in the Solaris 10 release.Darren Reed is a Senior Systems Engineer in the Solaris Networking group. He is the principal engineer for IPfilter in Solaris.Joep Vesseur is a Staff Engineer working for the Solaris Security Technologies group. His main focus areas are PAM, secure programming practices, and OS and network security in general. He has co-authored Solaris 9 and Solaris 10 Security exams, several Sun Blueprints, and he is a leader of the Dutch OpenSolaris User Group. His background is in digital forensic science and high-performance computing.Paul Wernau is a member of the Solaris Security Technologies group. He started his career in Sun Microsystems doing technical support over the phone for enterprise customers, specializing in networking, firewalls, encrypting gateways, and VPNs. Now an IPsec developer, he tries to remember to

focus on observability and meaningful error messages, keeping in mind his experience with debugging problems with system administrators.1Solaris Security ServicesThe Solaris Operating System (Solaris OS) offers a variety of security services, from single-user login authentication to network protections. In the Solaris 10 release, Sun expands its Solaris OS security offerings to provide containment, encryption, PKCS #11 functionality, Trusted Extensions (labeling), and other advanced security features. Many features are based on open industry standards, such as Kerberos and PKCS #11.This book is written by the engineers and architects who conceptualized the services, wrote the specifications, and coded the security software. This book is intended for users who are interested in maintaining security on a single system and for users who administer a network of systems.1.1 A Solaris Security StoryIn 1988, the Morris worm attacked UNIX systems on the Internet, replicated itself, and brought the network down. This was the first Internet worm.1 Many of the network systems were Sun systems. The Morris worm gained remote access by trying common passwords, using the debug mode of sendmail, and exploiting a buffer overflow in the fingerd program. Countermeasures were complex, because UNIX systems were constructed for openness and connectivity, not security. Today, security is part of Sun's DNA. The kernel and installation are designed to protect Solaris systems from attack.Fast forward to 2007, after Sun distributed Solaris OS code to external developers and users under an OpenSolaris license. An early scrutinizer of the code reported a previously unknown vulnerability in Solaris' telnet program. Because the code was open sourced, the report could and did include a pointer to the lines in the code that were responsible. The report was posted on a Sunday afternoon. By Monday, Sun engineers and testers in Australia, the United Kingdom, and the United States determined how the telnet -f -l root command could compromise a system, posted a Sun Alert, and provided a temporary fix. By Tuesday, Sun issued a patch.2Sun's countermeasures were not complex. Since 1988, Sun has implemented many security features that limit the damage that an attack can inflict. A Secure By Default (SBD) installation closes the telnet port. If the port is open, a simple svcadm command disables the telnet service. Containers, called zones, can limit the effect of an attack, so the system can be reached and fixed by an administrator with the proper privileges.So, unlike the Morris worm, the telnet worm required a small patch for the specific bug. Solaris security features were already in place. The patch was required on all systems that were running the Solaris 10 version of the OS. Solaris administrators were urged to install the patch, and until they could patch the system, they were urged to disable the telnet program.The danger of a telnet worm attack appeared to be averted due to a solid Solaris kernel, external eyes that pinpointed the offending code, a global response by Sun engineers, and the creation, testing, and distribution of a patch by Sun services. That was the engineering part of the fix; then came the applying part of the fix. The patch needed to be applied. Humans, or scripts that were run by humans, needed to patch all affected systems.Two weeks after the patch was released, hackers launched a telnet worm. The worm infected thousands of unpatched systems on the network of the most extensive installation of Solaris machines—SWAN, the Sun-wide area network of Sun

Microsystems.Sun engineers install and re-install their machines to test code. For the most part, these systems are administered by the individual engineer. When the worm invaded these systems, network traffic increased noticeably. Administrative disruption was minimal. The worm could not gain root access, so the SWAN was accessible and systems could be repaired remotely.Within six hours of detecting the worm at Sun, a process was implemented to help Sun engineers both report and repair infected systems. Within 24 hours of the initial outbreak, over 7000 systems had been inoculated by Sun IT and lab network volunteers, and the worm propagation had been controlled on SWAN.The process needed to handle systems that were not managed by Sun IT, such as systems in lab environments, individually managed systems, and systems that had been offline when the repairs were effected. Such systems could get infected upon going online and then spread the worm from within SWAN. Similarly, systems that were subsequently reinstalled with existing JumpStart scripts would re-infect Sun from within SWAN. The inoculation script handled this chaotic environment by locating every newly booted system and configuring it so that the worm could not re-infect. For systems that allowed root login, it disabled the telnet program and because root compromise was possible, advised the administrator in LARGE TYPE to patch the system.So, the Solaris security story is a series of short stories. Limit the attack surface—use an OS with built-in security. Find vulnerabilities quickly—provide the code as open source for more eyes on the code. Create a patch—the underlying security enables quick creation. End a vulnerability—apply the patch. Prevent the attack from being re-introduced—provide comprehensive network management, which might require scripts or firewalls between LANs at a site.1.2 Security Services in the Solaris OSSolaris OS security features protect network packets from being snooped on the wire, protect the system from rogue programs, and limit the damage from denial of service (DoS) attacks. Many features are in place at installation. Some features must be configured and then activated. Other features can be strengthened to meet more stringent requirements. This security strategy enables you to configure your system to have the smallest footprint with the largest security benefits. You decide.Table 1.1 describes the default security configuration and points to the chapters that cover the feature or its use in concert with other features.Table 1.1 Security Features, Default Configurations, and Chapters in Which They Are Covered1.3 Configurable Security Services in the Solaris OSBy editing configuration files and enabling services, you can customize the security profile of the system to match your site's requirements. Some features, such as ACLs and KMF, are easily configured on the command line. Table 1.2 lists the features that can be customized, the reasons why you might customize them, and the chapters where the features are covered.Table 1.2 Configurable Security ServicesNotes1. For more information, see Spafford, Eugene. The Internet Worm Program: an analysis. Purdue Technical Report CDS-TR-823. 1988.2. For a fuller account, see the Sun blogs, http://blogs.sun.com/drscholl/ and http://blogs.sun.com/tpenta/entry/the_in_telnetd_vulnerability_exploit.2Hardening Solaris SystemsSolaris systems are highly configurable. Even before installing additional software, the administrator has the opportunity to make hundreds of choices about how to set up the

operating system itself. These choices include which services to enable, the specific configuration settings of each service, and additional settings that apply to the system as a whole.There is no single “right answer” to any of these configuration choices. If there were, that choice could be set permanently in the operating system, and there would be no need for it to be configurable. Indeed, the process of hardening a computer system is one of making choices and trade-offs that maximize security, while still supporting the intended uses of the system. Beyond the basic techniques presented here, the publications listed at the end of this chapter provide a great deal of specific information to guide the administrator in selecting configuration settings to harden a system.Once the necessary configuration choices have been made and implemented, the system requires regular monitoring to ensure that the system remains in the hardened state. The latter half of this chapter describes tools in the Solaris OS and elsewhere that help to simplify this task.2.1 Securing Network ServicesTraditionally, many UNIX-based systems have made a wide variety of services available to remote clients over the network. By default, Solaris 10 OS installation enables a large number of network services. This open approach is convenient because it makes services available with minimal setup, and it may provide adequate security in environments where access to the network is limited to a known and trusted set of users. However, many Solaris systems are attached to the Internet or to other networks that cannot be fully trusted. In such an environment, it is prudent to limit the set of services provided to remote clients.Of course, a service that listens to the network for requests and provides responses is not necessarily insecure. Such services are typically designed to authenticate the identity of the client in some way before performing a requested operation or divulging any non-public information. However, sometimes even the “public” information that is provided without authentication can be useful to an attacker. More importantly, the programs that implement these services, like all programs, may contain bugs, and accepting input from the network makes it easier for an attacker to exploit remotely and anonymously any vulnerabilities that may exist.To reduce this risk to an acceptable level, the guiding principle is to minimize the set of opportunities available to an attacker, sometimes called the “attack surface,” by providing only the services that are necessary for the purpose of the system. Furthermore, it is important to disable unnecessary services from the moment the system is connected to the network. If a vulnerable service is enabled briefly after installation or each time the system is booted until the service is disabled, then a window of opportunity exists for an attacker to exploit the vulnerability until the service is disabled.2.1.1 Minimizing Installed SoftwareOne approach to limit the network services that are provided by a system is to avoid installing the software that provides unneeded services. All of the programs that are delivered with a Solaris release are divided into packages, with related programs grouped together in the same package. Of course, if a package is not installed, the programs it contains cannot run, and therefore they cannot provide any services that are accessible from the network.Beginning in the Solaris 9 release, the common network services provided by inetd(1M) were divided among several new packages; for example, the packages related to the telnet program are shown below. Discrete packaging

allows the administrator to limit the network services provided by customizing the installation to leave out some packages. Individual packages can also be removed after installation using pkgrm(1M).# pkginfo SUNWtnetc SUNWtnetd SUNWtnetrsystem SUNWtnetc Telnet Command (client)system SUNWtnetd Telnet Server Daemon (Usr)system SUNWtnetr Telnet Server Daemon (Root)The Solaris 10 release expanded upon this idea by adding a Reduced Networking selection at install time to complement the existing Core, End User, Developer, and Entire Distribution choices. Each of these options is an install metacluster, which defines the set of packages to be installed. The Reduced Networking metacluster eliminates as many networking services as possible while leaving the system operational enough to allow further configuration. Of course, this configuration must be done by an administrator at the system console, because there are no services running to allow for remote access and administration.In most cases, a system installed with the Reduced Networking metacluster is too limited to be useful without adding additional packages. Completely eliminating network services has a severe impact on local users as well; for example, a Reduced Networking system does not allow local users to log in via the window system. Rather than using the installed system as-is, the Reduced Networking metacluster is intended to serve as a foundation to which the administrator can add only those packages that are desired for the particular system.While removing packages is an effective way to eliminate network services, this approach has several practical drawbacks. The first is that the level of granularity may be insufficient. Although services are divided among more packages than in the past, it is still common to have several related services provided by a single package, and choosing to install a package means installing all of the services it provides. Second, it is inconvenient to add services later to adapt to changing requirements, since this requires access to the original installation media in order to add the relevant packages. Finally, packages that are added some time after installation will not have the benefit of any bug fixes contained in patches that may have been applied to the system after installation.For details about Sun's support for minimized systems, refer to http://www.opensolaris.org/os/community/security/files/minimization-support-rules-ext.pdf.2.1.2 Using SMF to Control Network ServicesThe Solaris Service Management Facility(SMF), introduced in the Solaris 10 OS, provides a general mechanism for the control and configuration of system services. The fundamental unit of administration in SMF is the service instance. An instance is a specific configuration of a service. For example, a Web server is a service. A single Web server daemon that is configured to accept Web service requests on port 80 is an instance of that service. Multiple instances of the same service can run on a single Solaris OS. Each service instance is given a Fault Management Resource Identifier (FMRI). The FMRI includes the service name and the instance name. When you list a service using the svcs(1) command, the listing includes the status, date, service name, and service instance. In the next case, "network/dhcp-server" identifies the service, and "default" identifies the instance.# svcs -a | grep dhcpdisabled Apr 10 svc:/network/dhcp-server:defaultBecause SMF separates configuration of a service from its installation, it is possible to install the entire Solaris OS but

enable only the services that are needed on the particular system. To simplify the management process, SMF bundles together the configuration settings for many services into a profile. Solaris includes two generic predefined profiles in /var/svc/profile called generic_open.xml and generic_limited_net.xml. The open profile provides the traditional Solaris configuration with many network services enabled. The limited_net profile provides no services to remote clients except for the Secure Shell daemon sshd(1M).An exception is made for sshd because it is frequently necessary to have some mechanism that allows the administrator to log in and administer the system remotely. Since the need to administer the system remotely may occur at unpredictable times, it is impractical to enable sshd only when it is needed. For this reason, sshd is left enabled by default. Secure Shell is a safer choice for remote administration than telnet or rlogin because it protects the privacy and integrity of the communication session. An administrator who does not want to leave sshd enabled can disable this service individually using the svcadm(1M) command as follows:# svcadm disable sshThe goal of the limited_net profile is to avoid providing network services to remote clients without affecting the normal activities of local users. The services affected by the limited_net profile are shown in Table 2.1.Table 2.1 Services Affected by the limited_net ProfileFor services that are normally used only by remote clients, it is straightforward for the profile to simply disable those services. Other services, including several that are associated with the window system, are also needed by local users. For these services, SMF properties are used to configure the services to respond only to requests originating from the local system. Two examples of properties that control the response to remote requests are shown below.# svcprop -t -p config/local_only rpc/bindconfig/local_only boolean true# svcprop -t -p config/log_from_remote system-logconfig/log_from_remote boolean falseThe first example, for the rpcbind service, is typical of services where a new Boolean “local_only” property has been added. When this property value is set to true, the service only accepts requests from the local system. The syslog service, shown in the second example, already contained a “log_from_remote” property in previous releases of the Solaris OS. To limit syslog to accepting data from the local system, this property is set to false.SMF profiles are normally applied by using the svccfg(1M) command. However, while the generic_limited_net profile disables services, it does not set the service properties needed to configure local-only operation. Therefore, the Solaris OS provides the netservices(1M) command that applies the SMF profile and also sets the necessary SMF service properties to restrict services to local requests only.2.1.3 Selecting Profiles at Install Time and AfterwardsWhen a system is upgraded from a previous release of Solaris, the system continues to provide the same services to the network that it did before the upgrade.On the other hand, an initial install presents the opportunity to select the limited networking configuration at install time. During an interactive installation, the user is prompted to choose whether or not to enable the traditional networking services. This choice can also be made automatically during a Jump-Start installation. Currently, a non-interactive installation defaults to the traditional, or open, configuration, but this default may change to the limited networking configuration in future releases of the Solaris OS.As

described later in this chapter, a different SMF profile can be applied at any time after installation. However, the advantage of selecting the limited_net profile during installation is that the system is protected from potentially hostile network traffic from the first time that it boots. If the system boots with services that accept requests from the network, then there is at least a brief period during which the system is vulnerable before the administrator can disable services or limit them to local requests only.For a system that is running in the open configuration, either because of a choice at install time or because it was upgraded from a previous Solaris release, the limited networking configuration can be applied at any time afterwards using the netservices command.# netservices limited2.1.4 Results of Applying Network ProfilesA system that is running the traditional open configuration shows a large number of enabled inetd services.Similarly, the SMF properties for syslogd(1M) and rpcbind(1M) show that they are configured to accept requests from the network.After the limited-networking profile is applied, the set of enabled inetd ser-vices is dramatically reduced.The syslogd and rpcbind services are still enabled for local clients, but these services are configured to ignore requests from the network.The difference can also be seen by examining the output of the nmap1 command, which shows the open ports that are visible to another system on the network. With the open configuration, we see the following open ports:After applying the limited networking configuration, the only open ports are for sshd and rpcbind. Although the rpcbind port is visible to a network scan, it does not accept connections from remote clients due to the setting of the config/local_only property.2.2 Configuration HardeningDisabling or restricting access to network services is just one aspect of operating-system hardening. Once the network attack surface has been minimized and unnecessary services have been disabled, you may want to turn inward and look at how the configuration of the operating system and the remaining operational services can be adjusted to improve the overall security of the system.This section highlights a number of ways in which you can adjust the default configuration to improve the overall security of the system. The degree to which you harden the OS is wholly dependent upon your operational and environmental conditions, but as a general rule the following are generally accepted operating system security recommendations. Protect the OpenBoot PROM Enable Solaris Non-Executable Stacks Log Core Dumps Lock Out AccountsThis list is not intended to be complete, but rather is representative of the types of steps often taken by organizations to further lock down their systems. Each of the steps described in this section changes a Solaris OS default value. Note that systems already in deployment must be evaluated to ensure that even the security settings provided by default are still in place or that any changes are understood and well documented.2.2.1 Protect the OpenBoot PROMSun’s SPARC hardware provides console-security features that can prevent EEPROM changes, hardware command execution, and even system startup without the appropriate password. This password protection only works while the system is at the OpenBoot PROM level (when the operating system is stopped) or immediately after the system is powered on before the operating system is started.The OpenBoot PROM password is not related to the Solaris OS root password, and it

should not be set as the same password. Once set, the OpenBoot PROM password is not displayed, but can be retrieved in clear text form. When changing the OpenBoot PROM password, the system does not ask for the old password prior to changing it to the new one. In some environments, it may make more sense to set the OpenBoot PROM password to something known to the hardware technicians. The two security modes available are command and full.Unless an authorized user has the correct password, the command security mode prevents EEPROM changes and hardware-command execution while at the OpenBoot PROM level. The full security mode provides the restrictions of the command mode and, in addition, does not allow the system to boot without the correct OpenBoot PROM password. The full security mode requires operator interaction to boot the system; the system will not boot without first entering the correct password. Do not use the full security mode on servers or other systems that must boot quickly without manual intervention.2Use the /usr/sbin/eeprom command to set the OpenBoot PROM security mode and password. In this example, the OpenBoot PROM security mode is set to command.2.2.2 Enable Solaris Non-Executable StacksNon-executable stacks provide a very useful technique for thwarting certain kinds of buffer overflow attacks. Initially developed for the SPARC platform, non-executable stack protection is also now available on x64 systems supporting either the Execute Disable (XD) bit on Intel platforms or the No Execute (NX) bit on AMD platforms.As with other kernel parameters, the non-executable stack state is configured (enabled or disabled) by a setting in the /etc/system file. For example, the following statement added to the /etc/system file would enable this feature:set noexec_user_stack=1Once this setting has been made, the system must be rebooted for the change to take effect. It is also possible to compile a program so that it runs with a non-executable stack regardless of the system configuration. For more details applicable to both developers and system administrators, see http://blogs.sun.com/gbrunett/tags/noexstk.2.2.3 Log Core DumpsThe operating system writes out a core file for a process when the process is terminated due to receiving certain signals. Such signals can be generated when the process is unable to handle an unexpected condition. Core dumps can provide an early warning of an attacker attempting to overflow a buffer in a running process.A core file is a disk copy of the contents of the process address space at the time the process received the signal, along with additional information about the state of the process. The Solaris OS allows you to adjust quite a few parameters that are related to the capture and logging of core dumps files.One configuration change often implemented is to send a notice to the system logging service when a program “dumps core.” This kind of notification can be used to trigger administrative action to determine if there is a problem with the service or if the system may be under attack.To enable the logging of core dump events, simply run the /usr/bin/coreadm command:# /usr/bin/coreadm -e logAfter this command is run, future core dump events are logged to the syslog facility.2.2.4 Lock Out AccountsAnother system-wide parameter that is often used is account lockout. Often referred to as a “three strikes policy,” account lockout offers you a way of locking individual user accounts after a specified number of consecutive, failed authentication attempts (note that this number

does not have to be three!).Account lockout can be enabled on a per-user basis or globally where it impacts all users on the system who are not specifically excluded from the policy. Account lockout is also service agnostic, in that the consecutive failed authentication attempts need not be from the same service. In this way, the operating system can detect brute-force authentication attempts that could start with Secure Shell and move to other services like TELNET or FTP.To enable account lockout globally, you set the LOCK_AFTER_RETRIES parameter to YES in the /etc/security/policy.conf file. By default, the root account is excluded from this policy. If you want to have other exclusions or simply want to specify a different per-user policy, you must edit the /etc/user_attr file and/or each impacted user, set the lock_after_retries parameter to either yes or no based on your intent. To specify how many consecutive failed authentication attempts are permitted before an account is locked, set the RETRIES parameter in the /etc/default/login file. By default, this value is 5.Note that once an account is locked, it can only be unlocked by administrative intervention. A system administrator must unlock the account using the -u option to the /usr/bin/passwd command before that account can be successfully accessed.For more information, see http://blogs.sun.com/gbrunett/entry/solaris_10_account_lockout_three.2.2.5 Other Security Configuration IdeasThis section has provided just a small window into the various security hardening configuration settings that exist in the Solaris OS. There are many more systemwide as well as service-specific configuration settings that can be adjusted if necessary. As a general rule, the default values provided in the Solaris OS are suitable for most organizations, but particular cases might require additional configuration changes. Beyond the Solaris product documentation, we recommend that you review the publications that are listed at the end of this chapter for specific hardening suggestions.32.3 Basic Audit and Reporting ToolThe Basic Audit and Reporting Tool (BART) provides a quick and easy way to collect information on filesystem objects and their attributes so that, at a later time, you can determine whether there have been any changes. While this kind of functionality is clearly useful for security incident detection, BART is also often used as part of a larger change management process to validate approved changes and to detect those that may have occurred outside of an approved process.The BART tool, /usr/bin/bart, collects such information as an object's UID, GID, permissions, access control lists, modification time, size, and type. In addition, for files, BART generates an MD5 fingerprint from the contents of the file so that the integrity of a file's contents can also be verified. The exact attributes that are collected depend on the type of object being evaluated. For a full list of the attributes that can be collected, see the bart_rules(4) manual page.BART has two primary modes of operation: create and compare.2.3.1 Collecting Filesystem Object Attributes (Create Mode)When run in create mode, BART collects file system object information from a system. You can control the scope of collection on a system. You can include the entire system, all files under a specified directory, or just a subset of files. You can even define a more granular policy by using a rules file that can be customized to meet your organization's requirements.When you use BART in create mode, BART reads its rules file from either standard input or from a regular file. As BART processes

