Solaris™ Security Toolkit 4.1 Reference Manual

Sun Microsystems, Inc.
www.sun.com

Part No. 817-7750-10
October 2004, Revision A

Submit comments about this document at: http://www.sun.com/hwdocs/feedback

Adobe PostScript

Contents

Tables

Code Samples

Preface

This *Solaris™ Security Toolkit 4.1 Reference Manual* contains reference information for understanding and using the internals of the Solaris Security Toolkit software. This manual is primarily intended for persons who use the Solaris Security Toolkit software to secure Solaris™ Operating System (OS) versions 2.5.1 through 9, such as administrators, consultants, and others, who are deploying new Sun systems or securing deployed systems. The instructions apply to using the software in either its JumpStart™ mode or standalone mode.

Following are terms used in this manual that are important to understand:

- **Hardening** – The modification of Solaris OS configurations to improve the security of a system.
- **Minimizing** – The removal of Solaris OS packages that are not needed on a particular system. (Because each system's requirements vary, what is deemed unnecessary also varies and has to be evaluated.) This removal reduces the number of components to be patched and made secure, which in turn reduces entry points available to a possible intruder.
- **Auditing** – The process of determining if a system's configuration is in compliance with a predefined security profile.
- **Scoring** – A score is a value associated with the number of failures uncovered during an audit run. So, if no failures (of any kind) are found, then the resulting score is `0`. The Solaris Security Toolkit increments the score (also known as a vulnerability value) by `1` whenever a failure is detected.

Before You Read This Book

You should be a Sun Certified System Administrator for Solaris™ or Sun Certified Network Administrator for Solaris™ Operating System. You should also have an understanding of standard network protocols and topologies.

Because this book is designed to be useful to people with varying degrees of experience or knowledge of security, your experience and knowledge determine how you use this book.

How This Book Is Organized

This manual contains reference information about the software components and is structured as follows:

Chapter 1 provides reference information for using, adding, modifying, and removing framework functions. Framework functions provide flexibility for you to change the behavior of the Solaris Security Toolkit software without modifying source code.

Chapter 2 provides reference information about how to use, modify, and customize the file templates included in the Solaris Security Toolkit software.

Chapter 3 provides reference information about using, adding, modifying, and removing drivers. This chapter describes the drivers used by the Solaris Security Toolkit software to harden, minimize, and audit Solaris OS systems.

Chapter 4 provides reference information about using, adding, modifying, and removing finish scripts. This chapter describes the scripts used by the Solaris Security Toolkit software to harden and minimize Solaris OS systems.

Chapter 5 provides reference information for using, adding, modifying, and removing audit scripts.

Chapter 6 provides reference information about using environment variables. This chapter describes all of the variables used by the Solaris Security Toolkit software and provides tips and techniques for customizing their values.

Using UNIX® Commands

This document might not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices. Refer to the following for this information:

- Software documentation that you received with your system
- Solaris Operating System documentation, which is at

  `http://docs.sun.com`

Shell Prompts

Shell	Prompt
C shell	*machine-name*%
C shell superuser	*machine-name*#
Bourne shell and Korn shell	$
Bourne shell and Korn shell superuser	#

Typographic Conventions

Typeface[1]	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123`	What you type, when contrasted with on-screen computer output	`% su` `Password:`
AaBbCc123	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this. To delete a file, type `rm` *filename*.

1 The settings on your browser might differ from these settings.

Accessing Sun Documentation

You can view, print, or purchase a broad selection of Sun documentation, including localized versions, at:

http://www.sun.com/documentation

Third-Party Web Sites

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Related Resources

Related publications and web sites are listed in this section.