individual file system objects, it records its results in a manifest file. This manifest is directed to standard output by default, although you can easily redirect the output to a file or to another process.For example, to create a BART manifest called /root/bart.control based upon the files under /usr/lib/nis, the following command could be used:# find /usr/lib/nis | bart create -I > /root/bart.control2.3.2 Comparing Filesystem Object Attributes (Compare Mode)To use BART in compare mode, you need two BART manifests and, optionally, a rules file. The first (and original) manifest, called the control manifest, is used as your baseline. The second manifest, called the test manifest, is then compared against the control. The manifests are compared in accordance with a set of rules, if supplied. If a rules file is specified, then BART uses the rules to determine how to make the various comparisons. One of the benefits of rules files is that they can help eliminate any false alarms in your reports, thereby allowing you to better focus your efforts on the remaining alarms.You can also use BART to compare any two independent BART snapshots to determine whether the objects being assessed have been changed. For example, with BART, you can quickly and easily answer the question: “Has this file changed since yesterday?”For example, building upon the last example, create a test manifest for the /usr/lib/nis directory:# find /usr/lib/nis | bart create -I > /root/bart.testOnce both the control and test manifests have been created, you can compare them to look for any differences:# bart compare /root/bart.control /root/bart.test/usr/lib/nis/foo: add/usr/lib/nis/nisping: uid control:0 test:2In this case, the /usr/lib/nis/foo file has been added to the /usr/lib/nis directory, and the /usr/lib/nis/nisping program has had its owner changed from root (uid 0) to bin (uid 2). These changes happened after the control manifest was first created.Using the cron(1M) command, you can generate BART manifests every minute, hour, day, week, or month. How often you actually run BART to generate new manifests is based on how critical your need is to detect change. Because one size does not fit all, you might even want to consider having different BART rules or policies. For example, you might have two policies: A smaller policy, targeting a few key files, that runs every minute or hour A larger policy, collecting information across the entire system, that runs only once a day or once a weekUse common sense when deciding how often to generate manifests. You need to balance your detection priorities with the I/O load that the collection process generates on the system. For more information on automating the collection and processing of BART manifests, see the Sun BluePrint article titled “Automating Solaris 10 File Integrity Checks,” which can be found at http://www.sun.com/blueprints/0305/819-2259.pdf.2.3.3 Determining Whether a Filesystem Object Is GenuineRegardless of the selected time intervals, however, BART is still not able to definitively answer the question: “Is this a genuine object that Sun shipped?” BART cannot answer this question because you need to manually create the control manifest after the system has been installed. Someone or something could have changed a Sun-provided file from its default before you performed your first BART snapshot, particularly if the first BART manifest is generated well after installation. Remember that, if your control manifest is somehow corrupt, then all of the later comparisons against the original manifest are suspect.2.4 Signed ELF Filesystem ObjectsTo determine if a filesystem object is

indeed genuine, you can query its cryptographic signature. Starting in the Solaris 10 OS, most operating system Executable and Linkable Format (ELF) objects have been cryptographically signed by Sun. The signatures on these files can be validated by using the elfsign(1) command.# elfsign verify -e /usr/bin/suelfsign: verification of /usr/bin/su passed.While this is a very strong way of validating the integrity of a file, this approach is only applicable to ELF objects such as binaries, libraries, device drivers, and other compiled code. Also, due to license or other restrictions, some binaries that are shipped in a Solaris release cannot be modified and therefore the signature cannot be added to those files. Last, text-based configuration and log files and other non-binary content are not protected in this manner. The Solaris Fingerprint Database, discussed in the next section, can validate files that are not signed.2.5 Solaris Fingerprint Database (sfpDB)The Solaris Fingerprint Database (sfpDB) is a free SunSolve Online service that enables organizations to verify the integrity of files that are distributed with the Solaris OS. Organizations can, at any time, query the sfpDB to determine whether an operating system file did indeed come from Sun.The sfpDB is a powerful tool for organizations that want to validate the integrity of their operating systems, baseline snapshots, and patches—or even to assist during digital forensic investigations in which the integrity of objects might be called into question.The Solaris Fingerprint Database is itself a collection of file fingerprints that have been created from the contents of Solaris OS media kits, unbundled software, and patches. These fingerprints are created as part of the release process in order to ensure that the fingerprints are current and reflect actual shipping versions of files provided by Sun. By submitting fingerprints to the Solaris Fingerprint Database service, organizations can then determine which, if any, of the submitted fingerprints belongs to an actual file supplied by Sun.2.5.1 How the sfpDB Tool WorksThe Solaris Fingerprint Database is a collection of file fingerprints that are generated by using the MD5 Message Digest Algorithm, which is defined in RFC 1321:The algorithm takes as input a message of arbitrary length and produces as output a 128-bit “fingerprint” or “message digest” of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest.By using this algorithm, it is expected that no two different files will share the same fingerprint. While this assurance is not necessarily as strong as a cryptographic signature, it is virtually impossible to modify a file in such as way as to retain its original MD5 fingerprint. Note that, in September 2004, researchers announced that the MD5 algorithm was “broken.” That is, they had developed a method to produce two pre-images (for example, files) with the same MD5 hash value. While the announcement is important, this breakthrough does not impact the validity of the Solaris Fingerprint Database. As of the publication of this book, no feasible method exists to produce a pre-image (such as a text or binary file) that matches a given hash value. That is, no file could be replaced with another, different file and still retain its original MD5 hash value. For more information about the resistance of hash functions to attack, see, for example, http://www.schneier.com/crypto-gram-0409.html#3.To use the Solaris Fingerprint Database service,

you calculate MD5 finger-prints for the files you want to check, and then submit them to the database service for processing. The sfpDB service evaluates the submitted fingerprints and determines if a match exists for any of the objects.For each fingerprint match, the information shown in Table 2.2 is provided.Table 2.2 Fingerprint Database FieldsNote that a single file fingerprint can result in several matches. Text files often generate several matches because they do not depend on the underlying operating system version or hardware platform. Examples of text files are configuration files and shell scripts.2.5.2 Creating an MD5 File FingerprintThe Solaris 10 OS contains a program that can be used to generate MD5 finger-prints: /usr/bin/digest. The following example uses the digest program to create an MD5 fingerprint:# digest -v -a md5 /usr/bin/sumd5 (/usr/bin/su) = bab18f089705c1628ccdc177b802d5712.5.3 Testing an MD5 File FingerprintOnce the MD5 fingerprints have been computed, they can be submitted to the Solaris Fingerprint Database to determine whether they correspond to values associated with valid Sun operating system files. To test the collected finger-prints:1. Visit the Solaris Fingerprint Database page at http://sunsolve.sun.com/pub-cgi/fileFingerprints.pl.After connecting to this URL, the Solaris Fingerprint Web form is displayed. Scroll to the bottom of the page to find the text window where the collected fingerprints can be entered.2. Type or copy and paste one or more MD5 fingerprints into the Web form. For example, to verify the su command fingerprint that was generated in the preceding example, paste the MD5 value into the Web form: bab18f089705c1628ccdc177b802d5713. Click the Submit button to view the results.For this example, the following result is returned:The Solaris Fingerprint Database query properly identified the /usr/bin/su program as the object to which the fingerprint belonged. It is important to not only examine the name and canonical path when evaluating the results, as there are many possible outcomes such as multiple data mismatches, matches indicating an upgrade or downgrade attack, and of course, a failure where the fingerprint that you provided was not found in the database. Each of these cases, along with a more in-depth treatment of the Solaris Fingerprint Database, is provided in the Sun BluePrint article titled The Solaris Fingerprint Database: A Security Tool for Solaris Operating Environment Files (Updated for Solaris 10), which can be found at http://www.sun.com/blueprints/0306/816-1148.pdf.It is good practice to use together both of the tools discussed in this chapter: the Basic Audit and Reporting Tool and the Solaris Fingerprint Database. More information on this integration can be found in the Sun BluePrint article titled “Integrating BART and the Solaris Fingerprint Database in the Solaris 10 Operating System,” which can be found at http://www.sun.com/blueprints/0405/819-2260.pdf.ReferencesSolaris™ 10 Security EssentialsSun Microsystems Security EngineersSun Microsystems PressUpper Saddle River, NJ • Boston • Indianapolis • San FranciscoNew York • Toronto • Montreal • London • Munich • Paris • MadridCapetown • Sydney • Tokyo • Singapore • Mexico CitySolaris™ 10 Security EssentialsSun Microsystems Security EngineersSun Microsystems PressUpper Saddle River, NJ • Boston • Indianapolis • San FranciscoNew York • Toronto • Montreal • London • Munich • Paris • MadridCapetown • Sydney • Tokyo • Singapore • Mexico CitySolaris™ 10 Security EssentialsSun Microsystems

Command7.2.2 encrypt and decrypt Commands7.2.3 elfsign Command7.2.4 OpenSSL Libraries7.2.5 Apache Web Server7.3 Administration of the Solaris Cryptographic Framework7.3.1 cryptoadm list Subcommand7.3.2 cryptoadm enable and disable Subcommands7.3.3 cryptoadm install, uninstall, load, and unload Subcommands7.4 Hardware Acceleration7.4.1 Solaris Drivers7.4.2 How to Use Existing Hardware Providers in the Solaris OS7.5 Examples of Using the Cryptographic Framework7.5.1 Troubleshooting the Cryptographic Framework7.5.2 Determining What Encryption Is Being Handled in Hardware7.5.3 Using the Cryptographic Framework Through NSS7.5.4 Configuring Java to Use the Solaris Cryptographic Framework7.5.5 Configuring Apache Web Server to Use the Cryptographic FrameworkChapter 8 Key Management Framework (KMF)8.1 Key Management Administrative Utility8.1.1 pktool(1)8.2 KMF Policy-Enforcement Mechanism8.3 Key Management Policy Configuration Utility8.3.1 kmfcfg(1)8.4 KMF Programming InterfacesChapter 9 Auditing9.1 Introduction and Background9.1.1 Auditing Principles9.1.2 Goals of Auditing9.2 Definitions and Concepts9.2.1 Preselection Mask9.2.2 Audit Trail9.2.3 Audit User ID9.2.4 Audit Policy9.2.5 Audit Quickstart Guide9.3 Configuring Auditing9.3.1 audit_class File9.3.2 audit_event File9.3.3 audit_control File9.3.4 audit_user File9.3.5 Audit Policy9.3.6 Enabling Auditing9.3.7 Verifying audit_control and audit_user Changes9.3.8 audit_warn Script9.4 Analyzing the Audit Trail9.4.1 Details of an Audit Record9.4.2 Examining Parts of the Audit Trail9.5 Managing the Audit Trail9.5.1 Log Rotation9.5.2 Remote Storage9.5.3 Compression9.6 Common Auditing Customizations9.6.1 Configuring an Audit Review Role That Is Not Audited9.6.2 Using the audit_syslog Plug-in9.6.3 Creating Your Own Audit ClassChapter 10 Solaris Network Security10.1 IP Filter10.1.1 IP Filter Configuration10.1.2 Starting Out10.1.3 Stateful Versus Stateless Filtering10.1.4 Troubleshooting IP Filter10.1.5 IP Filter in Zones10.1.6 Filtering Inter-Zone Packets—Loopback Packet Filtering10.1.7 IP Filter and Anti-Spoofing10.1.8 Using NAT with IP Filter10.1.9 Logging with IP Filter10.2 What Is IPsec?10.2.1 What Do IPsec Packets Look Like?10.2.2 How Is IPsec Configured?10.3 Solaris Secure Shell (SunSSH)10.3.1 Versions of SunSSH10.3.2 Notable SunSSH Differences from OpenSSH10.3.3 Starting and Stopping SunSSH10.4 Configuring SunSSH10.4.1 How the SSH Protocol Version 2 Works10.4.2 Authentication Methods10.4.3 SunSSH Commands10.5 OpenSSL10.5.1 PKCS #11 Engine10.6 Kerberos10.6.1 Why Use Kerberos?10.6.2 Introduction to Kerberos10.7 Kerberos in the Solaris OS10.7.1 Configuring a KDC10.7.2 Create the Kerberos Database10.7.3 Kerberos and LDAP10.8 Kerberos Administration10.8.1 Configure the kadmind Daemon10.8.2 Keytabs and Passwords10.8.3 Slave KDCs10.8.4 Configuring Kerberos Clients10.9 Application Servers10.9.1 Example 1: SunSSH10.9.2 Kerberos Authorization10.9.3 Example 2: Encrypted NFS10.10 Interoperability with Microsoft Active Directory10.10.1 Example: Generate a Keytab That Contains a Host Principal's Key on a Windows 2003 ServerChapter 11 Zones Virtualization Security11.1 The Concept of OS Virtualization: Introduction and Motivation11.2 The Architecture of Solaris Zones11.2.1 Branded Zones (BrandZ)11.2.2 Labeled Zones11.2.3 Zones and Networking11.2.4 Zone Identity, CPU Visibility,

and Packaging11.2.5 Zones and Devices11.2.6 Resource Control in Zones11.3 Getting Started with Zones11.3.1 Zone Administration11.3.2 Creating, Installing, and Booting a Zone for an Apache Web Server11.4 The Security Advantages of OS Virtualization11.4.1 Isolation and Encapsulation11.4.2 Offering Replicated or Redundant Services Using Zones11.4.3 Hardening Web-Facing Web Server Using Zones11.4.4 A Reduced Set of Privileges for Non-Global Zones11.4.5 Benefits of Exclusive IP Stack Instances11.5 Monitoring Events in Zones11.5.1 Auditing Events in Non-Global ZonesChapter 12 Configuring and Using Trusted Extensions12.1 Why Use Trusted Extensions?12.2 Enabling Trusted Extensions12.3 Getting Started12.3.1 How Labels Are Used12.4 Configuring Your Trusted Network12.4.1 Using the Solaris Management Console12.4.2 Construct Remote Host Templates12.4.3 Adding Hosts to the System's Known Network12.4.4 Assigning a Security Template to a Host or a Group of Hosts12.4.5 Limiting the Hosts That Can Be Contacted on the Trusted Network12.4.6 Configuring Multilevel Network Ports12.5 Creating Users and Roles12.5.1 Creating a Role12.5.2 Creating a User12.6 Creating Labeled Zones12.6.1 Sharing Network Interfaces12.6.2 Creating Your First Zone12.6.3 Creating Your Second Zone12.7 Using the Multilevel Desktop12.7.1 Changing Your Workspace Label12.7.2 Moving Windows into Other Workspaces12.7.3 Copying and Pasting Between Labels12.7.4 File System Protection12.7.5 Observing Processes12.7.6 Assuming a Role12.7.7 Accessing Devices12.7.8 Accessing Your NetworkIndexContentsPrefaceAbout the AuthorsChapter 1 Solaris Security Services1.1 A Solaris Security Story1.2 Security Services in the Solaris OS1.3 Configurable Security Services in the Solaris OSChapter 2 Hardening Solaris Systems2.1 Securing Network Services2.1.1 Minimizing Installed Software2.1.2 Using SMF to Control Network Services2.1.3 Selecting Profiles at Install Time and Afterwards2.1.4 Results of Applying Network Profiles2.2 Configuration Hardening2.2.1 Protect the OpenBoot PROM2.2.2 Enable Solaris Non-Executable Stacks2.2.3 Log Core Dumps2.2.4 Lock Out Accounts2.2.5 Other Security Configuration Ideas2.3 Basic Audit and Reporting Tool2.3.1 Collecting Filesystem Object Attributes (Create Mode)2.3.2 Comparing Filesystem Object Attributes (Compare Mode)2.3.3 Determining Whether a Filesystem Object Is Genuine2.4 Signed ELF Filesystem Objects2.5 Solaris Fingerprint Database (sfpDB)2.5.1 How the sfpDB Tool Works2.5.2 Creating an MD5 File Fingerprint2.5.3 Testing an MD5 File FingerprintChapter 3 System Protection with SMF3.1 Service Management Facility (SMF)3.2 How SMF Configuration Works3.3 Modifying Solaris Services Defaults3.3.1 Configuring the NFS Service3.3.2 Configuring the IP Filter Service3.3.3 Configuring the ftp Service3.3.4 Configuring the Apache2 Web ServiceChapter 4 File System Security4.1 Traditional UNIX File System Security4.1.1 setuid, setgid, and Sticky Bit4.1.2 chown/chgrp4.1.3 Default umask4.1.4 setfacl/getfacl4.1.5 Extended Attributes4.1.6 Promiscuous Execution4.2 ZFS/NFSv4 ACLs4.2.1 Access Permissions4.2.2 Inheritance Flags4.2.3 ZFS per Filesystem ACL Properties4.3 Maintaining File System Integrity4.3.1 Solaris Fingerprint Database (SFD)4.3.2 Solaris Fingerprint Database Command Line Tool4.3.3 Basic Audit Reporting Tool (BART)4.3.4 Integrating BART and the Solaris Fingerprint Database4.3.5 Signed ELF Objects4.4 UFS and NFSv4 Mount Options4.5

audit_warn Script9.4 Analyzing the Audit Trail9.4.1 Details of an Audit Record9.4.2 Examining Parts of the Audit Trail9.5 Managing the Audit Trail9.5.1 Log Rotation9.5.2 Remote Storage9.5.3 Compression9.6 Common Auditing Customizations9.6.1 Configuring an Audit Review Role That Is Not Audited9.6.2 Using the audit_syslog Plug-in9.6.3 Creating Your Own Audit ClassChapter 10 Solaris Network Security10.1 IP Filter10.1.1 IP Filter Configuration10.1.2 Starting Out10.1.3 Stateful Versus Stateless Filtering10.1.4 Troubleshooting IP Filter10.1.5 IP Filter in Zones10.1.6 Filtering Inter-Zone Packets—Loopback Packet Filtering10.1.7 IP Filter and Anti-Spoofing10.1.8 Using NAT with IP Filter10.1.9 Logging with IP Filter10.2 What Is IPsec?10.2.1 What Do IPsec Packets Look Like?10.2.2 How Is IPsec Configured?10.3 Solaris Secure Shell (SunSSH)10.3.1 Versions of SunSSH10.3.2 Notable SunSSH Differences from OpenSSH10.3.3 Starting and Stopping SunSSH10.4 Configuring SunSSH10.4.1 How the SSH Protocol Version 2 Works10.4.2 Authentication Methods10.4.3 SunSSH Commands10.5 OpenSSL10.5.1 PKCS #11 Engine10.6 Kerberos10.6.1 Why Use Kerberos?10.6.2 Introduction to Kerberos10.7 Kerberos in the Solaris OS10.7.1 Configuring a KDC10.7.2 Create the Kerberos Database10.7.3 Kerberos and LDAP10.8 Kerberos Administration10.8.1 Configure the kadmind Daemon10.8.2 Keytabs and Passwords10.8.3 Slave KDCs10.8.4 Configuring Kerberos Clients10.9 Application Servers10.9.1 Example 1: SunSSH10.9.2 Kerberos Authorization10.9.3 Example 2: Encrypted NFS10.10 Interoperability with Microsoft Active Directory10.10.1 Example: Generate a Keytab That Contains a Host Principal's Key on a Windows 2003 ServerChapter 11 Zones Virtualization Security11.1 The Concept of OS Virtualization: Introduction and Motivation11.2 The Architecture of Solaris Zones11.2.1 Branded Zones (BrandZ)11.2.2 Labeled Zones11.2.3 Zones and Networking11.2.4 Zone Identity, CPU Visibility, and Packaging11.2.5 Zones and Devices11.2.6 Resource Control in Zones11.3 Getting Started with Zones11.3.1 Zone Administration11.3.2 Creating, Installing, and Booting a Zone for an Apache Web Server11.4 The Security Advantages of OS Virtualization11.4.1 Isolation and Encapsulation11.4.2 Offering Replicated or Redundant Services Using Zones11.4.3 Hardening Web-Facing Web Server Using Zones11.4.4 A Reduced Set of Privileges for Non-Global Zones11.4.5 Benefits of Exclusive IP Stack Instances11.5 Monitoring Events in Zones11.5.1 Auditing Events in Non-Global ZonesChapter 12 Configuring and Using Trusted Extensions12.1 Why Use Trusted Extensions?12.2 Enabling Trusted Extensions12.3 Getting Started12.3.1 How Labels Are Used12.4 Configuring Your Trusted Network12.4.1 Using the Solaris Management Console12.4.2 Construct Remote Host Templates12.4.3 Adding Hosts to the System's Known Network12.4.4 Assigning a Security Template to a Host or a Group of Hosts12.4.5 Limiting the Hosts That Can Be Contacted on the Trusted Network12.4.6 Configuring Multilevel Network Ports12.5 Creating Users and Roles12.5.1 Creating a Role12.5.2 Creating a User12.6 Creating Labeled Zones12.6.1 Sharing Network Interfaces12.6.2 Creating Your First Zone12.6.3 Creating Your Second Zone12.7 Using the Multilevel Desktop12.7.1 Changing Your Workspace Label12.7.2 Moving Windows into Other Workspaces12.7.3 Copying and Pasting Between Labels12.7.4 File System Protection12.7.5 Observing Processes12.7.6 Assuming a Role12.7.7

Accessing Devices12.7.8 Accessing Your NetworkIndexContentsPrefaceAbout the AuthorsChapter 1 Solaris Security Services1.1 A Solaris Security Story1.2 Security Services in the Solaris OS1.3 Configurable Security Services in the Solaris OSChapter 2 Hardening Solaris Systems2.1 Securing Network Services2.1.1 Minimizing Installed Software2.1.2 Using SMF to Control Network Services2.1.3 Selecting Profiles at Install Time and Afterwards2.1.4 Results of Applying Network Profiles2.2 Configuration Hardening2.2.1 Protect the OpenBoot PROM2.2.2 Enable Solaris Non-Executable Stacks2.2.3 Log Core Dumps2.2.4 Lock Out Accounts2.2.5 Other Security Configuration Ideas2.3 Basic Audit and Reporting Tool2.3.1 Collecting Filesystem Object Attributes (Create Mode)2.3.2 Comparing Filesystem Object Attributes (Compare Mode)2.3.3 Determining Whether a Filesystem Object Is Genuine2.4 Signed ELF Filesystem Objects2.5 Solaris Fingerprint Database (sfpDB)2.5.1 How the sfpDB Tool Works2.5.2 Creating an MD5 File Fingerprint2.5.3 Testing an MD5 File FingerprintChapter 3 System Protection with SMF3.1 Service Management Facility (SMF)3.2 How SMF Configuration Works3.3 Modifying Solaris Services Defaults3.3.1 Configuring the NFS Service3.3.2 Configuring the IP Filter Service3.3.3 Configuring the ftp Service3.3.4 Configuring the Apache2 Web ServiceChapter 4 File System Security4.1 Traditional UNIX File System Security4.1.1 setuid, setgid, and Sticky Bit4.1.2 chown/chgrp4.1.3 Default umask4.1.4 setfacl/getfacl4.1.5 Extended Attributes4.1.6 Promiscuous Execution4.2 ZFS/NFSv4 ACLs4.2.1 Access Permissions4.2.2 Inheritance Flags4.2.3 ZFS per Filesystem ACL Properties4.3 Maintaining File System Integrity4.3.1 Solaris Fingerprint Database (SFD)4.3.2 Solaris Fingerprint Database Command Line Tool4.3.3 Basic Audit Reporting Tool (BART)4.3.4 Integrating BART and the Solaris Fingerprint Database4.3.5 Signed ELF Objects4.4 UFS and NFSv4 Mount Options4.5 ZFS Mount Options4.6 ZFS Delegated AdministrationChapter 5 Privileges and Role-Based Access Control5.1 Traditional UNIX Security Model5.1.1 Shortcomings of the Traditional UNIX Security Model5.1.2 Real and Effective IDs in the Traditional UNIX Model5.2 Solaris Fine-Grained Privileges5.2.1 Solaris Privilege Sets5.2.2 Privilege Bracketing5.2.3 Preventing Privilege Escalation5.2.4 Restricting Process Privileges5.2.5 Solaris Privileges Implementation Details5.3 Solaris Role-Based Access Control5.3.1 Solaris Roles5.3.2 Solaris Rights Profiles5.3.3 Managing Solaris Rights Profiles with the Solaris Management Console5.3.4 Solaris Predefined Rights Profiles5.3.5 Assigning Rights Profiles to Roles5.3.6 How Profiles Are Implemented5.3.7 Rights Profiles and Normal Users5.3.8 Solaris RBAC Components Put Together5.3.9 Using Solaris RBAC Locally5.3.10 Using Solaris RBAC on a Network5.4 Privileges for System Services5.4.1 Authorizations for System ServicesChapter 6 Pluggable Authentication Modules (PAM)6.1 The PAM Framework6.2 The PAM Modules6.2.1 Introduction to PAM Modules6.2.2 The PAM Stacks for Solaris Login6.2.3 Available PAM Modules6.2.4 Standard Solaris PAM Modules6.3 The PAM Configuration File6.3.1 Overview of the pam.conf(4) File6.3.2 PAM Stacks and Module Flags6.3.3 The Module Flags for the PAM Stack for Solaris Login6.4 PAM Consumers6.4.1 PAM Configuration for the Remote Shell Daemon, in.rshd6.4.2 PAM Configuration for sshd6.5 The PAM Library6.6 PAM Tasks6.6.1 Adding

Additional PAM Modules6.6.2 Debugging PAM StacksChapter 7 Solaris Cryptographic Framework7.1 PKCS #11 Standard and Library7.1.1 Consumers7.1.2 Providers and Plug-ins7.1.3 Mechanism7.1.4 Tokens7.1.5 Slots7.1.6 PKCS #11 Shared Library and the Software Token7.1.7 Metaslot7.1.8 Kernel Hardware Providers7.1.9 Application Access to Hardware Providers7.1.10 Kernel Software Providers7.2 User-Level Commands7.2.1 digest Command7.2.2 encrypt and decrypt Commands7.2.3 elfsign Command7.2.4 OpenSSL Libraries7.2.5 Apache Web Server7.3 Administration of the Solaris Cryptographic Framework7.3.1 cryptoadm list Subcommand7.3.2 cryptoadm enable and disable Subcommands7.3.3 cryptoadm install, uninstall, load, and unload Subcommands7.4 Hardware Acceleration7.4.1 Solaris Drivers7.4.2 How to Use Existing Hardware Providers in the Solaris OS7.5 Examples of Using the Cryptographic Framework7.5.1 Troubleshooting the Cryptographic Framework7.5.2 Determining What Encryption Is Being Handled in Hardware7.5.3 Using the Cryptographic Framework Through NSS7.5.4 Configuring Java to Use the Solaris Cryptographic Framework7.5.5 Configuring Apache Web Server to Use the Cryptographic FrameworkChapter 8 Key Management Framework (KMF)8.1 Key Management Administrative Utility8.1.1 pktool(1)8.2 KMF Policy-Enforcement Mechanism8.3 Key Management Policy Configuration Utility8.3.1 kmfcfg(1)8.4 KMF Programming InterfacesChapter 9 Auditing9.1 Introduction and Background9.1.1 Auditing Principles9.1.2 Goals of Auditing9.2 Definitions and Concepts9.2.1 Preselection Mask9.2.2 Audit Trail9.2.3 Audit User ID9.2.4 Audit Policy9.2.5 Audit Quickstart Guide9.3 Configuring Auditing9.3.1 audit_class File9.3.2 audit_event File9.3.3 audit_control File9.3.4 audit_user File9.3.5 Audit Policy9.3.6 Enabling Auditing9.3.7 Verifying audit_control and audit_user Changes9.3.8 audit_warn Script9.4 Analyzing the Audit Trail9.4.1 Details of an Audit Record9.4.2 Examining Parts of the Audit Trail9.5 Managing the Audit Trail9.5.1 Log Rotation9.5.2 Remote Storage9.5.3 Compression9.6 Common Auditing Customizations9.6.1 Configuring an Audit Review Role That Is Not Audited9.6.2 Using the audit_syslog Plug-in9.6.3 Creating Your Own Audit ClassChapter 10 Solaris Network Security10.1 IP Filter10.1.1 IP Filter Configuration10.1.2 Starting Out10.1.3 Stateful Versus Stateless Filtering10.1.4 Troubleshooting IP Filter10.1.5 IP Filter in Zones10.1.6 Filtering Inter-Zone Packets—Loopback Packet Filtering10.1.7 IP Filter and Anti-Spoofing10.1.8 Using NAT with IP Filter10.1.9 Logging with IP Filter10.2 What Is IPsec?10.2.1 What Do IPsec Packets Look Like?10.2.2 How Is IPsec Configured?10.3 Solaris Secure Shell (SunSSH)10.3.1 Versions of SunSSH10.3.2 Notable SunSSH Differences from OpenSSH10.3.3 Starting and Stopping SunSSH10.4 Configuring SunSSH10.4.1 How the SSH Protocol Version 2 Works10.4.2 Authentication Methods10.4.3 SunSSH Commands10.5 OpenSSL10.5.1 PKCS #11 Engine10.6 Kerberos10.6.1 Why Use Kerberos?10.6.2 Introduction to Kerberos10.7 Kerberos in the Solaris OS10.7.1 Configuring a KDC10.7.2 Create the Kerberos Database10.7.3 Kerberos and LDAP10.8 Kerberos Administration10.8.1 Configure the kadmind Daemon10.8.2 Keytabs and Passwords10.8.3 Slave KDCs10.8.4 Configuring Kerberos Clients10.9 Application Servers10.9.1 Example 1: SunSSH10.9.2 Kerberos Authorization10.9.3

Example 2: Encrypted NFS10.10 Interoperability with Microsoft Active Directory10.10.1 Example: Generate a Keytab That Contains a Host Principal's Key on a Windows 2003 ServerChapter 11 Zones Virtualization Security11.1 The Concept of OS Virtualization: Introduction and Motivation11.2 The Architecture of Solaris Zones11.2.1 Branded Zones (BrandZ)11.2.2 Labeled Zones11.2.3 Zones and Networking11.2.4 Zone Identity, CPU Visibility, and Packaging11.2.5 Zones and Devices11.2.6 Resource Control in Zones11.3 Getting Started with Zones11.3.1 Zone Administration11.3.2 Creating, Installing, and Booting a Zone for an Apache Web Server11.4 The Security Advantages of OS Virtualization11.4.1 Isolation and Encapsulation11.4.2 Offering Replicated or Redundant Services Using Zones11.4.3 Hardening Web-Facing Web Server Using Zones11.4.4 A Reduced Set of Privileges for Non-Global Zones11.4.5 Benefits of Exclusive IP Stack Instances11.5 Monitoring Events in Zones11.5.1 Auditing Events in Non-Global ZonesChapter 12 Configuring and Using Trusted Extensions12.1 Why Use Trusted Extensions?12.2 Enabling Trusted Extensions12.3 Getting Started12.3.1 How Labels Are Used12.4 Configuring Your Trusted Network12.4.1 Using the Solaris Management Console12.4.2 Construct Remote Host Templates12.4.3 Adding Hosts to the System's Known Network12.4.4 Assigning a Security Template to a Host or a Group of Hosts12.4.5 Limiting the Hosts That Can Be Contacted on the Trusted Network12.4.6 Configuring Multilevel Network Ports12.5 Creating Users and Roles12.5.1 Creating a Role12.5.2 Creating a User12.6 Creating Labeled Zones12.6.1 Sharing Network Interfaces12.6.2 Creating Your First Zone12.6.3 Creating Your Second Zone12.7 Using the Multilevel Desktop12.7.1 Changing Your Workspace Label12.7.2 Moving Windows into Other Workspaces12.7.3 Copying and Pasting Between Labels12.7.4 File System Protection12.7.5 Observing Processes12.7.6 Assuming a Role12.7.7 Accessing Devices12.7.8 Accessing Your NetworkIndexPrefaceSolaris™ 10 Security EssentialsSolaris™ 10 Security Essentials is the first book in the new series on Solaris system administration. It covers all of the features of the Solaris 10 Operating System that make it the best choice for meeting the present-day challenges to robust and secure computing. Other books in the series are Solaris™ 10 System Administration Essentials and Solaris™ 10 ZFS Essentials. The former covers all of the breakthrough features of the Solaris 10 Operating System in one place. Solaris™ 10 ZFS Essentials provides a hands-on look at the revolutionary new ZFS file system introduced in the Solaris 10 OS.The Solaris OS has a long history of innovation. The Solaris 10 OS is a watershed release that includes features such as: Zones, which provide application isolation and facilitate server consolidation ZFS, the file system that provides a new approach to managing your data with an easy administration interface The Fault Management Architecture, which automates fault detection and resolution The Service Management Facility, a unified model for services and service management on every Solaris system Dynamic Tracing (DTrace), for troubleshooting OS and application problems on production systems in real timeSecurity has long been a core strength of the Solaris OS and it has been significantly enhanced in the Solaris 10 version in areas such as: Zones virtualization security System hardening Trusted Extensions Privileges and Role-Based Access Control

(RBAC) Cryptographic services and key management Auditing Network security Pluggable Authentication Modules (PAM)The strength of Solaris operating system security is its scalability and adaptability. It can protect a single-user system with login authentication to Internet and intranet configurations.This book is the work of the engineers and architects who conceptualized the services, wrote the specifications, and coded the Solaris OS's security features. They bring a wide range of industry and academic experience to the business of creating and deploying secure operating systems. These are the people who know Solaris 10 security best. They have combined to write a book that speaks to readers who want to learn Solaris or who want to use Solaris for the first time in their company's or their own environment. Readers do not have to be experienced Solaris users or operating system developers to take advantage of this book.Books in the Solaris System Administration SeriesSolaris™ 10 System Administration EssentialsSolaris™ 10 System Administration Essentials covers all of the breakthrough features of the Solaris 10 Operating System in one place. It does so in a straightforward way that makes an enterprise-level operating system accessible to system administrators at all levels.Solaris™ 10 System Administration Essentials provides a comprehensive overview along with hands-on examples of the key features that have made Solaris the leading UNIX operating system for years and the significant new features of Solaris 10 that put it far ahead of its competitors. These features include Zones, the ZFS file system, Fault Management Architecture, Service Management Facility, and DTrace, the dynamic tracing tool for troubleshooting OS and application problems on production systems in real time.Solaris™ 10 ZFS EssentialsSolaris™ 10 ZFS Essentials presents the revolutionary Zettabyte File System introduced in Solaris 10. It is a file system that is elegant in its simplicity and the ease with which it allows system administrators to manage data and storage.ZFS is an all-purpose file system that is built on top of a pool of storage devices. File systems that are created from a storage pool share space with the other file systems in the pool. Administrators do not have to allocate storage space based on the intended size of a file system because file systems grow automatically within the space that is allocated to the storage pool. When new storage devices are added, all file systems in the pool can immediately use the additional space.Intended AudienceThe books in the Solaris System Administration Series can benefit anyone who wants to learn more about the Solaris 10 operating system. They are written to be particularly accessible to system administrators who are new to Solaris, and people who are perhaps already serving as administrators in companies running Linux, Windows, and/or other UNIX systems.If you are not presently a practicing system administrator but want to become one, then this series, starting with Solaris™ 10 System Administration Essentials, provides an excellent introduction. In fact, most of the examples used in the books are suited to or can be adapted to small learning environments like a home setup. Even before you venture into corporate system administration or deploy Solaris 10 in your existing IT installation, these books will help you experiment in a small test environment.OpenSolarisIn June 2005, Sun Microsystems introduced OpenSolaris, a fully functional Solaris operating system release built from open source. While the books in this series

focus on Solaris 10, they often incorporate aspects of OpenSolaris. Now that Solaris has been open-sourced, its evolution has accelerated even beyond its normally rapid pace. The authors of this series have often found it interesting to introduce features or nuances that are new in OpenSolaris. At the same, many of the enhancements introduced into OpenSolaris are finding their way into Solaris 10. Whether you are learning Solaris 10 or already have an eye on OpenSolaris, the books in this series are for you.PrefaceSolaris™ 10 Security EssentialsSolaris™ 10 Security Essentials is the first book in the new series on Solaris system administration. It covers all of the features of the Solaris 10 Operating System that make it the best choice for meeting the present-day challenges to robust and secure computing. Other books in the series are Solaris™ 10 System Administration Essentials and Solaris™ 10 ZFS Essentials. The former covers all of the breakthrough features of the Solaris 10 Operating System in one place. Solaris™ 10 ZFS Essentials provides a hands-on look at the revolutionary new ZFS file system introduced in the Solaris 10 OS.The Solaris OS has a long history of innovation. The Solaris 10 OS is a watershed release that includes features such as: Zones, which provide application isolation and facilitate server consolidation ZFS, the file system that provides a new approach to managing your data with an easy administration interface The Fault Management Architecture, which automates fault detection and resolution The Service Management Facility, a unified model for services and service management on every Solaris system Dynamic Tracing (DTrace), for troubleshooting OS and application problems on production systems in real timeSecurity has long been a core strength of the Solaris OS and it has been significantly enhanced in the Solaris 10 version in areas such as: Zones virtualization security System hardening Trusted Extensions Privileges and Role-Based Access Control (RBAC) Cryptographic services and key management Auditing Network security Pluggable Authentication Modules (PAM)The strength of Solaris operating system security is its scalability and adaptability. It can protect a single-user system with login authentication to Internet and intranet configurations.This book is the work of the engineers and architects who conceptualized the services, wrote the specifications, and coded the Solaris OS's security features. They bring a wide range of industry and academic experience to the business of creating and deploying secure operating systems. These are the people who know Solaris 10 security best. They have combined to write a book that speaks to readers who want to learn Solaris or who want to use Solaris for the first time in their company's or their own environment. Readers do not have to be experienced Solaris users or operating system developers to take advantage of this book.Books in the Solaris System Administration SeriesSolaris™ 10 System Administration EssentialsSolaris™ 10 System Administration Essentials covers all of the breakthrough features of the Solaris 10 Operating System in one place. It does so in a straightforward way that makes an enterprise-level operating system accessible to system administrators at all levels.Solaris™ 10 System Administration Essentials provides a comprehensive overview along with hands-on examples of the key features that have made Solaris the leading UNIX operating system for years and the significant new features of Solaris 10 that put it far ahead of its competitors. These

features include Zones, the ZFS file system, Fault Management Architecture, Service Management Facility, and DTrace, the dynamic tracing tool for troubleshooting OS and application problems on production systems in real time.Solaris™ 10 ZFS EssentialsSolaris™ 10 ZFS Essentials presents the revolutionary Zettabyte File System introduced in Solaris 10. It is a file system that is elegant in its simplicity and the ease with which it allows system administrators to manage data and storage.ZFS is an all-purpose file system that is built on top of a pool of storage devices. File systems that are created from a storage pool share space with the other file systems in the pool. Administrators do not have to allocate storage space based on the intended size of a file system because file systems grow automatically within the space that is allocated to the storage pool. When new storage devices are added, all file systems in the pool can immediately use the additional space.Intended AudienceThe books in the Solaris System Administration Series can benefit anyone who wants to learn more about the Solaris 10 operating system. They are written to be particularly accessible to system administrators who are new to Solaris, and people who are perhaps already serving as administrators in companies running Linux, Windows, and/or other UNIX systems.If you are not presently a practicing system administrator but want to become one, then this series, starting with Solaris™ 10 System Administration Essentials, provides an excellent introduction. In fact, most of the examples used in the books are suited to or can be adapted to small learning environments like a home setup. Even before you venture into corporate system administration or deploy Solaris 10 in your existing IT installation, these books will help you experiment in a small test environment.OpenSolarisIn June 2005, Sun Microsystems introduced OpenSolaris, a fully functional Solaris operating system release built from open source. While the books in this series focus on Solaris 10, they often incorporate aspects of OpenSolaris. Now that Solaris has been open-sourced, its evolution has accelerated even beyond its normally rapid pace. The authors of this series have often found it interesting to introduce features or nuances that are new in OpenSolaris. At the same, many of the enhancements introduced into OpenSolaris are finding their way into Solaris 10. Whether you are learning Solaris 10 or already have an eye on OpenSolaris, the books in this series are for you.PrefaceSolaris™ 10 Security EssentialsSolaris™ 10 Security Essentials is the first book in the new series on Solaris system administration. It covers all of the features of the Solaris 10 Operating System that make it the best choice for meeting the present-day challenges to robust and secure computing. Other books in the series are Solaris™ 10 System Administration Essentials and Solaris™ 10 ZFS Essentials. The former covers all of the breakthrough features of the Solaris 10 Operating System in one place. Solaris™ 10 ZFS Essentials provides a hands-on look at the revolutionary new ZFS file system introduced in the Solaris 10 OS.The Solaris OS has a long history of innovation. The Solaris 10 OS is a watershed release that includes features such as: Zones, which provide application isolation and facilitate server consolidation ZFS, the file system that provides a new approach to managing your data with an easy administration interface The Fault Management Architecture, which automates fault detection and resolution The Service

Management Facility, a unified model for services and service management on every Solaris system Dynamic Tracing (DTrace), for troubleshooting OS and application problems on production systems in real timeSecurity has long been a core strength of the Solaris OS and it has been significantly enhanced in the Solaris 10 version in areas such as: Zones virtualization security System hardening Trusted Extensions Privileges and Role-Based Access Control (RBAC) Cryptographic services and key management Auditing Network security Pluggable Authentication Modules (PAM)The strength of Solaris operating system security is its scalability and adaptability. It can protect a single-user system with login authentication to Internet and intranet configurations.This book is the work of the engineers and architects who conceptualized the services, wrote the specifications, and coded the Solaris OS's security features. They bring a wide range of industry and academic experience to the business of creating and deploying secure operating systems. These are the people who know Solaris 10 security best. They have combined to write a book that speaks to readers who want to learn Solaris or who want to use Solaris for the first time in their company's or their own environment. Readers do not have to be experienced Solaris users or operating system developers to take advantage of this book.Books in the Solaris System Administration SeriesSolaris™ 10 System Administration EssentialsSolaris™ 10 System Administration Essentials covers all of the breakthrough features of the Solaris 10 Operating System in one place. It does so in a straightforward way that makes an enterprise-level operating system accessible to system administrators at all levels.Solaris™ 10 System Administration Essentials provides a comprehensive overview along with hands-on examples of the key features that have made Solaris the leading UNIX operating system for years and the significant new features of Solaris 10 that put it far ahead of its competitors. These features include Zones, the ZFS file system, Fault Management Architecture, Service Management Facility, and DTrace, the dynamic tracing tool for troubleshooting OS and application problems on production systems in real time.Solaris™ 10 ZFS EssentialsSolaris™ 10 ZFS Essentials presents the revolutionary Zettabyte File System introduced in Solaris 10. It is a file system that is elegant in its simplicity and the ease with which it allows system administrators to manage data and storage.ZFS is an all-purpose file system that is built on top of a pool of storage devices. File systems that are created from a storage pool share space with the other file systems in the pool. Administrators do not have to allocate storage space based on the intended size of a file system because file systems grow automatically within the space that is allocated to the storage pool. When new storage devices are added, all file systems in the pool can immediately use the additional space.Intended AudienceThe books in the Solaris System Administration Series can benefit anyone who wants to learn more about the Solaris 10 operating system. They are written to be particularly accessible to system administrators who are new to Solaris, and people who are perhaps already serving as administrators in companies running Linux, Windows, and/or other UNIX systems.If you are not presently a practicing system administrator but want to become one, then this series, starting with Solaris™ 10 System Administration Essentials, provides an excellent introduction. In fact, most of the examples used

in the books are suited to or can be adapted to small learning environments like a home setup. Even before you venture into corporate system administration or deploy Solaris 10 in your existing IT installation, these books will help you experiment in a small test environment.OpenSolarisIn June 2005, Sun Microsystems introduced OpenSolaris, a fully functional Solaris operating system release built from open source. While the books in this series focus on Solaris 10, they often incorporate aspects of OpenSolaris. Now that Solaris has been open-sourced, its evolution has accelerated even beyond its normally rapid pace. The authors of this series have often found it interesting to introduce features or nuances that are new in OpenSolaris. At the same, many of the enhancements introduced into OpenSolaris are finding their way into Solaris 10. Whether you are learning Solaris 10 or already have an eye on OpenSolaris, the books in this series are for you.About the AuthorsThis book benefits from the contributions of numerous experts in Solaris security technologies. Below are brief biographies of each of the contributing authors.Glenn Brunette is a Distinguished Engineer, Global Systems Engineering Director, and Chief Security Architect at Sun Microsystems, where he leads a global team focused on information security and assurance. Glenn is the founder of Sun's Systemic Security approach; an OpenSolaris Security Community Leader; the co-founder of the Solaris Security Toolkit software; and a frequent author, contributor, and speaker at both Sun and industry events. Externally, Glenn has served in leadership positions at the National Cyber Security Partnership, the Enterprise Grid Alliance, and the Center for Internet Security.Hai-May Chao is a Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. For the past six years, she has been participating in the design and development of the cryptographic framework and key management framework for the Solaris OS. Hai-May started at Sun in the Trusted Solaris group, developing security features for Trusted Solaris software and the Solaris Management Console. Before joining Sun, Hai-May was a firmware developer at Amdahl, working on IBM-compatible features such as the Multiple Domain Facility (MDF).Martin Englund is a Principal Security Engineer in the Web Engineering organization at Sun Microsystems. He has over sixteen years of experience in data and network security. Web Engineering itself operates the majority of Sun's external Web presence to the Internet and operationally comes under hourly attack. Martin is responsible for the operational security of over 500 Solaris systems that collectively provide the platform for Sun's external Web presence. Prior to this role, Martin worked in Java Security Engineering and the Sun IT Security Office, where he was responsible for creating the standards and tools used by Sun IT to secure its servers. He has also authored numerous technical papers on security, including Sun Blueprints. Before joining Sun, Martin completed studies in Computer Science and Mathematics and worked as a research engineer at the University of Umeå in Sweden.Glenn Faden is a Distinguished Engineer in the Solaris Security Technologies group and has worked at Sun for nineteen years. He is currently the architect for Solaris Trusted Extensions and was one of the architects for the Trusted Solaris OS and role-based access control (RBAC). He designed Sun's multilevel desktops based on Open Look, CDE, and GNOME and holds a patent for the

underlying X11 security policy. Glenn has made extensive contributions to the Solaris security foundation, including access control lists, auditing, device allocation, and OS virtualization. He also developed the RBAC and process rights management tools for the Solaris Management Console. He has authored several articles for Sun's Blueprints Web site and the Solaris Developer Connection. Glenn previously worked for Qubix, OmniCad, and Gould Computer Systems in Desktop Publishing and OS development. He has a Master of Science in Computer Science from the Florida Institute of Technology.Mark Fenwick is a Developer in the Solaris Security Technologies group. His current focus is IPsec and IKE. With over two decades of hardware and UNIX software experience, he has extensive knowledge of firewalls, VPN technology, encryption, and networking protocols. Mark has taught technical classes and presented technical papers to Sun support staff and customers. He closely follows key Sun technologies such as ZFS, Sun Ray, and OpenSolaris, both at work and in his UNIX-only home. In his spare time, he enjoys riding one of his many bicycles and cooking.Valerie Anne Fenwick is a Staff Engineer at Sun Microsystems with over a decade of experience in computer security. She helped to design and develop the cryptographic framework for the Solaris OS. In addition, she chaired the team responsible for code revisions to the Operating System and Networking consolidation for OpenSolaris. Valerie has a Bachelor of Science in Computer Science from Purdue University. In her spare time, she enjoys performing at community theaters, riding her bike, and skiing.Wyllys Ingersoll is a Senior Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. He has been working in the UNIX security field for thirteen years. Since 2000, he has worked on many of the security features in the Solaris OS: Kerberos, the cryptographic framework, and the key management framework, to name a few. Prior to working at Sun, he developed application firewall software for MCI Worldcom and AOL. He received a Master of Science in Computer Science from Virginia Tech in 1996 and a Bachelor of Science in Computer Engineering from Lehigh University in 1988. He lives and works from his home in northern Virginia today.Wolfgang Ley received his diploma in computer science at the German Technical University of Clausthal-Zellerfeld. From 1994 to 1999, he worked at the DFN-CERT (German Computer Emergency Response Team) to build up this new service. During that time, he was also active in the Forum of Incident Response and Security Teams (FIRST). He then joined Sun Microsystems, working as Backend Support Engineer and is dealing with network, security, and kernel internals such as crashdump analysis.Darren Moffat is a Senior Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. Darren is the architect for the Solaris cryptographic framework and Solaris encrypted storage projects. He is also involved in various other OpenSolaris security-related technologies and features. He is an OpenSolaris Security Community leader. Before joining Solaris engineering, Darren worked in SunServices supporting the Trusted Solaris OS, NFS, name services, and Kerberos. Before joining Sun, Darren worked as an analyst/programmer for the UK Ministry of Defence. He is a graduate of the Computing Science Department of Glasgow University.Pravas Kumar Panda received his engineering degree from Biju Patnayak University of Technology in Orissa, India, in