Publications

- Andert, Donna, Wakefield, Robin, and Weise, Joel. "Trust Modeling for Security Architecture Development," Sun BluePrints™ OnLine, December 2002, `http://www.sun.com/blueprints/1202/817-0775.pdf`.
- Dasan, Vasanthan, Noordergraaf, Alex, and Ordica, Lou. "The Solaris Fingerprint Database - A Security Tool for Solaris Software and Files," Sun BluePrints OnLine, May 2001, `http://www.sun.com/blueprints/0501/Fingerprint.pdf`.
- Englund, Martin, "Securing Systems with Host-Based Firewalls - Implemented With SunScreen Lite 3.1 Software," Sun BluePrints OnLine, September 2001, `http://sun.com/blueprints/0901/sunscreenlite.pdf`.
- Garfinkel, Simon, and Spafford, Gene. *Practical UNIX and Internet Security*, 2nd Edition, O'Reilly & Associates, April 1996.
- Howard, John S., and Noordergraaf, Alex. *JumpStart Technology: Effective Use in the Solaris Operating Environment*, The Official Sun Microsystems Resource Series, Prentice Hall, October 2001.
- Moffat, Darren J., FOCUS on SUN: *Solaris BSM Auditing*, `http://www.securityfocus.com/infocus/1362`
- Noordergraaf, Alex. "Solaris™ Operating Environment Minimization for Security: A Simple, Reproducible and Secure Application Installation Methodology Updated for Solaris 8 Operating Environment," Sun BluePrints OnLine, November 2000, `http://sun.com/blueprints/1100/minimize-updt1.pdf`.
- Noordergraaf, Alex. "Minimizing the Solaris Operating Environment for Security: Updated for Solaris 9 Operating Environment," Sun BluePrints OnLine, November 2002,`http://sun.com/blueprints/1102/816-5241.pdf`.
- Noordergraaf, Alex. "Securing the Sun Cluster 3.x Software," Sun BluePrints OnLine article, February 2003, `http://www.sun.com/solutions/blueprints/0203/817-1079.pdf`.
- Noordergraaf, Alex, "Securing the Sun Enterprise 10000 System Service Processors," Sun BluePrints OnLine article, March 2002, `http://www.sun.com/blueprints/0302/securingenter.pdf`
- Noordergraaf, Alex, et. al. *Enterprise Security: Solaris Operating Environment Security Journal, Solaris Operating Environment Versions 2.5.1, 2.6, 7, and 8*, Sun Microsystems™, Prentice Hall Press, ISBN 0-13-100092-6, June 2002.

- Noordergraaf, Alex and Nimeh, Dina. "Securing the Sun Fire 12K and 15K Domains," Sun BluePrints OnLine article, February 2003, http://www.sun.com/blueprints/0203/817-1357.pdf.
- Noordergraaf, Alex and Nimeh, Dina. "Securing the Sun Fire 12K and 15K System Controllers," Sun BluePrints OnLine article, February 2003, http://www.sun.com/blueprints/0203/817-1358.pdf.
- Noordergraaf, Alex and Watson, Keith. "Solaris Operating Environment Security: Updated for the Solaris 9 Operating Environment," Sun BluePrints OnLine, December 2002, http://www.sun.com/blueprints/1202/816-5242.pdf.
- Osser, William and Noordergraaf, Alex. "Auditing in the Solaris 8 Operating Environment," Sun BluePrints OnLine, February 2001 http://www.sun.com/blueprints/0201/audit_config.pdf.
- Reid, Jason M. and Watson, Keith. "Building and Deploying OpenSSH in the Solaris Operating Environment," Sun BluePrints OnLine, July 2001, http://sun.com/blueprints/0701/openSSH.pdf.
- Reid, Jason M. "Configuring OpenSSH for the Solaris Operating Environment," Sun BluePrints OnLine article, January 2002, http://www.sun.com/blueprints/0102/configssh.pdf.
- Reid, Jason. *Secure Shell in the Enterprise*, The Official Sun Microsystems Resource Series, Prentice Hall, June 2003
- *Solaris Advanced Installation Guide*, Sun Microsystems, http://docs.sun.com.
- *SunSHIELD Basic Security Module Guide*, Sun Microsystems, Inc., http://docs.sun.com.
- Watson, Keith and Noordergraaf, Alex. "Solaris Operating Environment Network Settings for Security: Updated for Solaris 9 Operating Environment," Sun BluePrints OnLine, June 2003, http://www.sun.com/solutions/blueprints/0603/816-5240.pdf.
- Weise, Joel, and Martin, Charles R. "Developing a Security Policy," Sun BluePrints OnLine article, December 2001, http://www.sun.com/solutions/blueprints/1201/secpolicy.pdf.