2003 and joined Sun Microsystems soon after that. He has been working on security technologies for the past three years. During this time, he has focused on Kerberos, SSH, PAM, and packet capturing. He has delivered talks on OpenSolaris, secure programing, and security technologies at various universities in India. Pravas enjoys playing the keyboard. He lives in Bangalore with his wife Bobby and their young daughter Avishi.Jan Pechanec works at Sun Microsystems in the Solaris Security Technologies group. At the moment, he works mostly with technologies around the SSH protocol and the PKCS #11 standard. He graduated in 2001 from the Faculty of Mathematics and Physics at Charles University in Prague, where he majored in operating systems and computer networks. For the last few years, he has also been working at Charles University, lecturing on UNIX and UNIX C programming.Mark Phalan works at the Prague office of Sun Microsystems for the Solaris Security Technologies group. He works on Kerberos-related technologies for the Solaris OS and for OpenSolaris. He graduated in 2003 from the Computer Science department of Trinity College Dublin with a BA(Mod) in Computer Science.Scott Rotondo is a Principal Engineer at Sun Microsystems who works as a security architect for the Solaris operating system. He also serves as President of the Trusted Computing Group (TCG), an industry consortium that defines open standards for hardware-enabled trusted computing. In addition to his work on other security features, Scott leads Sun's development effort to support TCG technologies in Solaris. Scott has over twenty years of technical and management experience in UNIX operating system development. For more than a decade, his focus has been on security issues and features in the Solaris operating system.Christoph Schuba studied mathematics and management information systems at the Universität Heidelberg and at the Universität Mannheim in Germany. As a Fulbright scholar, he earned his Master of Science and Doctor of Philosophy degrees in Computer Science from Purdue University in 1993 and 1997, respectively, performing most of his dissertation research in the Computer Science Laboratory at the Xerox Palo Alto Research Center (PARC). Christoph has taught undergraduate and graduate courses in computer and network security, cryptography, operating systems, and distributed systems at San Jose State University, USA; at the Universtität Heidelberg, Germany; at the International University in Bruchsal, Germany; and at Linkopings universitet in Linköping, Sweden, where he held the chair in information security. Since 1997, Christoph has been working at Sun Labs and most recently in the Solaris Software Security Organization at Sun Microsystems, Inc. He holds 13 patents and is author and co-author of numerous scientific articles in computer and network security.Sharon Read Veach is a Technical Writer at Sun. She specializes in documenting inherent security features of the Solaris OS, such as auditing, cryptography, and IPsec. As part of a team of writers, she worked on the Trusted Solaris OS, the predecessor to the Trusted Extensions software in the Solaris 10 release.Darren Reed is a Senior Systems Engineer in the Solaris Networking group. He is the principal engineer for IPfilter in Solaris.Joep Vesseur is a Staff Engineer working for the Solaris Security Technologies group. His main focus areas are PAM, secure programming practices, and OS and network security in general. He has co-authored Solaris 9 and Solaris 10 Security

exams, several Sun Blueprints, and he is a leader of the Dutch OpenSolaris User Group. His background is in digital forensic science and high-performance computing.Paul Wernau is a member of the Solaris Security Technologies group. He started his career in Sun Microsystems doing technical support over the phone for enterprise customers, specializing in networking, firewalls, encrypting gateways, and VPNs. Now an IPsec developer, he tries to remember to focus on observability and meaningful error messages, keeping in mind his experience with debugging problems with system administrators.About the AuthorsThis book benefits from the contributions of numerous experts in Solaris security technologies. Below are brief biographies of each of the contributing authors.Glenn Brunette is a Distinguished Engineer, Global Systems Engineering Director, and Chief Security Architect at Sun Microsystems, where he leads a global team focused on information security and assurance. Glenn is the founder of Sun's Systemic Security approach; an OpenSolaris Security Community Leader; the co-founder of the Solaris Security Toolkit software; and a frequent author, contributor, and speaker at both Sun and industry events. Externally, Glenn has served in leadership positions at the National Cyber Security Partnership, the Enterprise Grid Alliance, and the Center for Internet Security.Hai-May Chao is a Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. For the past six years, she has been participating in the design and development of the cryptographic framework and key management framework for the Solaris OS. Hai-May started at Sun in the Trusted Solaris group, developing security features for Trusted Solaris software and the Solaris Management Console. Before joining Sun, Hai-May was a firmware developer at Amdahl, working on IBM-compatible features such as the Multiple Domain Facility (MDF).Martin Englund is a Principal Security Engineer in the Web Engineering organization at Sun Microsystems. He has over sixteen years of experience in data and network security. Web Engineering itself operates the majority of Sun's external Web presence to the Internet and operationally comes under hourly attack. Martin is responsible for the operational security of over 500 Solaris systems that collectively provide the platform for Sun's external Web presence. Prior to this role, Martin worked in Java Security Engineering and the Sun IT Security Office, where he was responsible for creating the standards and tools used by Sun IT to secure its servers. He has also authored numerous technical papers on security, including Sun Blueprints. Before joining Sun, Martin completed studies in Computer Science and Mathematics and worked as a research engineer at the University of Umeå in Sweden.Glenn Faden is a Distinguished Engineer in the Solaris Security Technologies group and has worked at Sun for nineteen years. He is currently the architect for Solaris Trusted Extensions and was one of the architects for the Trusted Solaris OS and role-based access control (RBAC). He designed Sun's multilevel desktops based on Open Look, CDE, and GNOME and holds a patent for the underlying X11 security policy. Glenn has made extensive contributions to the Solaris security foundation, including access control lists, auditing, device allocation, and OS virtualization. He also developed the RBAC and process rights management tools for the Solaris Management Console. He has authored several articles for Sun's Blueprints Web site and the Solaris

Developer Connection. Glenn previously worked for Qubix, OmniCad, and Gould Computer Systems in Desktop Publishing and OS development. He has a Master of Science in Computer Science from the Florida Institute of Technology.Mark Fenwick is a Developer in the Solaris Security Technologies group. His current focus is IPsec and IKE. With over two decades of hardware and UNIX software experience, he has extensive knowledge of firewalls, VPN technology, encryption, and networking protocols. Mark has taught technical classes and presented technical papers to Sun support staff and customers. He closely follows key Sun technologies such as ZFS, Sun Ray, and OpenSolaris, both at work and in his UNIX-only home. In his spare time, he enjoys riding one of his many bicycles and cooking.Valerie Anne Fenwick is a Staff Engineer at Sun Microsystems with over a decade of experience in computer security. She helped to design and develop the cryptographic framework for the Solaris OS. In addition, she chaired the team responsible for code revisions to the Operating System and Networking consolidation for OpenSolaris. Valerie has a Bachelor of Science in Computer Science from Purdue University. In her spare time, she enjoys performing at community theaters, riding her bike, and skiing.Wyllys Ingersoll is a Senior Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. He has been working in the UNIX security field for thirteen years. Since 2000, he has worked on many of the security features in the Solaris OS: Kerberos, the cryptographic framework, and the key management framework, to name a few. Prior to working at Sun, he developed application firewall software for MCI Worldcom and AOL. He received a Master of Science in Computer Science from Virginia Tech in 1996 and a Bachelor of Science in Computer Engineering from Lehigh University in 1988. He lives and works from his home in northern Virginia today.Wolfgang Ley received his diploma in computer science at the German Technical University of Clausthal-Zellerfeld. From 1994 to 1999, he worked at the DFN-CERT (German Computer Emergency Response Team) to build up this new service. During that time, he was also active in the Forum of Incident Response and Security Teams (FIRST). He then joined Sun Microsystems, working as Backend Support Engineer and is dealing with network, security, and kernel internals such as crashdump analysis.Darren Moffat is a Senior Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. Darren is the architect for the Solaris cryptographic framework and Solaris encrypted storage projects. He is also involved in various other OpenSolaris security-related technologies and features. He is an OpenSolaris Security Community leader. Before joining Solaris engineering, Darren worked in SunServices supporting the Trusted Solaris OS, NFS, name services, and Kerberos. Before joining Sun, Darren worked as an analyst/programmer for the UK Ministry of Defence. He is a graduate of the Computing Science Department of Glasgow University.Pravas Kumar Panda received his engineering degree from Biju Patnayak University of Technology in Orissa, India, in 2003 and joined Sun Microsystems soon after that. He has been working on security technologies for the past three years. During this time, he has focused on Kerberos, SSH, PAM, and packet capturing. He has delivered talks on OpenSolaris, secure programing, and security technologies at various universities in India. Pravas enjoys playing the keyboard. He lives in

Bangalore with his wife Bobby and their young daughter Avishi.Jan Pechanec works at Sun Microsystems in the Solaris Security Technologies group. At the moment, he works mostly with technologies around the SSH protocol and the PKCS #11 standard. He graduated in 2001 from the Faculty of Mathematics and Physics at Charles University in Prague, where he majored in operating systems and computer networks. For the last few years, he has also been working at Charles University, lecturing on UNIX and UNIX C programming.Mark Phalan works at the Prague office of Sun Microsystems for the Solaris Security Technologies group. He works on Kerberos-related technologies for the Solaris OS and for OpenSolaris. He graduated in 2003 from the Computer Science department of Trinity College Dublin with a BA(Mod) in Computer Science.Scott Rotondo is a Principal Engineer at Sun Microsystems who works as a security architect for the Solaris operating system. He also serves as President of the Trusted Computing Group (TCG), an industry consortium that defines open standards for hardware-enabled trusted computing. In addition to his work on other security features, Scott leads Sun's development effort to support TCG technologies in Solaris. Scott has over twenty years of technical and management experience in UNIX operating system development. For more than a decade, his focus has been on security issues and features in the Solaris operating system.Christoph Schuba studied mathematics and management information systems at the Universität Heidelberg and at the Universität Mannheim in Germany. As a Fulbright scholar, he earned his Master of Science and Doctor of Philosophy degrees in Computer Science from Purdue University in 1993 and 1997, respectively, performing most of his dissertation research in the Computer Science Laboratory at the Xerox Palo Alto Research Center (PARC). Christoph has taught undergraduate and graduate courses in computer and network security, cryptography, operating systems, and distributed systems at San Jose State University, USA; at the Universtität Heidelberg, Germany; at the International University in Bruchsal, Germany; and at Linkopings universitet in Linköping, Sweden, where he held the chair in information security. Since 1997, Christoph has been working at Sun Labs and most recently in the Solaris Software Security Organization at Sun Microsystems, Inc. He holds 13 patents and is author and co-author of numerous scientific articles in computer and network security.Sharon Read Veach is a Technical Writer at Sun. She specializes in documenting inherent security features of the Solaris OS, such as auditing, cryptography, and IPsec. As part of a team of writers, she worked on the Trusted Solaris OS, the predecessor to the Trusted Extensions software in the Solaris 10 release.Darren Reed is a Senior Systems Engineer in the Solaris Networking group. He is the principal engineer for IPfilter in Solaris.Joep Vesseur is a Staff Engineer working for the Solaris Security Technologies group. His main focus areas are PAM, secure programming practices, and OS and network security in general. He has co-authored Solaris 9 and Solaris 10 Security exams, several Sun Blueprints, and he is a leader of the Dutch OpenSolaris User Group. His background is in digital forensic science and high-performance computing.Paul Wernau is a member of the Solaris Security Technologies group. He started his career in Sun Microsystems doing technical support over the phone for enterprise customers, specializing in networking,

firewalls, encrypting gateways, and VPNs. Now an IPsec developer, he tries to remember to focus on observability and meaningful error messages, keeping in mind his experience with debugging problems with system administrators.About the AuthorsThis book benefits from the contributions of numerous experts in Solaris security technologies. Below are brief biographies of each of the contributing authors.Glenn Brunette is a Distinguished Engineer, Global Systems Engineering Director, and Chief Security Architect at Sun Microsystems, where he leads a global team focused on information security and assurance. Glenn is the founder of Sun's Systemic Security approach; an OpenSolaris Security Community Leader; the co-founder of the Solaris Security Toolkit software; and a frequent author, contributor, and speaker at both Sun and industry events. Externally, Glenn has served in leadership positions at the National Cyber Security Partnership, the Enterprise Grid Alliance, and the Center for Internet Security.Hai-May Chao is a Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. For the past six years, she has been participating in the design and development of the cryptographic framework and key management framework for the Solaris OS. Hai-May started at Sun in the Trusted Solaris group, developing security features for Trusted Solaris software and the Solaris Management Console. Before joining Sun, Hai-May was a firmware developer at Amdahl, working on IBM-compatible features such as the Multiple Domain Facility (MDF).Martin Englund is a Principal Security Engineer in the Web Engineering organization at Sun Microsystems. He has over sixteen years of experience in data and network security. Web Engineering itself operates the majority of Sun's external Web presence to the Internet and operationally comes under hourly attack. Martin is responsible for the operational security of over 500 Solaris systems that collectively provide the platform for Sun's external Web presence. Prior to this role, Martin worked in Java Security Engineering and the Sun IT Security Office, where he was responsible for creating the standards and tools used by Sun IT to secure its servers. He has also authored numerous technical papers on security, including Sun Blueprints. Before joining Sun, Martin completed studies in Computer Science and Mathematics and worked as a research engineer at the University of Umeå in Sweden.Glenn Faden is a Distinguished Engineer in the Solaris Security Technologies group and has worked at Sun for nineteen years. He is currently the architect for Solaris Trusted Extensions and was one of the architects for the Trusted Solaris OS and role-based access control (RBAC). He designed Sun's multilevel desktops based on Open Look, CDE, and GNOME and holds a patent for the underlying X11 security policy. Glenn has made extensive contributions to the Solaris security foundation, including access control lists, auditing, device allocation, and OS virtualization. He also developed the RBAC and process rights management tools for the Solaris Management Console. He has authored several articles for Sun's Blueprints Web site and the Solaris Developer Connection. Glenn previously worked for Qubix, OmniCad, and Gould Computer Systems in Desktop Publishing and OS development. He has a Master of Science in Computer Science from the Florida Institute of Technology.Mark Fenwick is a Developer in the Solaris Security Technologies group. His current focus is IPsec and IKE. With over two decades of

hardware and UNIX software experience, he has extensive knowledge of firewalls, VPN technology, encryption, and networking protocols. Mark has taught technical classes and presented technical papers to Sun support staff and customers. He closely follows key Sun technologies such as ZFS, Sun Ray, and OpenSolaris, both at work and in his UNIX-only home. In his spare time, he enjoys riding one of his many bicycles and cooking.Valerie Anne Fenwick is a Staff Engineer at Sun Microsystems with over a decade of experience in computer security. She helped to design and develop the cryptographic framework for the Solaris OS. In addition, she chaired the team responsible for code revisions to the Operating System and Networking consolidation for OpenSolaris. Valerie has a Bachelor of Science in Computer Science from Purdue University. In her spare time, she enjoys performing at community theaters, riding her bike, and skiing.Wyllys Ingersoll is a Senior Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. He has been working in the UNIX security field for thirteen years. Since 2000, he has worked on many of the security features in the Solaris OS: Kerberos, the cryptographic framework, and the key management framework, to name a few. Prior to working at Sun, he developed application firewall software for MCI Worldcom and AOL. He received a Master of Science in Computer Science from Virginia Tech in 1996 and a Bachelor of Science in Computer Engineering from Lehigh University in 1988. He lives and works from his home in northern Virginia today.Wolfgang Ley received his diploma in computer science at the German Technical University of Clausthal-Zellerfeld. From 1994 to 1999, he worked at the DFN-CERT (German Computer Emergency Response Team) to build up this new service. During that time, he was also active in the Forum of Incident Response and Security Teams (FIRST). He then joined Sun Microsystems, working as Backend Support Engineer and is dealing with network, security, and kernel internals such as crashdump analysis.Darren Moffat is a Senior Staff Engineer at Sun Microsystems in the Solaris Security Technologies group. Darren is the architect for the Solaris cryptographic framework and Solaris encrypted storage projects. He is also involved in various other OpenSolaris security-related technologies and features. He is an OpenSolaris Security Community leader. Before joining Solaris engineering, Darren worked in SunServices supporting the Trusted Solaris OS, NFS, name services, and Kerberos. Before joining Sun, Darren worked as an analyst/programmer for the UK Ministry of Defence. He is a graduate of the Computing Science Department of Glasgow University.Pravas Kumar Panda received his engineering degree from Biju Patnayak University of Technology in Orissa, India, in 2003 and joined Sun Microsystems soon after that. He has been working on security technologies for the past three years. During this time, he has focused on Kerberos, SSH, PAM, and packet capturing. He has delivered talks on OpenSolaris, secure programing, and security technologies at various universities in India. Pravas enjoys playing the keyboard. He lives in Bangalore with his wife Bobby and their young daughter Avishi.Jan Pechanec works at Sun Microsystems in the Solaris Security Technologies group. At the moment, he works mostly with technologies around the SSH protocol and the PKCS #11 standard. He graduated in 2001 from the Faculty of Mathematics and Physics at Charles University in Prague, where he majored in

operating systems and computer networks. For the last few years, he has also been working at Charles University, lecturing on UNIX and UNIX C programming.Mark Phalan works at the Prague office of Sun Microsystems for the Solaris Security Technologies group. He works on Kerberos-related technologies for the Solaris OS and for OpenSolaris. He graduated in 2003 from the Computer Science department of Trinity College Dublin with a BA(Mod) in Computer Science.Scott Rotondo is a Principal Engineer at Sun Microsystems who works as a security architect for the Solaris operating system. He also serves as President of the Trusted Computing Group (TCG), an industry consortium that defines open standards for hardware-enabled trusted computing. In addition to his work on other security features, Scott leads Sun's development effort to support TCG technologies in Solaris. Scott has over twenty years of technical and management experience in UNIX operating system development. For more than a decade, his focus has been on security issues and features in the Solaris operating system.Christoph Schuba studied mathematics and management information systems at the Universität Heidelberg and at the Universität Mannheim in Germany. As a Fulbright scholar, he earned his Master of Science and Doctor of Philosophy degrees in Computer Science from Purdue University in 1993 and 1997, respectively, performing most of his dissertation research in the Computer Science Laboratory at the Xerox Palo Alto Research Center (PARC). Christoph has taught undergraduate and graduate courses in computer and network security, cryptography, operating systems, and distributed systems at San Jose State University, USA; at the Universtität Heidelberg, Germany; at the International University in Bruchsal, Germany; and at Linkopings universitet in Linköping, Sweden, where he held the chair in information security. Since 1997, Christoph has been working at Sun Labs and most recently in the Solaris Software Security Organization at Sun Microsystems, Inc. He holds 13 patents and is author and co-author of numerous scientific articles in computer and network security.Sharon Read Veach is a Technical Writer at Sun. She specializes in documenting inherent security features of the Solaris OS, such as auditing, cryptography, and IPsec. As part of a team of writers, she worked on the Trusted Solaris OS, the predecessor to the Trusted Extensions software in the Solaris 10 release.Darren Reed is a Senior Systems Engineer in the Solaris Networking group. He is the principal engineer for IPfilter in Solaris.Joep Vesseur is a Staff Engineer working for the Solaris Security Technologies group. His main focus areas are PAM, secure programming practices, and OS and network security in general. He has co-authored Solaris 9 and Solaris 10 Security exams, several Sun Blueprints, and he is a leader of the Dutch OpenSolaris User Group. His background is in digital forensic science and high-performance computing.Paul Wernau is a member of the Solaris Security Technologies group. He started his career in Sun Microsystems doing technical support over the phone for enterprise customers, specializing in networking, firewalls, encrypting gateways, and VPNs. Now an IPsec developer, he tries to remember to focus on observability and meaningful error messages, keeping in mind his experience with debugging problems with system administrators.1Solaris Security ServicesThe Solaris Operating System (Solaris OS) offers a variety of security services, from single-user login