Web Sites

- AUSCERT, *UNIX Security Checklist*, http://www.auscert.org.au/render.html?it=1935&cid=1920
- CERT/CC at http://www.cert.org is a federally funded research and development center working with computer security issues.
- Chkrootkit, http://www.chkrootkit.org
- Galvin, Peter Baer, *The Solaris Security FAQ*, http://www.itworld.com/Comp/2377/security-faq/

- HoneyNet Project, "Know Your Enemy: Motives" `http://project.honeynet.org/papers/motives/`
- List open files software, `ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/`
- Nmap Port Scanner, `http://www.insecure.org`
- OpenSSH tool, `http://www.openssh.com/`
- Pomeranz, Hal, *Solaris Security Step by Step*, `http://www.sans.org/`
- Rhoads, Jason, *Solaris Security Guide*, `http://www.sabernet.net/papers/Solaris.html`
- Security Focus at `http://www.securityfocus.org` is a web site dedicated to discussing pertinent security topics.
- Sendmail Consortium, `sendmail` configuration information, `http://www.sendmail.org/`
- Spitzner, Lance, *Armoring Solaris*, `http://secinf.net/unix_security/Armoring_Solaris.html`
- SSH Communications Security, Secure Shell (SSH) tool, `http://www.ssh.com/`
- Sun BluePrints OnLine, `http://sun.com/blueprints`
- Sun BluePrints OnLine Tools for FixModes software and MD5 scripts, `http://jsecom15k.sun.com/ECom/EComActionServlet?StoreId=8&PartDetailId=817-0074-10&TransactionId=try&LMLoadBalanced=`
- Sun Enterprise Authentication Mechanism™ information, `http://www.sun.com/software/solaris/ds/ds-seam`
- SunSolve℠ – `http://sunsolve.sun.com`

Running Supported Solaris OS Versions

Sun support for Solaris Security Toolkit software is available only for its use in the Solaris 8 and Solaris 9 Operating Systems. While the software can be used in the Solaris 2.5.1, Solaris 2.6 and Solaris 7 Operating Systems, Sun support is not available for its use in those operating systems.

The Solaris Security Toolkit software automatically detects which version of the Solaris Operating System software is installed, then runs tasks appropriate for that operating system version.

Running Supported SMS Versions

If you are using System Management Services (SMS) to run your system controller (SC), Sun support is available for Solaris Security Toolkit 4.1 software if you are using SMS version 1.3 through 1.4.1.

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in this document, go to:

http://www.sun.com/service/contacting

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

http://www.sun.com/hwdocs/feedback

Please include the title and part number of your document with your feedback:

Solaris Security Toolkit 4.1 Reference Manual, part number 817-7750-10

CHAPTER 1

Framework Functions

This chapter provides reference information on using, adding, modifying, and removing framework functions. Framework functions provide flexibility for you to change the behavior of the Solaris Security Toolkit software without modifying source code.

Use framework functions to limit the amount of coding that is needed to develop new finish and audit scripts, and to keep common functionality consistent. For example, by using the common logging functions, you can configure the reporting mechanism without needing to develop or alter any additional source code. Similarly, by using this modular code for common functions, bugs and enhancements can be more systematically addressed.

In addition, framework functions support the undo option. For example, using the framework function `backup_file` in place of a `cp` or `mv` command allows that operation to be reversed during an undo run.

This chapter contains the following topics:

Customizing Framework Functions

The Solaris Security Toolkit software is based on a modular framework that you can combine in various ways to suit your organization's needs. Sometimes, however, the standard features provided by the Solaris Security Toolkit software might not meet your site's needs. You can supplement the standard features by customizing framework functions to enhance and extend the functionality provided by the

Solaris Security Toolkit software. The framework functions configure how the Solaris Security Toolkit software runs, define the functions that it uses, and initialize environment variables.

In most cases, you can easily copy standard framework function files and scripts, and then customize the functionality for your use. For example, using the `user.run` file, you can add, modify, replace, or extend the standard framework functions. The `user.run` file is similar in purpose to the `user.init` file, except that you use the `user.init` file to add or modify environment variables.

In some cases, you might need to develop new framework functions. In this case, use similar framework functions as a guide or template for coding, and be sure to follow the recommendations provided in this book. Development should only be undertaken by users who are familiar with the Solaris Security Toolkit software's design and implementation.

Caution – Take extreme care when developing your own framework functions. Incorrect programming might compromise the Solaris Security Toolkit software's ability to properly implement or undo changes or to audit a system's configuration. Furthermore, changes made to the software could adversely impact the target platform on which the software is run.

CODE EXAMPLE 1-1 show how Solaris Security Toolkit functionality can be extended by customizing the standard framework. In this example, the `mount_filesystems` function is modified to enable the developer to mount additional file systems during a JumpStart installation. The `mount_filesystems` function is copied directly from the `driver.funcs` script into the `user.run` file. The modifications to it are in line numbers 8 and 9.