authentication to network protections. In the Solaris 10 release, Sun expands its Solaris OS security offerings to provide containment, encryption, PKCS #11 functionality, Trusted Extensions (labeling), and other advanced security features. Many features are based on open industry standards, such as Kerberos and PKCS #11.This book is written by the engineers and architects who conceptualized the services, wrote the specifications, and coded the security software. This book is intended for users who are interested in maintaining security on a single system and for users who administer a network of systems.1.1 A Solaris Security StoryIn 1988, the Morris worm attacked UNIX systems on the Internet, replicated itself, and brought the network down. This was the first Internet worm.1 Many of the network systems were Sun systems. The Morris worm gained remote access by trying common passwords, using the debug mode of sendmail, and exploiting a buffer overflow in the fingerd program. Countermeasures were complex, because UNIX systems were constructed for openness and connectivity, not security. Today, security is part of Sun's DNA. The kernel and installation are designed to protect Solaris systems from attack.Fast forward to 2007, after Sun distributed Solaris OS code to external developers and users under an OpenSolaris license. An early scrutinizer of the code reported a previously unknown vulnerability in Solaris' telnet program. Because the code was open sourced, the report could and did include a pointer to the lines in the code that were responsible. The report was posted on a Sunday afternoon. By Monday, Sun engineers and testers in Australia, the United Kingdom, and the United States determined how the telnet -f -l root command could compromise a system, posted a Sun Alert, and provided a temporary fix. By Tuesday, Sun issued a patch.2Sun's countermeasures were not complex. Since 1988, Sun has implemented many security features that limit the damage that an attack can inflict. A Secure By Default (SBD) installation closes the telnet port. If the port is open, a simple svcadm command disables the telnet service. Containers, called zones, can limit the effect of an attack, so the system can be reached and fixed by an administrator with the proper privileges.So, unlike the Morris worm, the telnet worm required a small patch for the specific bug. Solaris security features were already in place. The patch was required on all systems that were running the Solaris 10 version of the OS. Solaris administrators were urged to install the patch, and until they could patch the system, they were urged to disable the telnet program.The danger of a telnet worm attack appeared to be averted due to a solid Solaris kernel, external eyes that pinpointed the offending code, a global response by Sun engineers, and the creation, testing, and distribution of a patch by Sun services. That was the engineering part of the fix; then came the applying part of the fix. The patch needed to be applied. Humans, or scripts that were run by humans, needed to patch all affected systems.Two weeks after the patch was released, hackers launched a telnet worm. The worm infected thousands of unpatched systems on the network of the most extensive installation of Solaris machines—SWAN, the Sun-wide area network of Sun Microsystems.Sun engineers install and re-install their machines to test code. For the most part, these systems are administered by the individual engineer. When the worm invaded these systems, network traffic increased noticeably. Administrative disruption was minimal. The worm

could not gain root access, so the SWAN was accessible and systems could be repaired remotely.Within six hours of detecting the worm at Sun, a process was implemented to help Sun engineers both report and repair infected systems. Within 24 hours of the initial outbreak, over 7000 systems had been inoculated by Sun IT and lab network volunteers, and the worm propagation had been controlled on SWAN.The process needed to handle systems that were not managed by Sun IT, such as systems in lab environments, individually managed systems, and systems that had been offline when the repairs were effected. Such systems could get infected upon going online and then spread the worm from within SWAN. Similarly, systems that were subsequently reinstalled with existing JumpStart scripts would re-infect Sun from within SWAN. The inoculation script handled this chaotic environment by locating every newly booted system and configuring it so that the worm could not re-infect. For systems that allowed root login, it disabled the telnet program and because root compromise was possible, advised the administrator in LARGE TYPE to patch the system.So, the Solaris security story is a series of short stories. Limit the attack surface—use an OS with built-in security. Find vulnerabilities quickly—provide the code as open source for more eyes on the code. Create a patch—the underlying security enables quick creation. End a vulnerability—apply the patch. Prevent the attack from being re-introduced—provide comprehensive network management, which might require scripts or firewalls between LANs at a site.1.2 Security Services in the Solaris OSSolaris OS security features protect network packets from being snooped on the wire, protect the system from rogue programs, and limit the damage from denial of service (DoS) attacks. Many features are in place at installation. Some features must be configured and then activated. Other features can be strengthened to meet more stringent requirements. This security strategy enables you to configure your system to have the smallest footprint with the largest security benefits. You decide.Table 1.1 describes the default security configuration and points to the chapters that cover the feature or its use in concert with other features.Table 1.1 Security Features, Default Configurations, and Chapters in Which They Are Covered1.3 Configurable Security Services in the Solaris OSBy editing configuration files and enabling services, you can customize the security profile of the system to match your site's requirements. Some features, such as ACLs and KMF, are easily configured on the command line. Table 1.2 lists the features that can be customized, the reasons why you might customize them, and the chapters where the features are covered.Table 1.2 Configurable Security ServicesNotes1. For more information, see Spafford, Eugene. The Internet Worm Program: an analysis. Purdue Technical Report CDS-TR-823. 1988.2. For a fuller account, see the Sun blogs, http://blogs.sun.com/drscholl/ and http://blogs.sun.com/tpenta/entry/the_in_telnetd_vulnerability_exploit.1Solaris Security ServicesThe Solaris Operating System (Solaris OS) offers a variety of security services, from single-user login authentication to network protections. In the Solaris 10 release, Sun expands its Solaris OS security offerings to provide containment, encryption, PKCS #11 functionality, Trusted Extensions (labeling), and other advanced security features. Many features are based on open industry standards, such as Kerberos and PKCS #11.This book is written by the engineers and

architects who conceptualized the services, wrote the specifications, and coded the security software. This book is intended for users who are interested in maintaining security on a single system and for users who administer a network of systems.1.1 A Solaris Security StoryIn 1988, the Morris worm attacked UNIX systems on the Internet, replicated itself, and brought the network down. This was the first Internet worm.1 Many of the network systems were Sun systems. The Morris worm gained remote access by trying common passwords, using the debug mode of sendmail, and exploiting a buffer overflow in the fingerd program. Countermeasures were complex, because UNIX systems were constructed for openness and connectivity, not security. Today, security is part of Sun's DNA. The kernel and installation are designed to protect Solaris systems from attack.Fast forward to 2007, after Sun distributed Solaris OS code to external developers and users under an OpenSolaris license. An early scrutinizer of the code reported a previously unknown vulnerability in Solaris' telnet program. Because the code was open sourced, the report could and did include a pointer to the lines in the code that were responsible. The report was posted on a Sunday afternoon. By Monday, Sun engineers and testers in Australia, the United Kingdom, and the United States determined how the telnet -f -l root command could compromise a system, posted a Sun Alert, and provided a temporary fix. By Tuesday, Sun issued a patch.2Sun's countermeasures were not complex. Since 1988, Sun has implemented many security features that limit the damage that an attack can inflict. A Secure By Default (SBD) installation closes the telnet port. If the port is open, a simple svcadm command disables the telnet service. Containers, called zones, can limit the effect of an attack, so the system can be reached and fixed by an administrator with the proper privileges.So, unlike the Morris worm, the telnet worm required a small patch for the specific bug. Solaris security features were already in place. The patch was required on all systems that were running the Solaris 10 version of the OS. Solaris administrators were urged to install the patch, and until they could patch the system, they were urged to disable the telnet program.The danger of a telnet worm attack appeared to be averted due to a solid Solaris kernel, external eyes that pinpointed the offending code, a global response by Sun engineers, and the creation, testing, and distribution of a patch by Sun services. That was the engineering part of the fix; then came the applying part of the fix. The patch needed to be applied. Humans, or scripts that were run by humans, needed to patch all affected systems.Two weeks after the patch was released, hackers launched a telnet worm. The worm infected thousands of unpatched systems on the network of the most extensive installation of Solaris machines—SWAN, the Sun-wide area network of Sun Microsystems.Sun engineers install and re-install their machines to test code. For the most part, these systems are administered by the individual engineer. When the worm invaded these systems, network traffic increased noticeably. Administrative disruption was minimal. The worm could not gain root access, so the SWAN was accessible and systems could be repaired remotely.Within six hours of detecting the worm at Sun, a process was implemented to help Sun engineers both report and repair infected systems. Within 24 hours of the initial outbreak, over 7000 systems had been inoculated by Sun IT and lab network volunteers, and the worm

propagation had been controlled on SWAN.The process needed to handle systems that were not managed by Sun IT, such as systems in lab environments, individually managed systems, and systems that had been offline when the repairs were effected. Such systems could get infected upon going online and then spread the worm from within SWAN. Similarly, systems that were subsequently reinstalled with existing JumpStart scripts would re-infect Sun from within SWAN. The inoculation script handled this chaotic environment by locating every newly booted system and configuring it so that the worm could not re-infect. For systems that allowed root login, it disabled the telnet program and because root compromise was possible, advised the administrator in LARGE TYPE to patch the system.So, the Solaris security story is a series of short stories. Limit the attack surface—use an OS with built-in security. Find vulnerabilities quickly—provide the code as open source for more eyes on the code. Create a patch—the underlying security enables quick creation. End a vulnerability—apply the patch. Prevent the attack from being re-introduced—provide comprehensive network management, which might require scripts or firewalls between LANs at a site.1.2 Security Services in the Solaris OSSolaris OS security features protect network packets from being snooped on the wire, protect the system from rogue programs, and limit the damage from denial of service (DoS) attacks. Many features are in place at installation. Some features must be configured and then activated. Other features can be strengthened to meet more stringent requirements. This security strategy enables you to configure your system to have the smallest footprint with the largest security benefits. You decide.Table 1.1 describes the default security configuration and points to the chapters that cover the feature or its use in concert with other features.Table 1.1 Security Features, Default Configurations, and Chapters in Which They Are Covered1.3 Configurable Security Services in the Solaris OSBy editing configuration files and enabling services, you can customize the security profile of the system to match your site's requirements. Some features, such as ACLs and KMF, are easily configured on the command line. Table 1.2 lists the features that can be customized, the reasons why you might customize them, and the chapters where the features are covered.Table 1.2 Configurable Security ServicesNotes1. For more information, see Spafford, Eugene. The Internet Worm Program: an analysis. Purdue Technical Report CDS-TR-823. 1988.2. For a fuller account, see the Sun blogs, http://blogs.sun.com/drscholl/ and http://blogs.sun.com/tpenta/entry/the_in_telnetd_vulnerability_exploit.1Solaris Security ServicesThe Solaris Operating System (Solaris OS) offers a variety of security services, from single-user login authentication to network protections. In the Solaris 10 release, Sun expands its Solaris OS security offerings to provide containment, encryption, PKCS #11 functionality, Trusted Extensions (labeling), and other advanced security features. Many features are based on open industry standards, such as Kerberos and PKCS #11.This book is written by the engineers and architects who conceptualized the services, wrote the specifications, and coded the security software. This book is intended for users who are interested in maintaining security on a single system and for users who administer a network of systems.1.1 A Solaris Security StoryIn 1988, the Morris worm attacked UNIX systems on the Internet, replicated itself, and brought the

network down. This was the first Internet worm.1 Many of the network systems were Sun systems. The Morris worm gained remote access by trying common passwords, using the debug mode of sendmail, and exploiting a buffer overflow in the fingerd program. Countermeasures were complex, because UNIX systems were constructed for openness and connectivity, not security. Today, security is part of Sun's DNA. The kernel and installation are designed to protect Solaris systems from attack.Fast forward to 2007, after Sun distributed Solaris OS code to external developers and users under an OpenSolaris license. An early scrutinizer of the code reported a previously unknown vulnerability in Solaris' telnet program. Because the code was open sourced, the report could and did include a pointer to the lines in the code that were responsible. The report was posted on a Sunday afternoon. By Monday, Sun engineers and testers in Australia, the United Kingdom, and the United States determined how the telnet -f -l root command could compromise a system, posted a Sun Alert, and provided a temporary fix. By Tuesday, Sun issued a patch.2Sun's countermeasures were not complex. Since 1988, Sun has implemented many security features that limit the damage that an attack can inflict. A Secure By Default (SBD) installation closes the telnet port. If the port is open, a simple svcadm command disables the telnet service. Containers, called zones, can limit the effect of an attack, so the system can be reached and fixed by an administrator with the proper privileges.So, unlike the Morris worm, the telnet worm required a small patch for the specific bug. Solaris security features were already in place. The patch was required on all systems that were running the Solaris 10 version of the OS. Solaris administrators were urged to install the patch, and until they could patch the system, they were urged to disable the telnet program.The danger of a telnet worm attack appeared to be averted due to a solid Solaris kernel, external eyes that pinpointed the offending code, a global response by Sun engineers, and the creation, testing, and distribution of a patch by Sun services. That was the engineering part of the fix; then came the applying part of the fix. The patch needed to be applied. Humans, or scripts that were run by humans, needed to patch all affected systems.Two weeks after the patch was released, hackers launched a telnet worm. The worm infected thousands of unpatched systems on the network of the most extensive installation of Solaris machines—SWAN, the Sun-wide area network of Sun Microsystems.Sun engineers install and re-install their machines to test code. For the most part, these systems are administered by the individual engineer. When the worm invaded these systems, network traffic increased noticeably. Administrative disruption was minimal. The worm could not gain root access, so the SWAN was accessible and systems could be repaired remotely.Within six hours of detecting the worm at Sun, a process was implemented to help Sun engineers both report and repair infected systems. Within 24 hours of the initial outbreak, over 7000 systems had been inoculated by Sun IT and lab network volunteers, and the worm propagation had been controlled on SWAN.The process needed to handle systems that were not managed by Sun IT, such as systems in lab environments, individually managed systems, and systems that had been offline when the repairs were effected. Such systems could get infected upon going online and then spread the worm from within SWAN. Similarly, systems that

were subsequently reinstalled with existing JumpStart scripts would re-infect Sun from within SWAN. The inoculation script handled this chaotic environment by locating every newly booted system and configuring it so that the worm could not re-infect. For systems that allowed root login, it disabled the telnet program and because root compromise was possible, advised the administrator in LARGE TYPE to patch the system.So, the Solaris security story is a series of short stories. Limit the attack surface—use an OS with built-in security. Find vulnerabilities quickly—provide the code as open source for more eyes on the code. Create a patch—the underlying security enables quick creation. End a vulnerability—apply the patch. Prevent the attack from being re-introduced—provide comprehensive network management, which might require scripts or firewalls between LANs at a site.1.2 Security Services in the Solaris OSSolaris OS security features protect network packets from being snooped on the wire, protect the system from rogue programs, and limit the damage from denial of service (DoS) attacks. Many features are in place at installation. Some features must be configured and then activated. Other features can be strengthened to meet more stringent requirements. This security strategy enables you to configure your system to have the smallest footprint with the largest security benefits. You decide.Table 1.1 describes the default security configuration and points to the chapters that cover the feature or its use in concert with other features.Table 1.1 Security Features, Default Configurations, and Chapters in Which They Are Covered1.3 Configurable Security Services in the Solaris OSBy editing configuration files and enabling services, you can customize the security profile of the system to match your site's requirements. Some features, such as ACLs and KMF, are easily configured on the command line. Table 1.2 lists the features that can be customized, the reasons why you might customize them, and the chapters where the features are covered.Table 1.2 Configurable Security ServicesNotes1. For more information, see Spafford, Eugene. The Internet Worm Program: an analysis. Purdue Technical Report CDS-TR-823. 1988.2. For a fuller account, see the Sun blogs, http://blogs.sun.com/drscholl/ and http://blogs.sun.com/tpenta/entry/the_in_telnetd_vulnerability_exploit.2Hardening Solaris SystemsSolaris systems are highly configurable. Even before installing additional software, the administrator has the opportunity to make hundreds of choices about how to set up the operating system itself. These choices include which services to enable, the specific configuration settings of each service, and additional settings that apply to the system as a whole.There is no single "right answer" to any of these configuration choices. If there were, that choice could be set permanently in the operating system, and there would be no need for it to be configurable. Indeed, the process of hardening a computer system is one of making choices and trade-offs that maximize security, while still supporting the intended uses of the system. Beyond the basic techniques presented here, the publications listed at the end of this chapter provide a great deal of specific information to guide the administrator in selecting configuration settings to harden a system.Once the necessary configuration choices have been made and implemented, the system requires regular monitoring to ensure that the system remains in the hardened state. The latter half of this chapter describes tools in the Solaris OS and elsewhere that help to

simplify this task.2.1 Securing Network ServicesTraditionally, many UNIX-based systems have made a wide variety of services available to remote clients over the network. By default, Solaris 10 OS installation enables a large number of network services. This open approach is convenient because it makes services available with minimal setup, and it may provide adequate security in environments where access to the network is limited to a known and trusted set of users. However, many Solaris systems are attached to the Internet or to other networks that cannot be fully trusted. In such an environment, it is prudent to limit the set of services provided to remote clients.Of course, a service that listens to the network for requests and provides responses is not necessarily insecure. Such services are typically designed to authenticate the identity of the client in some way before performing a requested operation or divulging any non-public information. However, sometimes even the “public” information that is provided without authentication can be useful to an attacker. More importantly, the programs that implement these services, like all programs, may contain bugs, and accepting input from the network makes it easier for an attacker to exploit remotely and anonymously any vulnerabilities that may exist.To reduce this risk to an acceptable level, the guiding principle is to minimize the set of opportunities available to an attacker, sometimes called the “attack surface,” by providing only the services that are necessary for the purpose of the system. Furthermore, it is important to disable unnecessary services from the moment the system is connected to the network. If a vulnerable service is enabled briefly after installation or each time the system is booted until the service is disabled, then a window of opportunity exists for an attacker to exploit the vulnerability until the service is disabled.2.1.1 Minimizing Installed SoftwareOne approach to limit the network services that are provided by a system is to avoid installing the software that provides unneeded services. All of the programs that are delivered with a Solaris release are divided into packages, with related programs grouped together in the same package. Of course, if a package is not installed, the programs it contains cannot run, and therefore they cannot provide any services that are accessible from the network.Beginning in the Solaris 9 release, the common network services provided by inetd(1M) were divided among several new packages; for example, the packages related to the telnet program are shown below. Discrete packaging allows the administrator to limit the network services provided by customizing the installation to leave out some packages. Individual packages can also be removed after installation using pkgrm(1M).# pkginfo SUNWtnetc SUNWtnetd SUNWtnetrsystem SUNWtnetc Telnet Command (client)system SUNWtnetd Telnet Server Daemon (Usr)system SUNWtnetr Telnet Server Daemon (Root)The Solaris 10 release expanded upon this idea by adding a Reduced Networking selection at install time to complement the existing Core, End User, Developer, and Entire Distribution choices. Each of these options is an install metacluster, which defines the set of packages to be installed. The Reduced Networking metacluster eliminates as many networking services as possible while leaving the system operational enough to allow further configuration. Of course, this configuration must be done by an administrator at the system console, because there are no services running to allow for remote

access and administration.In most cases, a system installed with the Reduced Networking metacluster is too limited to be useful without adding additional packages. Completely eliminating network services has a severe impact on local users as well; for example, a Reduced Networking system does not allow local users to log in via the window system. Rather than using the installed system as-is, the Reduced Networking metacluster is intended to serve as a foundation to which the administrator can add only those packages that are desired for the particular system.While removing packages is an effective way to eliminate network services, this approach has several practical drawbacks. The first is that the level of granularity may be insufficient. Although services are divided among more packages than in the past, it is still common to have several related services provided by a single package, and choosing to install a package means installing all of the services it provides. Second, it is inconvenient to add services later to adapt to changing requirements, since this requires access to the original installation media in order to add the relevant packages. Finally, packages that are added some time after installation will not have the benefit of any bug fixes contained in patches that may have been applied to the system after installation.For details about Sun's support for minimized systems, refer to http://www.opensolaris.org/os/community/security/files/minimization-support-rules-ext.pdf.2.1.2 Using SMF to Control Network ServicesThe Solaris Service Management Facility(SMF), introduced in the Solaris 10 OS, provides a general mechanism for the control and configuration of system services. The fundamental unit of administration in SMF is the service instance. An instance is a specific configuration of a service. For example, a Web server is a service. A single Web server daemon that is configured to accept Web service requests on port 80 is an instance of that service. Multiple instances of the same service can run on a single Solaris OS. Each service instance is given a Fault Management Resource Identifier (FMRI). The FMRI includes the service name and the instance name. When you list a service using the svcs(1) command, the listing includes the status, date, service name, and service instance. In the next case, "network/dhcp-server" identifies the service, and "default" identifies the instance.# svcs -a | grep dhcpdisabled Apr 10 svc:/network/dhcp-server:defaultBecause SMF separates configuration of a service from its installation, it is possible to install the entire Solaris OS but enable only the services that are needed on the particular system. To simplify the management process, SMF bundles together the configuration settings for many services into a profile. Solaris includes two generic predefined profiles in /var/svc/profile called generic_open.xml and generic_limited_net.xml. The open profile provides the traditional Solaris configuration with many network services enabled. The limited_net profile provides no services to remote clients except for the Secure Shell daemon sshd(1M).An exception is made for sshd because it is frequently necessary to have some mechanism that allows the administrator to log in and administer the system remotely. Since the need to administer the system remotely may occur at unpredictable times, it is impractical to enable sshd only when it is needed. For this reason, sshd is left enabled by default. Secure Shell is a safer choice for remote administration than telnet or rlogin because it protects the privacy and integrity of the communication session. An