CODE EXAMPLE 1-1 Extending Functionality by Customizing the Framework

```
1   mount_filesystems()
2   {
3       if [ "${JASS_STANDALONE}" = "0" ]; then
4           mount_fs ${JASS_PACKAGE_MOUNT} ${JASS_ROOT_DIR} \
5               ${JASS_PACKAGE_DIR}
6           mount_fs ${JASS_PATCH_MOUNT} ${JASS_ROOT_DIR} \
7               ${JASS_PATCH_DIR}
8           mount_fs 192.168.0.1:/apps01/oracle \
9               ${JASS_ROOT_DIR}/tmp/apps-oracle
10      fi
11  }
```

For the sake of simplicity, the variable used to mount the new file system is not converted to Solaris Security Toolkit environment variables. To aid in portability and flexibility, abstract the actual values using environment variables. This approach

allows changes to be made consistently, because the software is deployed into environments with different requirements, such as production, quality assurance, and development.

Note – You could implement the same functionality within a finish script that uses this mount point, so that the mounting, use, and unmounting of the file system is self-contained within the script. However, it might be more effective and efficient to mount the file system using `mount_filesystems` when a single file system is used by more than one script.

Caution – A disadvantage to modifying `mount_filesystems` is that when you install updates of the Solaris Security Toolkit software, you might need to modify the `mount_filesystems` again.

Using Common Log Functions

These functions control all logging and reporting functions and are located in the `Drivers` directory in a file called `common_log.funcs`. The logging and reporting functions are used in all of the Solaris Security Toolkit software's operational modes; therefore, they are considered common functions. For example, common functions such as `logWarning` and `logError` are in this file.

This section describes the following common log functions.

logBanner

This function displays banner messages. These messages typically precede driver, finish, or audit script run output. Also, banner messages are used at the start and end of a run and are only displayed if the logging verbosity is at least 3. For more information on verbosity levels, see Chapter 6.

Banner messages take two forms. If you pass an empty string to this function, then a single line separator is displayed. This line is often used to force a "break" in the displayed output. If you enter a single string value, then the output is displayed between a pair of single line separators. CODE EXAMPLE 1-2 shows a sample of a banner message.

CODE EXAMPLE 1-2 Sample Banner Message

```
================================================================
Solaris Security Toolkit Version: 4.1
Node name:                        imbulu
Host ID:                          8085816e
Host address:                     192.168.0.1
MAC address:                      0:0:80:85:81:6e
OS version:                       5.9
Date:                             Wed Jan  1 22:27:15 EST 2003
================================================================
```

You can control banner messages through the JASS_LOG_BANNER environment variable. For more information on this environment variable, see Chapter 6.

logDebug

This function displays debugging messages. This function accepts a single string argument to be displayed as a debugging message. By default, no debugging messages are displayed by the Solaris Security Toolkit software. This functionality is

for future use and for you to add debugging log messages to your code. Debugging messages are only displayed if the verbosity is at least 4. For more information about verbosity levels, see Chapter 6.

logError

This function displays error messages. This function accepts a single string value that is displayed as an error message. Error messages are those that contain the string [ERR].

Example usage:

```
logError "getScore: Score value is not defined."
```

Example output:

```
[ERR ] getScore: Score value is not defined.
```

You can control error messages through the JASS_LOG_ERROR environment variable. For more information on this environment variable, see Chapter 6.

logFailure

This function displays failure messages. This function accepts a single string value that is displayed as a failure message. Failure messages are those that contain the string [FAIL].

Example usage:

```
logFailure "Package SUNWatfsr is installed."
```

Example output:

```
[FAIL] Package SUNWatfsr is installed.
```

You can control failure messages through the JASS_LOG_FAILURE environment variable. For more information on this environment variable, see Chapter 6.

logFileContentsExist and logFileContentsNotExist

Use these functions to log messages associated with the success or failure of checks. These functions together report on the results of a file content check. These functions are used primarily with the `check_fileContentsExist` and `check_fileContentsNotExist` functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the file to test
- String value representing the search pattern
- Non-negative integer representing the vulnerability value result
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional)

Example usage:

```
logFileContentsExist /etc/default/inetinit "TCP_STRONG_ISS=2" 0
```

Example output:

```
[PASS] File /etc/default/inetinit has content matching

TCP_STRONG_ISS=2.
```