administrator who does not want to leave sshd enabled can disable this service individually using the svcadm(1M) command as follows:# svcadm disable sshThe goal of the limited_net profile is to avoid providing network services to remote clients without affecting the normal activities of local users. The services affected by the limited_net profile are shown in Table 2.1.Table 2.1 Services Affected by the limited_net ProfileFor services that are normally used only by remote clients, it is straightforward for the profile to simply disable those services. Other services, including several that are associated with the window system, are also needed by local users. For these services, SMF properties are used to configure the services to respond only to requests originating from the local system. Two examples of properties that control the response to remote requests are shown below.# svcprop -t -p config/local_only rpc/bindconfig/local_only boolean true# svcprop -t -p config/log_from_remote system-logconfig/log_from_remote boolean falseThe first example, for the rpcbind service, is typical of services where a new Boolean "local_only" property has been added. When this property value is set to true, the service only accepts requests from the local system. The syslog service, shown in the second example, already contained a "log_from_remote" property in previous releases of the Solaris OS. To limit syslog to accepting data from the local system, this property is set to false.SMF profiles are normally applied by using the svccfg(1M) command. However, while the generic_limited_net profile disables services, it does not set the service properties needed to configure local-only operation. Therefore, the Solaris OS provides the netservices(1M) command that applies the SMF profile and also sets the necessary SMF service properties to restrict services to local requests only.2.1.3 Selecting Profiles at Install Time and AfterwardsWhen a system is upgraded from a previous release of Solaris, the system continues to provide the same services to the network that it did before the upgrade.On the other hand, an initial install presents the opportunity to select the limited networking configuration at install time. During an interactive installation, the user is prompted to choose whether or not to enable the traditional networking services. This choice can also be made automatically during a Jump-Start installation. Currently, a non-interactive installation defaults to the traditional, or open, configuration, but this default may change to the limited networking configuration in future releases of the Solaris OS.As described later in this chapter, a different SMF profile can be applied at any time after installation. However, the advantage of selecting the limited_net profile during installation is that the system is protected from potentially hostile network traffic from the first time that it boots. If the system boots with services that accept requests from the network, then there is at least a brief period during which the system is vulnerable before the administrator can disable services or limit them to local requests only.For a system that is running in the open configuration, either because of a choice at install time or because it was upgraded from a previous Solaris release, the limited networking configuration can be applied at any time afterwards using the netservices command.# netservices limited2.1.4 Results of Applying Network ProfilesA system that is running the traditional open configuration shows a large number of enabled inetd services.Similarly, the SMF properties for syslogd(1M) and rpcbind(1M) show that they are

configured to accept requests from the network.After the limited-networking profile is applied, the set of enabled inetd ser-vices is dramatically reduced.The syslogd and rpcbind services are still enabled for local clients, but these services are configured to ignore requests from the network.The difference can also be seen by examining the output of the nmap1 command, which shows the open ports that are visible to another system on the network. With the open configuration, we see the following open ports:After applying the limited networking configuration, the only open ports are for sshd and rpcbind. Although the rpcbind port is visible to a network scan, it does not accept connections from remote clients due to the setting of the config/local_only property.2.2 Configuration HardeningDisabling or restricting access to network services is just one aspect of operating-system hardening. Once the network attack surface has been minimized and unnecessary services have been disabled, you may want to turn inward and look at how the configuration of the operating system and the remaining operational services can be adjusted to improve the overall security of the system.This section highlights a number of ways in which you can adjust the default configuration to improve the overall security of the system. The degree to which you harden the OS is wholly dependent upon your operational and environmental conditions, but as a general rule the following are generally accepted operating system security recommendations. Protect the OpenBoot PROM Enable Solaris Non-Executable Stacks Log Core Dumps Lock Out AccountsThis list is not intended to be complete, but rather is representative of the types of steps often taken by organizations to further lock down their systems. Each of the steps described in this section changes a Solaris OS default value. Note that systems already in deployment must be evaluated to ensure that even the security settings provided by default are still in place or that any changes are understood and well documented.2.2.1 Protect the OpenBoot PROMSun's SPARC hardware provides console-security features that can prevent EEPROM changes, hardware command execution, and even system startup without the appropriate password. This password protection only works while the system is at the OpenBoot PROM level (when the operating system is stopped) or immediately after the system is powered on before the operating system is started.The OpenBoot PROM password is not related to the Solaris OS root password, and it should not be set as the same password. Once set, the OpenBoot PROM password is not displayed, but can be retrieved in clear text form. When changing the OpenBoot PROM password, the system does not ask for the old password prior to changing it to the new one. In some environments, it may make more sense to set the OpenBoot PROM password to something known to the hardware technicians. The two security modes available are command and full.Unless an authorized user has the correct password, the command security mode prevents EEPROM changes and hardware-command execution while at the OpenBoot PROM level. The full security mode provides the restrictions of the command mode and, in addition, does not allow the system to boot without the correct OpenBoot PROM password. The full security mode requires operator interaction to boot the system; the system will not boot without first entering the correct password. Do not use the full security mode on servers or other systems

that must boot quickly without manual intervention.2Use the /usr/sbin/eeprom command to set the OpenBoot PROM security mode and password. In this example, the OpenBoot PROM security mode is set to command.2.2.2 Enable Solaris Non-Executable StacksNon-executable stacks provide a very useful technique for thwarting certain kinds of buffer overflow attacks. Initially developed for the SPARC platform, non-executable stack protection is also now available on x64 systems supporting either the Execute Disable (XD) bit on Intel platforms or the No Execute (NX) bit on AMD platforms.As with other kernel parameters, the non-executable stack state is configured (enabled or disabled) by a setting in the /etc/system file. For example, the following statement added to the /etc/system file would enable this feature:set noexec_user_stack=1Once this setting has been made, the system must be rebooted for the change to take effect. It is also possible to compile a program so that it runs with a non-executable stack regardless of the system configuration. For more details applicable to both developers and system administrators, see http://blogs.sun.com/gbrunett/tags/noexstk.2.2.3 Log Core DumpsThe operating system writes out a core file for a process when the process is terminated due to receiving certain signals. Such signals can be generated when the process is unable to handle an unexpected condition. Core dumps can provide an early warning of an attacker attempting to overflow a buffer in a running process.A core file is a disk copy of the contents of the process address space at the time the process received the signal, along with additional information about the state of the process. The Solaris OS allows you to adjust quite a few parameters that are related to the capture and logging of core dumps files.One configuration change often implemented is to send a notice to the system logging service when a program “dumps core.” This kind of notification can be used to trigger administrative action to determine if there is a problem with the service or if the system may be under attack.To enable the logging of core dump events, simply run the /usr/bin/coreadm command:# /usr/bin/coreadm -e logAfter this command is run, future core dump events are logged to the syslog facility.2.2.4 Lock Out AccountsAnother system-wide parameter that is often used is account lockout. Often referred to as a “three strikes policy,” account lockout offers you a way of locking individual user accounts after a specified number of consecutive, failed authentication attempts (note that this number does not have to be three!).Account lockout can be enabled on a per-user basis or globally where it impacts all users on the system who are not specifically excluded from the policy. Account lockout is also service agnostic, in that the consecutive failed authentication attempts need not be from the same service. In this way, the operating system can detect brute-force authentication attempts that could start with Secure Shell and move to other services like TELNET or FTP.To enable account lockout globally, you set the LOCK_AFTER_RETRIES parameter to YES in the /etc/security/policy.conf file. By default, the root account is excluded from this policy. If you want to have other exclusions or simply want to specify a different per-user policy, you must edit the /etc/user_attr file and/or each impacted user, set the lock_after_retries parameter to either yes or no based on your intent. To specify how many consecutive failed authentication attempts are permitted before an account is locked, set the RETRIES parameter

in the /etc/default/login file. By default, this value is 5.Note that once an account is locked, it can only be unlocked by administrative intervention. A system administrator must unlock the account using the -u option to the /usr/bin/passwd command before that account can be successfully accessed.For more information, see http://blogs.sun.com/gbrunett/entry/solaris_10_account_lockout_three.2.2.5 Other Security Configuration IdeasThis section has provided just a small window into the various security hardening configuration settings that exist in the Solaris OS. There are many more systemwide as well as service-specific configuration settings that can be adjusted if necessary. As a general rule, the default values provided in the Solaris OS are suitable for most organizations, but particular cases might require additional configuration changes. Beyond the Solaris product documentation, we recommend that you review the publications that are listed at the end of this chapter for specific hardening suggestions.32.3 Basic Audit and Reporting ToolThe Basic Audit and Reporting Tool (BART) provides a quick and easy way to collect information on filesystem objects and their attributes so that, at a later time, you can determine whether there have been any changes. While this kind of functionality is clearly useful for security incident detection, BART is also often used as part of a larger change management process to validate approved changes and to detect those that may have occurred outside of an approved process.The BART tool, /usr/bin/bart, collects such information as an object's UID, GID, permissions, access control lists, modification time, size, and type. In addition, for files, BART generates an MD5 fingerprint from the contents of the file so that the integrity of a file's contents can also be verified. The exact attributes that are collected depend on the type of object being evaluated. For a full list of the attributes that can be collected, see the bart_rules(4) manual page.BART has two primary modes of operation: create and compare.2.3.1 Collecting Filesystem Object Attributes (Create Mode)When run in create mode, BART collects file system object information from a system. You can control the scope of collection on a system. You can include the entire system, all files under a specified directory, or just a subset of files. You can even define a more granular policy by using a rules file that can be customized to meet your organization's requirements.When you use BART in create mode, BART reads its rules file from either standard input or from a regular file. As BART processes individual file system objects, it records its results in a manifest file. This manifest is directed to standard output by default, although you can easily redirect the output to a file or to another process.For example, to create a BART manifest called /root/bart.control based upon the files under /usr/lib/nis, the following command could be used:# find /usr/lib/nis | bart create -I > /root/bart.control2.3.2 Comparing Filesystem Object Attributes (Compare Mode)To use BART in compare mode, you need two BART manifests and, optionally, a rules file. The first (and original) manifest, called the control manifest, is used as your baseline. The second manifest, called the test manifest, is then compared against the control. The manifests are compared in accordance with a set of rules, if supplied. If a rules file is specified, then BART uses the rules to determine how to make the various comparisons. One of the benefits of rules files is that they can help eliminate any false alarms in your reports, thereby allowing you to better focus your efforts on the

remaining alarms.You can also use BART to compare any two independent BART snapshots to determine whether the objects being assessed have been changed. For example, with BART, you can quickly and easily answer the question: “Has this file changed since yesterday?”For example, building upon the last example, create a test manifest for the /usr/lib/nis directory:# find /usr/lib/nis | bart create -I > /root/bart.testOnce both the control and test manifests have been created, you can compare them to look for any differences:# bart compare /root/bart.control /root/bart.test/usr/lib/nis/foo: add/usr/lib/nis/nisping: uid control:0 test:2In this case, the /usr/lib/nis/foo file has been added to the /usr/lib/nis directory, and the /usr/lib/nis/nisping program has had its owner changed from root (uid 0) to bin (uid 2). These changes happened after the control manifest was first created.Using the cron(1M) command, you can generate BART manifests every minute, hour, day, week, or month. How often you actually run BART to generate new manifests is based on how critical your need is to detect change. Because one size does not fit all, you might even want to consider having different BART rules or policies. For example, you might have two policies: A smaller policy, targeting a few key files, that runs every minute or hour A larger policy, collecting information across the entire system, that runs only once a day or once a weekUse common sense when deciding how often to generate manifests. You need to balance your detection priorities with the I/O load that the collection process generates on the system. For more information on automating the collection and processing of BART manifests, see the Sun BluePrint article titled “Automating Solaris 10 File Integrity Checks,” which can be found at http://www.sun.com/blueprints/0305/819-2259.pdf.2.3.3 Determining Whether a Filesystem Object Is GenuineRegardless of the selected time intervals, however, BART is still not able to definitively answer the question: “Is this a genuine object that Sun shipped?” BART cannot answer this question because you need to manually create the control manifest after the system has been installed. Someone or something could have changed a Sun-provided file from its default before you performed your first BART snapshot, particularly if the first BART manifest is generated well after installation. Remember that, if your control manifest is somehow corrupt, then all of the later comparisons against the original manifest are suspect.2.4 Signed ELF Filesystem ObjectsTo determine if a filesystem object is indeed genuine, you can query its cryptographic signature. Starting in the Solaris 10 OS, most operating system Executable and Linkable Format (ELF) objects have been cryptographically signed by Sun. The signatures on these files can be validated by using the elfsign(1) command.# elfsign verify -e /usr/bin/suelfsign: verification of /usr/bin/su passed.While this is a very strong way of validating the integrity of a file, this approach is only applicable to ELF objects such as binaries, libraries, device drivers, and other compiled code. Also, due to license or other restrictions, some binaries that are shipped in a Solaris release cannot be modified and therefore the signature cannot be added to those files. Last, text-based configuration and log files and other non-binary content are not protected in this manner. The Solaris Fingerprint Database, discussed in the next section, can validate files that are not signed.2.5 Solaris Fingerprint Database (sfpDB)The Solaris Fingerprint Database (sfpDB) is a free SunSolve

Online service that enables organizations to verify the integrity of files that are distributed with the Solaris OS. Organizations can, at any time, query the sfpDB to determine whether an operating system file did indeed come from Sun.The sfpDB is a powerful tool for organizations that want to validate the integrity of their operating systems, baseline snapshots, and patches—or even to assist during digital forensic investigations in which the integrity of objects might be called into question.The Solaris Fingerprint Database is itself a collection of file fingerprints that have been created from the contents of Solaris OS media kits, unbundled software, and patches. These fingerprints are created as part of the release process in order to ensure that the fingerprints are current and reflect actual shipping versions of files provided by Sun. By submitting fingerprints to the Solaris Fingerprint Database service, organizations can then determine which, if any, of the submitted fingerprints belongs to an actual file supplied by Sun.2.5.1 How the sfpDB Tool WorksThe Solaris Fingerprint Database is a collection of file fingerprints that are generated by using the MD5 Message Digest Algorithm, which is defined in RFC 1321:The algorithm takes as input a message of arbitrary length and produces as output a 128-bit “fingerprint” or “message digest” of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest.By using this algorithm, it is expected that no two different files will share the same fingerprint. While this assurance is not necessarily as strong as a cryptographic signature, it is virtually impossible to modify a file in such as way as to retain its original MD5 fingerprint. Note that, in September 2004, researchers announced that the MD5 algorithm was “broken.” That is, they had developed a method to produce two pre-images (for example, files) with the same MD5 hash value. While the announcement is important, this breakthrough does not impact the validity of the Solaris Fingerprint Database. As of the publication of this book, no feasible method exists to produce a pre-image (such as a text or binary file) that matches a given hash value. That is, no file could be replaced with another, different file and still retain its original MD5 hash value. For more information about the resistance of hash functions to attack, see, for example, http://www.schneier.com/crypto-gram-0409.html#3.To use the Solaris Fingerprint Database service, you calculate MD5 finger-prints for the files you want to check, and then submit them to the database service for processing. The sfpDB service evaluates the submitted fingerprints and determines if a match exists for any of the objects.For each fingerprint match, the information shown in Table 2.2 is provided.Table 2.2 Fingerprint Database FieldsNote that a single file fingerprint can result in several matches. Text files often generate several matches because they do not depend on the underlying operating system version or hardware platform. Examples of text files are configuration files and shell scripts.2.5.2 Creating an MD5 File FingerprintThe Solaris 10 OS contains a program that can be used to generate MD5 finger-prints: /usr/bin/digest. The following example uses the digest program to create an MD5 fingerprint:# digest -v -a md5 /usr/bin/sumd5 (/usr/bin/su) = bab18f089705c1628ccdc177b802d5712.5.3 Testing an MD5 File FingerprintOnce the MD5 fingerprints have been computed, they can be submitted to

the Solaris Fingerprint Database to determine whether they correspond to values associated with valid Sun operating system files. To test the collected finger-prints:1. Visit the Solaris Fingerprint Database page at http://sunsolve.sun.com/pub-cgi/fileFingerprints.pl.After connecting to this URL, the Solaris Fingerprint Web form is displayed. Scroll to the bottom of the page to find the text window where the collected fingerprints can be entered.2. Type or copy and paste one or more MD5 fingerprints into the Web form. For example, to verify the su command fingerprint that was generated in the preceding example, paste the MD5 value into the Web form: bab18f089705c1628ccdc177b802d5713. Click the Submit button to view the results.For this example, the following result is returned:The Solaris Fingerprint Database query properly identified the /usr/bin/su program as the object to which the fingerprint belonged. It is important to not only examine the name and canonical path when evaluating the results, as there are many possible outcomes such as multiple data mismatches, matches indicating an upgrade or downgrade attack, and of course, a failure where the fingerprint that you provided was not found in the database. Each of these cases, along with a more in-depth treatment of the Solaris Fingerprint Database, is provided in the Sun BluePrint article titled The Solaris Fingerprint Database: A Security Tool for Solaris Operating Environment Files (Updated for Solaris 10), which can be found at http://www.sun.com/blueprints/0306/816-1148.pdf.It is good practice to use together both of the tools discussed in this chapter: the Basic Audit and Reporting Tool and the Solaris Fingerprint Database. More information on this integration can be found in the Sun BluePrint article titled “Integrating BART and the Solaris Fingerprint Database in the Solaris 10 Operating System,” which can be found at http://www.sun.com/blueprints/0405/819-2260.pdf.References2Hardening Solaris SystemsSolaris systems are highly configurable. Even before installing additional software, the administrator has the opportunity to make hundreds of choices about how to set up the operating system itself. These choices include which services to enable, the specific configuration settings of each service, and additional settings that apply to the system as a whole.There is no single “right answer” to any of these configuration choices. If there were, that choice could be set permanently in the operating system, and there would be no need for it to be configurable. Indeed, the process of hardening a computer system is one of making choices and trade-offs that maximize security, while still supporting the intended uses of the system. Beyond the basic techniques presented here, the publications listed at the end of this chapter provide a great deal of specific information to guide the administrator in selecting configuration settings to harden a system.Once the necessary configuration choices have been made and implemented, the system requires regular monitoring to ensure that the system remains in the hardened state. The latter half of this chapter describes tools in the Solaris OS and elsewhere that help to simplify this task.2.1 Securing Network ServicesTraditionally, many UNIX-based systems have made a wide variety of services available to remote clients over the network. By default, Solaris 10 OS installation enables a large number of network services. This open approach is convenient because it makes services available with minimal setup, and it may provide adequate security in

environments where access to the network is limited to a known and trusted set of users. However, many Solaris systems are attached to the Internet or to other networks that cannot be fully trusted. In such an environment, it is prudent to limit the set of services provided to remote clients.Of course, a service that listens to the network for requests and provides responses is not necessarily insecure. Such services are typically designed to authenticate the identity of the client in some way before performing a requested operation or divulging any non-public information. However, sometimes even the “public” information that is provided without authentication can be useful to an attacker. More importantly, the programs that implement these services, like all programs, may contain bugs, and accepting input from the network makes it easier for an attacker to exploit remotely and anonymously any vulnerabilities that may exist.To reduce this risk to an acceptable level, the guiding principle is to minimize the set of opportunities available to an attacker, sometimes called the “attack surface,” by providing only the services that are necessary for the purpose of the system. Furthermore, it is important to disable unnecessary services from the moment the system is connected to the network. If a vulnerable service is enabled briefly after installation or each time the system is booted until the service is disabled, then a window of opportunity exists for an attacker to exploit the vulnerability until the service is disabled.2.1.1 Minimizing Installed SoftwareOne approach to limit the network services that are provided by a system is to avoid installing the software that provides unneeded services. All of the programs that are delivered with a Solaris release are divided into packages, with related programs grouped together in the same package. Of course, if a package is not installed, the programs it contains cannot run, and therefore they cannot provide any services that are accessible from the network.Beginning in the Solaris 9 release, the common network services provided by inetd(1M) were divided among several new packages; for example, the packages related to the telnet program are shown below. Discrete packaging allows the administrator to limit the network services provided by customizing the installation to leave out some packages. Individual packages can also be removed after installation using pkgrm(1M).# pkginfo SUNWtnetc SUNWtnetd SUNWtnetrsystem SUNWtnetc Telnet Command (client)system SUNWtnetd Telnet Server Daemon (Usr)system SUNWtnetr Telnet Server Daemon (Root)The Solaris 10 release expanded upon this idea by adding a Reduced Networking selection at install time to complement the existing Core, End User, Developer, and Entire Distribution choices. Each of these options is an install metacluster, which defines the set of packages to be installed. The Reduced Networking metacluster eliminates as many networking services as possible while leaving the system operational enough to allow further configuration. Of course, this configuration must be done by an administrator at the system console, because there are no services running to allow for remote access and administration.In most cases, a system installed with the Reduced Networking metacluster is too limited to be useful without adding additional packages. Completely eliminating network services has a severe impact on local users as well; for example, a Reduced Networking system does not allow local users to log in via the window system. Rather than using

the installed system as-is, the Reduced Networking metacluster is intended to serve as a foundation to which the administrator can add only those packages that are desired for the particular system.While removing packages is an effective way to eliminate network services, this approach has several practical drawbacks. The first is that the level of granularity may be insufficient. Although services are divided among more packages than in the past, it is still common to have several related services provided by a single package, and choosing to install a package means installing all of the services it provides. Second, it is inconvenient to add services later to adapt to changing requirements, since this requires access to the original installation media in order to add the relevant packages. Finally, packages that are added some time after installation will not have the benefit of any bug fixes contained in patches that may have been applied to the system after installation.For details about Sun's support for minimized systems, refer to http://www.opensolaris.org/os/community/security/files/minimization-support-rules-ext.pdf.2.1.2 Using SMF to Control Network ServicesThe Solaris Service Management Facility(SMF), introduced in the Solaris 10 OS, provides a general mechanism for the control and configuration of system services. The fundamental unit of administration in SMF is the service instance. An instance is a specific configuration of a service. For example, a Web server is a service. A single Web server daemon that is configured to accept Web service requests on port 80 is an instance of that service. Multiple instances of the same service can run on a single Solaris OS. Each service instance is given a Fault Management Resource Identifier (FMRI). The FMRI includes the service name and the instance name. When you list a service using the svcs(1) command, the listing includes the status, date, service name, and service instance. In the next case, "network/dhcp-server" identifies the service, and "default" identifies the instance.# svcs -a | grep dhcpdisabled Apr 10 svc:/network/dhcp-server:defaultBecause SMF separates configuration of a service from its installation, it is possible to install the entire Solaris OS but enable only the services that are needed on the particular system. To simplify the management process, SMF bundles together the configuration settings for many services into a profile. Solaris includes two generic predefined profiles in /var/svc/profile called generic_open.xml and generic_limited_net.xml. The open profile provides the traditional Solaris configuration with many network services enabled. The limited_net profile provides no services to remote clients except for the Secure Shell daemon sshd(1M).An exception is made for sshd because it is frequently necessary to have some mechanism that allows the administrator to log in and administer the system remotely. Since the need to administer the system remotely may occur at unpredictable times, it is impractical to enable sshd only when it is needed. For this reason, sshd is left enabled by default. Secure Shell is a safer choice for remote administration than telnet or rlogin because it protects the privacy and integrity of the communication session. An administrator who does not want to leave sshd enabled can disable this service individually using the svcadm(1M) command as follows:# svcadm disable sshThe goal of the limited_net profile is to avoid providing network services to remote clients without affecting the normal activities of local users. The services affected by the limited_net profile are shown in Table