These functions display either success or failure messages. You can control these messages through the `JASS_LOG_FAILURE` and `JASS_LOG_SUCCESS` environment variables. For more information on these environment variables, see Chapter 6.

logFileExists and logFileNotExists

Use these functions to log messages associated with the success or failure of a check. These functions together report on the results of a file check. These functions are primarily used with the `check_fileExists` and `check_fileNotExists` functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the file to test

- Non-negative integer representing the vulnerability value result

  If this argument is passed a null string value, then this function reports the result in the form of a notice using the `logNotice` function. Otherwise, it reports the result as a failure using the `logFailure` function.
- String value representing related information that you want displayed for users after a `PASS`, `FAIL`, or `NOTE` message (optional)

Example usage:

```
logFileExists /etc/issue
```

Example output:

```
[NOTE] File /etc/issue was found.
```

These functions display either success or failure messages. You can control these messages through the `JASS_LOG_FAILURE` and `JASS_LOG_SUCCESS` environment variables. For more information on these environment variables, see Chapter 6.

`logFileGroupMatch` and `logFileGroupNoMatch`

Use these functions to log messages associated with the success or failure of a check. These functions together report on the results of a file group membership check. These functions are used primarily with the `check_fileGroupMatch` and `check_fileGroupNoMatch` functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the file to test
- String value representing the group to check
- Non-negative integer representing the vulnerability value result
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional)

Example usage:

```
logFileGroupMatch /etc/motd sys 0
```

Example output:

```
[PASS] File /etc/motd has group sys.
```

These functions display either success or failure messages. You can control these messages through the JASS_LOG_FAILURE and JASS_LOG_SUCCESS environment variables. For more information on these environment variables, see Chapter 6.

logFileModeMatch and logFileModeNoMatch

Use these functions to log messages associated with the success or failure of a check. These functions together report on the results of a file permissions check. These functions are used primarily with the check_fileModeMatch and check_fileModeNoMatch functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the file to test
- String value representing the permissions to check
- Non-negative integer representing the vulnerability value result
- String value representing related information that you want displayed for users after a PASS or FAIL message (optional)

Example usage:

```
logFileModeMatch /etc/motd 0644 0
```

Example output:

```
[PASS] File /etc/motd has mode 0644.
```

These functions display either success or failure messages. You can control these messages through the JASS_LOG_FAILURE and JASS_LOG_SUCCESS environment variables. For more information on these environment variables, see Chapter 6.

logFileNotFound

This function is used by the software to display file not found messages. This function is used throughout the Solaris Security Toolkit code in both hardening and audit runs to provide a standard message when a designated file was not found on the system.

You can supply the following arguments to this function:

- String value representing the name of the file to test
- Non-negative integer representing the vulnerability value result

  If this argument is passed a null string value, then this function reports the result in the form of a notice using the `logNotice` function. Otherwise, it reports the result as a failure using the `logFailure` function.
- String value representing related information that you want displayed for users after a `FAIL` or `NOTE` message (optional)

Example usage:

```
logFileNotFound /etc/motd
```

Example output:

```
[NOTE] File /etc/issue was not found.
```

You can control notice and failure messages through the `JASS_LOG_NOTICE` and `JASS_LOG_FAILURE` environment variables, respectively. For more information on this environment variable, see Chapter 6.

logFileOwnerMatch and logFileOwnerNoMatch

Use these functions to log the messages associated with the success or failure of a check. These functions report on the results of a file ownership check. These functions are used primarily with the `check_fileOwnerMatch` and `check_fileOwnerNoMatch` functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the file to test
- String value representing the ownership to check

- Non-negative integer representing the vulnerability value result
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional)

Example usage:

```
logFileOwnerMatch /etc/motd root 0
```

Example output:

```
[PASS] File /etc/motd has owner root.
```

These functions display either success or failure messages. You can control these messages through the `JASS_LOG_FAILURE` and `JASS_LOG_SUCCESS` environment variables. For more information on these environment variables, see Chapter 6.