2.1.Table 2.1 Services Affected by the limited_net ProfileFor services that are normally used only by remote clients, it is straightforward for the profile to simply disable those services. Other services, including several that are associated with the window system, are also needed by local users. For these services, SMF properties are used to configure the services to respond only to requests originating from the local system. Two examples of properties that control the response to remote requests are shown below.# svcprop -t -p config/local_only rpc/bindconfig/local_only boolean true# svcprop -t -p config/log_from_remote system-logconfig/log_from_remote boolean falseThe first example, for the rpcbind service, is typical of services where a new Boolean “local_only” property has been added. When this property value is set to true, the service only accepts requests from the local system. The syslog service, shown in the second example, already contained a “log_from_remote” property in previous releases of the Solaris OS. To limit syslog to accepting data from the local system, this property is set to false.SMF profiles are normally applied by using the svccfg(1M) command. However, while the generic_limited_net profile disables services, it does not set the service properties needed to configure local-only operation. Therefore, the Solaris OS provides the netservices(1M) command that applies the SMF profile and also sets the necessary SMF service properties to restrict services to local requests only.2.1.3 Selecting Profiles at Install Time and AfterwardsWhen a system is upgraded from a previous release of Solaris, the system continues to provide the same services to the network that it did before the upgrade.On the other hand, an initial install presents the opportunity to select the limited networking configuration at install time. During an interactive installation, the user is prompted to choose whether or not to enable the traditional networking services. This choice can also be made automatically during a Jump-Start installation. Currently, a non-interactive installation defaults to the traditional, or open, configuration, but this default may change to the limited networking configuration in future releases of the Solaris OS.As described later in this chapter, a different SMF profile can be applied at any time after installation. However, the advantage of selecting the limited_net profile during installation is that the system is protected from potentially hostile network traffic from the first time that it boots. If the system boots with services that accept requests from the network, then there is at least a brief period during which the system is vulnerable before the administrator can disable services or limit them to local requests only.For a system that is running in the open configuration, either because of a choice at install time or because it was upgraded from a previous Solaris release, the limited networking configuration can be applied at any time afterwards using the netservices command.# netservices limited2.1.4 Results of Applying Network ProfilesA system that is running the traditional open configuration shows a large number of enabled inetd services.Similarly, the SMF properties for syslogd(1M) and rpcbind(1M) show that they are configured to accept requests from the network.After the limited-networking profile is applied, the set of enabled inetd ser-vices is dramatically reduced.The syslogd and rpcbind services are still enabled for local clients, but these services are configured to ignore requests from the network.The difference can also be seen by examining the output of the nmap1 command,

which shows the open ports that are visible to another system on the network. With the open configuration, we see the following open ports:After applying the limited networking configuration, the only open ports are for sshd and rpcbind. Although the rpcbind port is visible to a network scan, it does not accept connections from remote clients due to the setting of the config/local_only property.2.2 Configuration HardeningDisabling or restricting access to network services is just one aspect of operating-system hardening. Once the network attack surface has been minimized and unnecessary services have been disabled, you may want to turn inward and look at how the configuration of the operating system and the remaining operational services can be adjusted to improve the overall security of the system.This section highlights a number of ways in which you can adjust the default configuration to improve the overall security of the system. The degree to which you harden the OS is wholly dependent upon your operational and environmental conditions, but as a general rule the following are generally accepted operating system security recommendations. Protect the OpenBoot PROM Enable Solaris Non-Executable Stacks Log Core Dumps Lock Out AccountsThis list is not intended to be complete, but rather is representative of the types of steps often taken by organizations to further lock down their systems. Each of the steps described in this section changes a Solaris OS default value. Note that systems already in deployment must be evaluated to ensure that even the security settings provided by default are still in place or that any changes are understood and well documented.2.2.1 Protect the OpenBoot PROMSun's SPARC hardware provides console-security features that can prevent EEPROM changes, hardware command execution, and even system startup without the appropriate password. This password protection only works while the system is at the OpenBoot PROM level (when the operating system is stopped) or immediately after the system is powered on before the operating system is started.The OpenBoot PROM password is not related to the Solaris OS root password, and it should not be set as the same password. Once set, the OpenBoot PROM password is not displayed, but can be retrieved in clear text form. When changing the OpenBoot PROM password, the system does not ask for the old password prior to changing it to the new one. In some environments, it may make more sense to set the OpenBoot PROM password to something known to the hardware technicians. The two security modes available are command and full.Unless an authorized user has the correct password, the command security mode prevents EEPROM changes and hardware-command execution while at the OpenBoot PROM level. The full security mode provides the restrictions of the command mode and, in addition, does not allow the system to boot without the correct OpenBoot PROM password. The full security mode requires operator interaction to boot the system; the system will not boot without first entering the correct password. Do not use the full security mode on servers or other systems that must boot quickly without manual intervention.2Use the /usr/sbin/eeprom command to set the OpenBoot PROM security mode and password. In this example, the OpenBoot PROM security mode is set to command.2.2.2 Enable Solaris Non-Executable StacksNon-executable stacks provide a very useful technique for thwarting certain kinds of buffer overflow attacks.

Initially developed for the SPARC platform, non-executable stack protection is also now available on x64 systems supporting either the Execute Disable (XD) bit on Intel platforms or the No Execute (NX) bit on AMD platforms.As with other kernel parameters, the non-executable stack state is configured (enabled or disabled) by a setting in the /etc/system file. For example, the following statement added to the /etc/system file would enable this feature:set noexec_user_stack=1Once this setting has been made, the system must be rebooted for the change to take effect. It is also possible to compile a program so that it runs with a non-executable stack regardless of the system configuration. For more details applicable to both developers and system administrators, see http://blogs.sun.com/gbrunett/tags/noexstk.2.2.3 Log Core DumpsThe operating system writes out a core file for a process when the process is terminated due to receiving certain signals. Such signals can be generated when the process is unable to handle an unexpected condition. Core dumps can provide an early warning of an attacker attempting to overflow a buffer in a running process.A core file is a disk copy of the contents of the process address space at the time the process received the signal, along with additional information about the state of the process. The Solaris OS allows you to adjust quite a few parameters that are related to the capture and logging of core dumps files.One configuration change often implemented is to send a notice to the system logging service when a program "dumps core." This kind of notification can be used to trigger administrative action to determine if there is a problem with the service or if the system may be under attack.To enable the logging of core dump events, simply run the /usr/bin/coreadm command:# /usr/bin/coreadm -e logAfter this command is run, future core dump events are logged to the syslog facility.2.2.4 Lock Out AccountsAnother system-wide parameter that is often used is account lockout. Often referred to as a "three strikes policy," account lockout offers you a way of locking individual user accounts after a specified number of consecutive, failed authentication attempts (note that this number does not have to be three!).Account lockout can be enabled on a per-user basis or globally where it impacts all users on the system who are not specifically excluded from the policy. Account lockout is also service agnostic, in that the consecutive failed authentication attempts need not be from the same service. In this way, the operating system can detect brute-force authentication attempts that could start with Secure Shell and move to other services like TELNET or FTP.To enable account lockout globally, you set the LOCK_AFTER_RETRIES parameter to YES in the /etc/security/policy.conf file. By default, the root account is excluded from this policy. If you want to have other exclusions or simply want to specify a different per-user policy, you must edit the /etc/user_attr file and/or each impacted user, set the lock_after_retries parameter to either yes or no based on your intent. To specify how many consecutive failed authentication attempts are permitted before an account is locked, set the RETRIES parameter in the /etc/default/login file. By default, this value is 5.Note that once an account is locked, it can only be unlocked by administrative intervention. A system administrator must unlock the account using the -u option to the /usr/bin/passwd command before that account can be successfully accessed.For more information, see http://blogs.sun.com/gbrunett/entry/

solaris_10_account_lockout_three.2.2.5 Other Security Configuration IdeasThis section has provided just a small window into the various security hardening configuration settings that exist in the Solaris OS. There are many more systemwide as well as service-specific configuration settings that can be adjusted if necessary. As a general rule, the default values provided in the Solaris OS are suitable for most organizations, but particular cases might require additional configuration changes. Beyond the Solaris product documentation, we recommend that you review the publications that are listed at the end of this chapter for specific hardening suggestions.32.3 Basic Audit and Reporting ToolThe Basic Audit and Reporting Tool (BART) provides a quick and easy way to collect information on filesystem objects and their attributes so that, at a later time, you can determine whether there have been any changes. While this kind of functionality is clearly useful for security incident detection, BART is also often used as part of a larger change management process to validate approved changes and to detect those that may have occurred outside of an approved process.The BART tool, /usr/bin/bart, collects such information as an object's UID, GID, permissions, access control lists, modification time, size, and type. In addition, for files, BART generates an MD5 fingerprint from the contents of the file so that the integrity of a file's contents can also be verified. The exact attributes that are collected depend on the type of object being evaluated. For a full list of the attributes that can be collected, see the bart_rules(4) manual page.BART has two primary modes of operation: create and compare.2.3.1 Collecting Filesystem Object Attributes (Create Mode)When run in create mode, BART collects file system object information from a system. You can control the scope of collection on a system. You can include the entire system, all files under a specified directory, or just a subset of files. You can even define a more granular policy by using a rules file that can be customized to meet your organization's requirements.When you use BART in create mode, BART reads its rules file from either standard input or from a regular file. As BART processes individual file system objects, it records its results in a manifest file. This manifest is directed to standard output by default, although you can easily redirect the output to a file or to another process.For example, to create a BART manifest called /root/bart.control based upon the files under /usr/lib/nis, the following command could be used:# find /usr/lib/nis | bart create -I > /root/bart.control2.3.2 Comparing Filesystem Object Attributes (Compare Mode)To use BART in compare mode, you need two BART manifests and, optionally, a rules file. The first (and original) manifest, called the control manifest, is used as your baseline. The second manifest, called the test manifest, is then compared against the control. The manifests are compared in accordance with a set of rules, if supplied. If a rules file is specified, then BART uses the rules to determine how to make the various comparisons. One of the benefits of rules files is that they can help eliminate any false alarms in your reports, thereby allowing you to better focus your efforts on the remaining alarms.You can also use BART to compare any two independent BART snapshots to determine whether the objects being assessed have been changed. For example, with BART, you can quickly and easily answer the question: "Has this file changed since yesterday?"For example, building upon the last example, create a test manifest for the /usr/lib/nis directory:#

find /usr/lib/nis | bart create -I > /root/bart.testOnce both the control and test manifests have been created, you can compare them to look for any differences:# bart compare /root/bart.control /root/bart.test/usr/lib/nis/foo: add/usr/lib/nis/nisping: uid control:0 test:2In this case, the /usr/lib/nis/foo file has been added to the /usr/lib/nis directory, and the /usr/lib/nis/nisping program has had its owner changed from root (uid 0) to bin (uid 2). These changes happened after the control manifest was first created.Using the cron(1M) command, you can generate BART manifests every minute, hour, day, week, or month. How often you actually run BART to generate new manifests is based on how critical your need is to detect change. Because one size does not fit all, you might even want to consider having different BART rules or policies. For example, you might have two policies: A smaller policy, targeting a few key files, that runs every minute or hour A larger policy, collecting information across the entire system, that runs only once a day or once a weekUse common sense when deciding how often to generate manifests. You need to balance your detection priorities with the I/O load that the collection process generates on the system. For more information on automating the collection and processing of BART manifests, see the Sun BluePrint article titled “Automating Solaris 10 File Integrity Checks,” which can be found at http://www.sun.com/blueprints/0305/819-2259.pdf.2.3.3 Determining Whether a Filesystem Object Is GenuineRegardless of the selected time intervals, however, BART is still not able to definitively answer the question: “Is this a genuine object that Sun shipped?” BART cannot answer this question because you need to manually create the control manifest after the system has been installed. Someone or something could have changed a Sun-provided file from its default before you performed your first BART snapshot, particularly if the first BART manifest is generated well after installation. Remember that, if your control manifest is somehow corrupt, then all of the later comparisons against the original manifest are suspect.2.4 Signed ELF Filesystem ObjectsTo determine if a filesystem object is indeed genuine, you can query its cryptographic signature. Starting in the Solaris 10 OS, most operating system Executable and Linkable Format (ELF) objects have been cryptographically signed by Sun. The signatures on these files can be validated by using the elfsign(1) command.# elfsign verify -e /usr/bin/suelfsign: verification of /usr/bin/su passed.While this is a very strong way of validating the integrity of a file, this approach is only applicable to ELF objects such as binaries, libraries, device drivers, and other compiled code. Also, due to license or other restrictions, some binaries that are shipped in a Solaris release cannot be modified and therefore the signature cannot be added to those files. Last, text-based configuration and log files and other non-binary content are not protected in this manner. The Solaris Fingerprint Database, discussed in the next section, can validate files that are not signed.2.5 Solaris Fingerprint Database (sfpDB)The Solaris Fingerprint Database (sfpDB) is a free SunSolve Online service that enables organizations to verify the integrity of files that are distributed with the Solaris OS. Organizations can, at any time, query the sfpDB to determine whether an operating system file did indeed come from Sun.The sfpDB is a powerful tool for organizations that want to validate the integrity of their operating systems, baseline snapshots, and patches—

or even to assist during digital forensic investigations in which the integrity of objects might be called into question.The Solaris Fingerprint Database is itself a collection of file fingerprints that have been created from the contents of Solaris OS media kits, unbundled software, and patches. These fingerprints are created as part of the release process in order to ensure that the fingerprints are current and reflect actual shipping versions of files provided by Sun. By submitting fingerprints to the Solaris Fingerprint Database service, organizations can then determine which, if any, of the submitted fingerprints belongs to an actual file supplied by Sun.2.5.1 How the sfpDB Tool WorksThe Solaris Fingerprint Database is a collection of file fingerprints that are generated by using the MD5 Message Digest Algorithm, which is defined in RFC 1321:The algorithm takes as input a message of arbitrary length and produces as output a 128-bit “fingerprint” or “message digest” of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest.By using this algorithm, it is expected that no two different files will share the same fingerprint. While this assurance is not necessarily as strong as a cryptographic signature, it is virtually impossible to modify a file in such as way as to retain its original MD5 fingerprint. Note that, in September 2004, researchers announced that the MD5 algorithm was “broken.” That is, they had developed a method to produce two pre-images (for example, files) with the same MD5 hash value. While the announcement is important, this breakthrough does not impact the validity of the Solaris Fingerprint Database. As of the publication of this book, no feasible method exists to produce a pre-image (such as a text or binary file) that matches a given hash value. That is, no file could be replaced with another, different file and still retain its original MD5 hash value. For more information about the resistance of hash functions to attack, see, for example, http://www.schneier.com/crypto-gram-0409.html#3.To use the Solaris Fingerprint Database service, you calculate MD5 finger-prints for the files you want to check, and then submit them to the database service for processing. The sfpDB service evaluates the submitted fingerprints and determines if a match exists for any of the objects.For each fingerprint match, the information shown in Table 2.2 is provided.Table 2.2 Fingerprint Database FieldsNote that a single file fingerprint can result in several matches. Text files often generate several matches because they do not depend on the underlying operating system version or hardware platform. Examples of text files are configuration files and shell scripts.2.5.2 Creating an MD5 File FingerprintThe Solaris 10 OS contains a program that can be used to generate MD5 finger-prints: /usr/bin/digest. The following example uses the digest program to create an MD5 fingerprint:# digest -v -a md5 /usr/bin/sumd5 (/usr/bin/su) = bab18f089705c1628ccdc177b802d5712.5.3 Testing an MD5 File FingerprintOnce the MD5 fingerprints have been computed, they can be submitted to the Solaris Fingerprint Database to determine whether they correspond to values associated with valid Sun operating system files. To test the collected finger-prints:1. Visit the Solaris Fingerprint Database page at http://sunsolve.sun.com/pub-cgi/fileFingerprints.pl.After connecting to this URL, the Solaris Fingerprint Web form is displayed. Scroll to the bottom of the

page to find the text window where the collected fingerprints can be entered.2. Type or copy and paste one or more MD5 fingerprints into the Web form. For example, to verify the su command fingerprint that was generated in the preceding example, paste the MD5 value into the Web form: bab18f089705c1628ccdc177b802d5713. Click the Submit button to view the results.For this example, the following result is returned:The Solaris Fingerprint Database query properly identified the /usr/bin/su program as the object to which the fingerprint belonged. It is important to not only examine the name and canonical path when evaluating the results, as there are many possible outcomes such as multiple data mismatches, matches indicating an upgrade or downgrade attack, and of course, a failure where the fingerprint that you provided was not found in the database. Each of these cases, along with a more in-depth treatment of the Solaris Fingerprint Database, is provided in the Sun BluePrint article titled The Solaris Fingerprint Database: A Security Tool for Solaris Operating Environment Files (Updated for Solaris 10), which can be found at http://www.sun.com/blueprints/0306/816-1148.pdf.It is good practice to use together both of the tools discussed in this chapter: the Basic Audit and Reporting Tool and the Solaris Fingerprint Database. More information on this integration can be found in the Sun BluePrint article titled “Integrating BART and the Solaris Fingerprint Database in the Solaris 10 Operating System,” which can be found at http://www.sun.com/blueprints/0405/819-2260.pdf.References2Hardening Solaris SystemsSolaris systems are highly configurable. Even before installing additional software, the administrator has the opportunity to make hundreds of choices about how to set up the operating system itself. These choices include which services to enable, the specific configuration settings of each service, and additional settings that apply to the system as a whole.There is no single “right answer” to any of these configuration choices. If there were, that choice could be set permanently in the operating system, and there would be no need for it to be configurable. Indeed, the process of hardening a computer system is one of making choices and trade-offs that maximize security, while still supporting the intended uses of the system. Beyond the basic techniques presented here, the publications listed at the end of this chapter provide a great deal of specific information to guide the administrator in selecting configuration settings to harden a system.Once the necessary configuration choices have been made and implemented, the system requires regular monitoring to ensure that the system remains in the hardened state. The latter half of this chapter describes tools in the Solaris OS and elsewhere that help to simplify this task.2.1 Securing Network ServicesTraditionally, many UNIX-based systems have made a wide variety of services available to remote clients over the network. By default, Solaris 10 OS installation enables a large number of network services. This open approach is convenient because it makes services available with minimal setup, and it may provide adequate security in environments where access to the network is limited to a known and trusted set of users. However, many Solaris systems are attached to the Internet or to other networks that cannot be fully trusted. In such an environment, it is prudent to limit the set of services provided to remote clients.Of course, a service that listens to the network for requests and provides responses is

not necessarily insecure. Such services are typically designed to authenticate the identity of the client in some way before performing a requested operation or divulging any non-public information. However, sometimes even the “public” information that is provided without authentication can be useful to an attacker. More importantly, the programs that implement these services, like all programs, may contain bugs, and accepting input from the network makes it easier for an attacker to exploit remotely and anonymously any vulnerabilities that may exist.To reduce this risk to an acceptable level, the guiding principle is to minimize the set of opportunities available to an attacker, sometimes called the “attack surface,” by providing only the services that are necessary for the purpose of the system. Furthermore, it is important to disable unnecessary services from the moment the system is connected to the network. If a vulnerable service is enabled briefly after installation or each time the system is booted until the service is disabled, then a window of opportunity exists for an attacker to exploit the vulnerability until the service is disabled.2.1.1 Minimizing Installed SoftwareOne approach to limit the network services that are provided by a system is to avoid installing the software that provides unneeded services. All of the programs that are delivered with a Solaris release are divided into packages, with related programs grouped together in the same package. Of course, if a package is not installed, the programs it contains cannot run, and therefore they cannot provide any services that are accessible from the network.Beginning in the Solaris 9 release, the common network services provided by inetd(1M) were divided among several new packages; for example, the packages related to the telnet program are shown below. Discrete packaging allows the administrator to limit the network services provided by customizing the installation to leave out some packages. Individual packages can also be removed after installation using pkgrm(1M).# pkginfo SUNWtnetc SUNWtnetd SUNWtnetrsystem SUNWtnetc Telnet Command (client)system SUNWtnetd Telnet Server Daemon (Usr)system SUNWtnetr Telnet Server Daemon (Root)The Solaris 10 release expanded upon this idea by adding a Reduced Networking selection at install time to complement the existing Core, End User, Developer, and Entire Distribution choices. Each of these options is an install metacluster, which defines the set of packages to be installed. The Reduced Networking metacluster eliminates as many networking services as possible while leaving the system operational enough to allow further configuration. Of course, this configuration must be done by an administrator at the system console, because there are no services running to allow for remote access and administration.In most cases, a system installed with the Reduced Networking metacluster is too limited to be useful without adding additional packages. Completely eliminating network services has a severe impact on local users as well; for example, a Reduced Networking system does not allow local users to log in via the window system. Rather than using the installed system as-is, the Reduced Networking metacluster is intended to serve as a foundation to which the administrator can add only those packages that are desired for the particular system.While removing packages is an effective way to eliminate network services, this approach has several practical drawbacks. The first is that the level of granularity may be

insufficient. Although services are divided among more packages than in the past, it is still common to have several related services provided by a single package, and choosing to install a package means installing all of the services it provides. Second, it is inconvenient to add services later to adapt to changing requirements, since this requires access to the original installation media in order to add the relevant packages. Finally, packages that are added some time after installation will not have the benefit of any bug fixes contained in patches that may have been applied to the system after installation.For details about Sun's support for minimized systems, refer to http://www.opensolaris.org/os/community/security/files/minimization-support-rules-ext.pdf.2.1.2 Using SMF to Control Network ServicesThe Solaris Service Management Facility(SMF), introduced in the Solaris 10 OS, provides a general mechanism for the control and configuration of system services. The fundamental unit of administration in SMF is the service instance. An instance is a specific configuration of a service. For example, a Web server is a service. A single Web server daemon that is configured to accept Web service requests on port 80 is an instance of that service. Multiple instances of the same service can run on a single Solaris OS. Each service instance is given a Fault Management Resource Identifier (FMRI). The FMRI includes the service name and the instance name. When you list a service using the svcs(1) command, the listing includes the status, date, service name, and service instance. In the next case, "network/dhcp-server" identifies the service, and "default" identifies the instance.# svcs -a | grep dhcpdisabled Apr 10 svc:/network/dhcp-server:defaultBecause SMF separates configuration of a service from its installation, it is possible to install the entire Solaris OS but enable only the services that are needed on the particular system. To simplify the management process, SMF bundles together the configuration settings for many services into a profile. Solaris includes two generic predefined profiles in /var/svc/profile called generic_open.xml and generic_limited_net.xml. The open profile provides the traditional Solaris configuration with many network services enabled. The limited_net profile provides no services to remote clients except for the Secure Shell daemon sshd(1M).An exception is made for sshd because it is frequently necessary to have some mechanism that allows the administrator to log in and administer the system remotely. Since the need to administer the system remotely may occur at unpredictable times, it is impractical to enable sshd only when it is needed. For this reason, sshd is left enabled by default. Secure Shell is a safer choice for remote administration than telnet or rlogin because it protects the privacy and integrity of the communication session. An administrator who does not want to leave sshd enabled can disable this service individually using the svcadm(1M) command as follows:# svcadm disable sshThe goal of the limited_net profile is to avoid providing network services to remote clients without affecting the normal activities of local users. The services affected by the limited_net profile are shown in Table 2.1.Table 2.1 Services Affected by the limited_net ProfileFor services that are normally used only by remote clients, it is straightforward for the profile to simply disable those services. Other services, including several that are associated with the window system, are also needed by local users. For these services, SMF properties are used to configure the services to respond only to

requests originating from the local system. Two examples of properties that control the response to remote requests are shown below.# svcprop -t -p config/local_only rpc/bindconfig/local_only boolean true# svcprop -t -p config/log_from_remote system-logconfig/log_from_remote boolean falseThe first example, for the rpcbind service, is typical of services where a new Boolean “local_only” property has been added. When this property value is set to true, the service only accepts requests from the local system. The syslog service, shown in the second example, already contained a “log_from_remote” property in previous releases of the Solaris OS. To limit syslog to accepting data from the local system, this property is set to false.SMF profiles are normally applied by using the svccfg(1M) command. However, while the generic_limited_net profile disables services, it does not set the service properties needed to configure local-only operation. Therefore, the Solaris OS provides the netservices(1M) command that applies the SMF profile and also sets the necessary SMF service properties to restrict services to local requests only.2.1.3 Selecting Profiles at Install Time and AfterwardsWhen a system is upgraded from a previous release of Solaris, the system continues to provide the same services to the network that it did before the upgrade.On the other hand, an initial install presents the opportunity to select the limited networking configuration at install time. During an interactive installation, the user is prompted to choose whether or not to enable the traditional networking services. This choice can also be made automatically during a Jump-Start installation. Currently, a non-interactive installation defaults to the traditional, or open, configuration, but this default may change to the limited networking configuration in future releases of the Solaris OS.As described later in this chapter, a different SMF profile can be applied at any time after installation. However, the advantage of selecting the limited_net profile during installation is that the system is protected from potentially hostile network traffic from the first time that it boots. If the system boots with services that accept requests from the network, then there is at least a brief period during which the system is vulnerable before the administrator can disable services or limit them to local requests only.For a system that is running in the open configuration, either because of a choice at install time or because it was upgraded from a previous Solaris release, the limited networking configuration can be applied at any time afterwards using the netservices command.# netservices limited2.1.4 Results of Applying Network ProfilesA system that is running the traditional open configuration shows a large number of enabled inetd services.Similarly, the SMF properties for syslogd(1M) and rpcbind(1M) show that they are configured to accept requests from the network.After the limited-networking profile is applied, the set of enabled inetd ser-vices is dramatically reduced.The syslogd and rpcbind services are still enabled for local clients, but these services are configured to ignore requests from the network.The difference can also be seen by examining the output of the nmap1 command, which shows the open ports that are visible to another system on the network. With the open configuration, we see the following open ports:After applying the limited networking configuration, the only open ports are for sshd and rpcbind. Although the rpcbind port is visible to a network scan, it does not accept connections from remote clients due to the setting of the

config/local_only property.2.2 Configuration HardeningDisabling or restricting access to network services is just one aspect of operating-system hardening. Once the network attack surface has been minimized and unnecessary services have been disabled, you may want to turn inward and look at how the configuration of the operating system and the remaining operational services can be adjusted to improve the overall security of the system.This section highlights a number of ways in which you can adjust the default configuration to improve the overall security of the system. The degree to which you harden the OS is wholly dependent upon your operational and environmental conditions, but as a general rule the following are generally accepted operating system security recommendations. Protect the OpenBoot PROM Enable Solaris Non-Executable Stacks Log Core Dumps Lock Out AccountsThis list is not intended to be complete, but rather is representative of the types of steps often taken by organizations to further lock down their systems. Each of the steps described in this section changes a Solaris OS default value. Note that systems already in deployment must be evaluated to ensure that even the security settings provided by default are still in place or that any changes are understood and well documented.2.2.1 Protect the OpenBoot PROMSun's SPARC hardware provides console-security features that can prevent EEPROM changes, hardware command execution, and even system startup without the appropriate password. This password protection only works while the system is at the OpenBoot PROM level (when the operating system is stopped) or immediately after the system is powered on before the operating system is started.The OpenBoot PROM password is not related to the Solaris OS root password, and it should not be set as the same password. Once set, the OpenBoot PROM password is not displayed, but can be retrieved in clear text form. When changing the OpenBoot PROM password, the system does not ask for the old password prior to changing it to the new one. In some environments, it may make more sense to set the OpenBoot PROM password to something known to the hardware technicians. The two security modes available are command and full.Unless an authorized user has the correct password, the command security mode prevents EEPROM changes and hardware-command execution while at the OpenBoot PROM level. The full security mode provides the restrictions of the command mode and, in addition, does not allow the system to boot without the correct OpenBoot PROM password. The full security mode requires operator interaction to boot the system; the system will not boot without first entering the correct password. Do not use the full security mode on servers or other systems that must boot quickly without manual intervention.2Use the /usr/sbin/eeprom command to set the OpenBoot PROM security mode and password. In this example, the OpenBoot PROM security mode is set to command.2.2.2 Enable Solaris Non-Executable StacksNon-executable stacks provide a very useful technique for thwarting certain kinds of buffer overflow attacks. Initially developed for the SPARC platform, non-executable stack protection is also now available on x64 systems supporting either the Execute Disable (XD) bit on Intel platforms or the No Execute (NX) bit on AMD platforms.As with other kernel parameters, the non-executable stack state is configured (enabled or disabled) by a setting in the /etc/system file. For example, the

following statement added to the /etc/system file would enable this feature:set noexec_user_stack=1Once this setting has been made, the system must be rebooted for the change to take effect. It is also possible to compile a program so that it runs with a non-executable stack regardless of the system configuration. For more details applicable to both developers and system administrators, see http://blogs.sun.com/gbrunett/tags/noexstk.2.2.3 Log Core DumpsThe operating system writes out a core file for a process when the process is terminated due to receiving certain signals. Such signals can be generated when the process is unable to handle an unexpected condition. Core dumps can provide an early warning of an attacker attempting to overflow a buffer in a running process.A core file is a disk copy of the contents of the process address space at the time the process received the signal, along with additional information about the state of the process. The Solaris OS allows you to adjust quite a few parameters that are related to the capture and logging of core dumps files.One configuration change often implemented is to send a notice to the system logging service when a program "dumps core." This kind of notification can be used to trigger administrative action to determine if there is a problem with the service or if the system may be under attack.To enable the logging of core dump events, simply run the /usr/bin/coreadm command:# /usr/bin/coreadm -e logAfter this command is run, future core dump events are logged to the syslog facility.2.2.4 Lock Out AccountsAnother system-wide parameter that is often used is account lockout. Often referred to as a "three strikes policy," account lockout offers you a way of locking individual user accounts after a specified number of consecutive, failed authentication attempts (note that this number does not have to be three!).Account lockout can be enabled on a per-user basis or globally where it impacts all users on the system who are not specifically excluded from the policy. Account lockout is also service agnostic, in that the consecutive failed authentication attempts need not be from the same service. In this way, the operating system can detect brute-force authentication attempts that could start with Secure Shell and move to other services like TELNET or FTP.To enable account lockout globally, you set the LOCK_AFTER_RETRIES parameter to YES in the /etc/security/policy.conf file. By default, the root account is excluded from this policy. If you want to have other exclusions or simply want to specify a different per-user policy, you must edit the /etc/user_attr file and/or each impacted user, set the lock_after_retries parameter to either yes or no based on your intent. To specify how many consecutive failed authentication attempts are permitted before an account is locked, set the RETRIES parameter in the /etc/default/login file. By default, this value is 5.Note that once an account is locked, it can only be unlocked by administrative intervention. A system administrator must unlock the account using the -u option to the /usr/bin/passwd command before that account can be successfully accessed.For more information, see http://blogs.sun.com/gbrunett/entry/solaris_10_account_lockout_three.2.2.5 Other Security Configuration IdeasThis section has provided just a small window into the various security hardening configuration settings that exist in the Solaris OS. There are many more systemwide as well as service-specific configuration settings that can be adjusted if necessary. As a general rule, the default values provided in the

Solaris OS are suitable for most organizations, but particular cases might require additional configuration changes. Beyond the Solaris product documentation, we recommend that you review the publications that are listed at the end of this chapter for specific hardening suggestions.32.3 Basic Audit and Reporting ToolThe Basic Audit and Reporting Tool (BART) provides a quick and easy way to collect information on filesystem objects and their attributes so that, at a later time, you can determine whether there have been any changes. While this kind of functionality is clearly useful for security incident detection, BART is also often used as part of a larger change management process to validate approved changes and to detect those that may have occurred outside of an approved process.The BART tool, /usr/bin/bart, collects such information as an object's UID, GID, permissions, access control lists, modification time, size, and type. In addition, for files, BART generates an MD5 fingerprint from the contents of the file so that the integrity of a file's contents can also be verified. The exact attributes that are collected depend on the type of object being evaluated. For a full list of the attributes that can be collected, see the bart_rules(4) manual page.BART has two primary modes of operation: create and compare.2.3.1 Collecting Filesystem Object Attributes (Create Mode)When run in create mode, BART collects file system object information from a system. You can control the scope of collection on a system. You can include the entire system, all files under a specified directory, or just a subset of files. You can even define a more granular policy by using a rules file that can be customized to meet your organization's requirements.When you use BART in create mode, BART reads its rules file from either standard input or from a regular file. As BART processes individual file system objects, it records its results in a manifest file. This manifest is directed to standard output by default, although you can easily redirect the output to a file or to another process.For example, to create a BART manifest called /root/bart.control based upon the files under /usr/lib/nis, the following command could be used:# find /usr/lib/nis | bart create -I > /root/bart.control2.3.2 Comparing Filesystem Object Attributes (Compare Mode)To use BART in compare mode, you need two BART manifests and, optionally, a rules file. The first (and original) manifest, called the control manifest, is used as your baseline. The second manifest, called the test manifest, is then compared against the control. The manifests are compared in accordance with a set of rules, if supplied. If a rules file is specified, then BART uses the rules to determine how to make the various comparisons. One of the benefits of rules files is that they can help eliminate any false alarms in your reports, thereby allowing you to better focus your efforts on the remaining alarms.You can also use BART to compare any two independent BART snapshots to determine whether the objects being assessed have been changed. For example, with BART, you can quickly and easily answer the question: "Has this file changed since yesterday?"For example, building upon the last example, create a test manifest for the /usr/lib/nis directory:# find /usr/lib/nis | bart create -I > /root/bart.testOnce both the control and test manifests have been created, you can compare them to look for any differences:# bart compare /root/bart.control /root/bart.test/usr/lib/nis/foo: add/usr/lib/nis/nisping: uid control:0 test:2In this case, the /usr/lib/nis/foo file has been added to the /usr/lib/nis directory, and the /usr/lib/nis/nisping

program has had its owner changed from root (uid 0) to bin (uid 2). These changes happened after the control manifest was first created.Using the cron(1M) command, you can generate BART manifests every minute, hour, day, week, or month. How often you actually run BART to generate new manifests is based on how critical your need is to detect change. Because one size does not fit all, you might even want to consider having different BART rules or policies. For example, you might have two policies: A smaller policy, targeting a few key files, that runs every minute or hour A larger policy, collecting information across the entire system, that runs only once a day or once a weekUse common sense when deciding how often to generate manifests. You need to balance your detection priorities with the I/O load that the collection process generates on the system. For more information on automating the collection and processing of BART manifests, see the Sun BluePrint article titled “Automating Solaris 10 File Integrity Checks,” which can be found at http://www.sun.com/blueprints/0305/819-2259.pdf.2.3.3 Determining Whether a Filesystem Object Is GenuineRegardless of the selected time intervals, however, BART is still not able to definitively answer the question: “Is this a genuine object that Sun shipped?” BART cannot answer this question because you need to manually create the control manifest after the system has been installed. Someone or something could have changed a Sun-provided file from its default before you performed your first BART snapshot, particularly if the first BART manifest is generated well after installation. Remember that, if your control manifest is somehow corrupt, then all of the later comparisons against the original manifest are suspect.2.4 Signed ELF Filesystem ObjectsTo determine if a filesystem object is indeed genuine, you can query its cryptographic signature. Starting in the Solaris 10 OS, most operating system Executable and Linkable Format (ELF) objects have been cryptographically signed by Sun. The signatures on these files can be validated by using the elfsign(1) command.# elfsign verify -e /usr/bin/suelfsign: verification of /usr/bin/su passed.While this is a very strong way of validating the integrity of a file, this approach is only applicable to ELF objects such as binaries, libraries, device drivers, and other compiled code. Also, due to license or other restrictions, some binaries that are shipped in a Solaris release cannot be modified and therefore the signature cannot be added to those files. Last, text-based configuration and log files and other non-binary content are not protected in this manner. The Solaris Fingerprint Database, discussed in the next section, can validate files that are not signed.2.5 Solaris Fingerprint Database (sfpDB)The Solaris Fingerprint Database (sfpDB) is a free SunSolve Online service that enables organizations to verify the integrity of files that are distributed with the Solaris OS. Organizations can, at any time, query the sfpDB to determine whether an operating system file did indeed come from Sun.The sfpDB is a powerful tool for organizations that want to validate the integrity of their operating systems, baseline snapshots, and patches—or even to assist during digital forensic investigations in which the integrity of objects might be called into question.The Solaris Fingerprint Database is itself a collection of file fingerprints that have been created from the contents of Solaris OS media kits, unbundled software, and patches. These fingerprints are created as part of the release process in order to ensure that the

fingerprints are current and reflect actual shipping versions of files provided by Sun. By submitting fingerprints to the Solaris Fingerprint Database service, organizations can then determine which, if any, of the submitted fingerprints belongs to an actual file supplied by Sun.2.5.1 How the sfpDB Tool WorksThe Solaris Fingerprint Database is a collection of file fingerprints that are generated by using the MD5 Message Digest Algorithm, which is defined in RFC 1321:The algorithm takes as input a message of arbitrary length and produces as output a 128-bit “fingerprint” or “message digest” of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest.By using this algorithm, it is expected that no two different files will share the same fingerprint. While this assurance is not necessarily as strong as a cryptographic signature, it is virtually impossible to modify a file in such as way as to retain its original MD5 fingerprint. Note that, in September 2004, researchers announced that the MD5 algorithm was “broken.” That is, they had developed a method to produce two pre-images (for example, files) with the same MD5 hash value. While the announcement is important, this breakthrough does not impact the validity of the Solaris Fingerprint Database. As of the publication of this book, no feasible method exists to produce a pre-image (such as a text or binary file) that matches a given hash value. That is, no file could be replaced with another, different file and still retain its original MD5 hash value. For more information about the resistance of hash functions to attack, see, for example, http://www.schneier.com/crypto-gram-0409.html#3.To use the Solaris Fingerprint Database service, you calculate MD5 finger-prints for the files you want to check, and then submit them to the database service for processing. The sfpDB service evaluates the submitted fingerprints and determines if a match exists for any of the objects.For each fingerprint match, the information shown in Table 2.2 is provided.Table 2.2 Fingerprint Database FieldsNote that a single file fingerprint can result in several matches. Text files often generate several matches because they do not depend on the underlying operating system version or hardware platform. Examples of text files are configuration files and shell scripts.2.5.2 Creating an MD5 File FingerprintThe Solaris 10 OS contains a program that can be used to generate MD5 finger-prints: /usr/bin/digest. The following example uses the digest program to create an MD5 fingerprint:# digest -v -a md5 /usr/bin/sumd5 (/usr/bin/su) = bab18f089705c1628ccdc177b802d5712.5.3 Testing an MD5 File FingerprintOnce the MD5 fingerprints have been computed, they can be submitted to the Solaris Fingerprint Database to determine whether they correspond to values associated with valid Sun operating system files. To test the collected finger-prints:1. Visit the Solaris Fingerprint Database page at http://sunsolve.sun.com/pub-cgi/fileFingerprints.pl.After connecting to this URL, the Solaris Fingerprint Web form is displayed. Scroll to the bottom of the page to find the text window where the collected fingerprints can be entered.2. Type or copy and paste one or more MD5 fingerprints into the Web form. For example, to verify the su command fingerprint that was generated in the preceding example, paste the MD5 value into the Web form: bab18f089705c1628ccdc177b802d5713. Click the Submit button to view the results.For

this example, the following result is returned:The Solaris Fingerprint Database query properly identified the /usr/bin/su program as the object to which the fingerprint belonged. It is important to not only examine the name and canonical path when evaluating the results, as there are many possible outcomes such as multiple data mismatches, matches indicating an upgrade or downgrade attack, and of course, a failure where the fingerprint that you provided was not found in the database. Each of these cases, along with a more in-depth treatment of the Solaris Fingerprint Database, is provided in the Sun BluePrint article titled The Solaris Fingerprint Database: A Security Tool for Solaris Operating Environment Files (Updated for Solaris 10), which can be found at http://www.sun.com/blueprints/0306/816-1148.pdf.It is good practice to use together both of the tools discussed in this chapter: the Basic Audit and Reporting Tool and the Solaris Fingerprint Database. More information on this integration can be found in the Sun BluePrint article titled “Integrating BART and the Solaris Fingerprint Database in the Solaris 10 Operating System,” which can be found at http://www.sun.com/blueprints/0405/819-2260.pdf.References

What people say about this book

Michael Ernest, “Solid and helpful introduction to the subject. As I mentioned in my review of

Solaris 10 System Administration Essentials

, I bought the two books together. I read the other book first and was disappointed by the hand-waving treatment of important topics and sometimes outdated concepts. I also felt misled by the author attribution. While it was my mistake to infer Sun engineers had written the book, "Solaris System Engineers" still misrepresents many of that book's contributors. If I had written that content and dared call myself a system engineer, I'd imagine my employer would want to know what I mean by that phrase, followed by a quiet resignation.Thankfully, this book is quite different. It is, for starters, attributed to "Sun Microsystems Security Engineers" and there's no mistaking that by the quality of the content. The first two chapters present exactly what I think such a volume should include, namely, a motivation for the subject and an example-driven overview of all the aspects of security that enter into administering this operatiing system. These chapters are simple, clear, and set a solid foundation that makes the sometimes-dry content to follow easier to digest.The chapters that follow could be longer, but I think the content mostly delivers on the "essentials" idea. The treatments aren't comprehensive, but they're enough to get started in several areas and explore what the Solaris tools make possible. Chapter 3, "System Protection with SMF," highlights a few aspects of Solaris' Service Management Facility that are often overlooked, and provides useful examples the reader can work through. I also liked Chapter 5, "Privileges and Role-Based Access Control," despite the 4-5 unnecessary screen shots of Solaris Management Console. This topic often gets mired in half-finished explanations of the configuration components. Here the roles of these components are laid out in a deliberate, patient manner that makes it easier to connect the dots.Most software engineers are not great writers. The easiest thing for many of them to do, it seems, is to transliterate from design documents and/or code to an exposition of features and possibly how they are implemented. Some chapters follow that dull, dry pattern. A beginner, grateful for any kind of help, might not mind. But an editor who insists the writer dig deeper for the intended audience would have helped a lot here. A detail-heavy topic like Chapter 10, "Solaris 10 Network

Security," tends to suffer from too much show and not enough tell. The imbalance is strongest in Chapter 8, "Key Management Framework," which is all of nine pages -- closer to "skeletal" than "essential" but oh well. In fact, if you read in order, starting with Chapter 6, "Pluggable Authentication Modules (PAM)" and Chapter 7, "Solaris Cryptographic Framework," you can almost hear the breathing get more labored as you go. The going generally gets tougher to the end. The last chapter, "Configuring and Using Trusted Extensions," is welcome relief from that trend, but it would also benefit from a discussion that motivates the reader: is it potentially useful? How? If it's merely cool to a lot of users, that's ok to me, but please do tell me what fun I'm missing.Overall, the information provided in this book is strong and useful. It is certainly more accessible than a lot of existing documentation. On price: if I could save $15 or more by printing the PDF of the book, I would happily live without the binding and the nice cover. With that in mind, I wouldn't pay retail for this book, which I find a shade too thin."

Husto, "Solaris can be made harder. A very good starting place to further harden an obserdly hard operating system. For those that have an interest in IT security, this is another useful point of referrence -- Microsoft types waste not your time, this far and beyond what you all think of security."

Matthew A Phelps, "Fast shipping. Great price.. Great seller. Great quality product."

leonard S Culpepper, "Good book. Good book"

Trinity, "Essential. Just as it says on the cover, it's essential. I've had it for a couple days and now I can't live without it. It's answered so many nagging questions. Yay Solaris!"

The book by Margaret Miller has a rating of 5 out of 4.4. 10 people have provided feedback.

Title Page Copyright Page Contents Preface About the Authors Chapter 1. Solaris Security Services Chapter 2. Hardening Solaris Systems Chapter 3. System Protection with SMF Chapter 4. File System Security Chapter 5. Privileges and Role-Based Access Control Chapter 6. Pluggable Authentication Modules (PAM) Chapter 7. Solaris Cryptographic Framework Chapter 8. Key Management Framework (KMF) Chapter 9. Auditing Chapter 10. Solaris Network Security Chapter 11. Zones Virtualization Security Chapter 12. Configuring and Using Trusted Extensions Index

Book Information

Language: English
Paperback: 438 pages
Item Weight: 1.82 pounds
Dimensions: 7.5 x 1.05 x 9 inches
File size: 7616 KB
Text-to-Speech: Enabled
Enhanced typesetting: Enabled
X-Ray: Not Enabled
Word Wise: Not Enabled
Sticky notes: On Kindle Scribe
Print length: 392 pages
Screen Reader: Supported
Simultaneous device usage: Up to 5 simultaneous devices, per publisher limits