`logFileTypeMatch` and `logFileTypeNoMatch`

Use these functions to log the messages associated with the success or failure of a check. These functions report on the results of a file type check. These functions are used primarily with the `check_fileTypeMatch` and `check_fileTypeNoMatch` functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the file to test
- String value representing the file type to check

TABLE 1-1 lists the types detected by the software:

TABLE 1-1 File Types Detected by Using the `check_fileTemplate` Function

File Type	Description
`b`	Block special file
`c`	Character special file
`d`	Directory
`D`	Door
`f`	Regular file

TABLE 1-1 File Types Detected by Using the `check_fileTemplate` Function *(Continued)*

File Type	Description
`l`	Symbolic link
`p`	Named pipe (fifo)
`s`	Socket

- Non-negative integer representing the vulnerability value result
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional)

Example usage:

```
logFileTypeMatch /etc/motd f 0
```

Example output:

```
[PASS] File /etc/motd is a regular file.
```

These functions display either success or failure messages. You can control these messages through the `JASS_LOG_FAILURE` and `JASS_LOG_SUCCESS` environment variables. For more information on these environment variables, see Chapter 6.

logFinding

This function displays audit finding messages. This function accepts a single string argument to be displayed as a message. The input for this function is processed by the `printPrettyPath` function prior to display. In addition, if the verbosity level is equal to 2, then optional tags are prepended to the message. The following are the optional tags that you can prepend by this function:

- Timestamp – If the `JASS_DISPLAY_TIMESTAMP` environment variable is `1`, then the timestamp as defined by the `JASS_TIMESTAMP` environment variable prepends to the finding message.
- Target Host Name – If the `JASS_DISPLAY_HOSTNAME` environment variable is `1`, then the target's host name as defined by the `JASS_HOSTNAME` environment variable prepends to the finding message.
- Current Script Name – If the `JASS_DISPLAY_SCRIPTNAME` environment variable is `1`, then the name of the current audit script prepends to the finding message.

Note – If the finding occurs outside of an audit script, such as within the flow of the `driver.run` script, then the name of the current driver is used.

You can use all three output tags collectively or independently. The order of the position in the resulting output line is as they are listed. For more information on this function and verbosity levels, see Chapter 6.

Example usage:

```
logFinding "/etc/motd"
```

Example output:

```
test-script /etc/motd
```

logFormattedMessage

Use this function to generate formatted audit script headers that display information such as the script name, purpose, and rationale for the check. This function accepts a single string value and formats the message that is passed to the function.

These messages are reformatted as follows:

- Maximum width of 75 characters
- Prepended with the string " # " (pound symbol with a space before and after it)
- Duplicate slashes in path names are removed

Formatted messages are displayed only when the verbosity level is at least `3`. For more information on this function and verbosity levels, see Chapter 6.

Example usage:

```
logFormattedMessage "Check system controller secure shell
configuration."
```

Example output:

```
# Check system controller secure shell configuration.
```

logInvalidDisableMode

Use this function to display an error message when the JASS_DISABLE_MODE environment variable is set to an invalid value. This utility function reports on the state of the JASS_DISABLE_MODE environment variable. This function takes no arguments and generates the following output:

```
[ERR ] The JASS_DISABLE_MODE parameter has an invalid value: [...]

[ERR ] value must either be "script" or "conf".
```

For more information on this environment variable, see Chapter 6.

logInvalidOSRevision

Use this function when either the check_os_revision or check_os_min_revision functions fail their checks. This utility function reports when a function is being called on a version of the Solaris OS for which it does not apply. For example, use this function when there is an attempt to use a Solaris 8 OS script with the Solaris 2.6 OS.

Example usage:

```
logInvalidOSRevision "5.9"
```

Example output:

```
[NOTE] This script is only applicable for Solaris version

5.9.
```

To specify multiple versions, enter a hyphen (-) between versions, for example, "5.6-5.8."

This function displays notice messages. You can control messages through the JASS_LOG_NOTICE environment variable. For more information on this environment variable, see Chapter 6.

logMessage

Use this function to display any message that you want to display to users. Use this function for messages that do not have any tags associated with them. Similar to the `logFormattedMessage` function, this function displays an unformatted message. This function accepts a single string value that is displayed as is, with no modification.

Unformatted messages are only displayed if the verbosity level is at least 3. For more information on this function and verbosity levels, see Chapter 6.

Example usage:

```
logMessage "Verify system controller static ARP configuration."
```

Example output:

```
Verify system controller static ARP configuration.
```

logNotice

Use this function to display notice messages. This function accepts a single string value that is displayed as a notice message. Notice messages are those that contain the string `[NOTE]`.

Example usage:

```
logNotice "Service ${svc} does not exist in ${INETD}."
```

Example output:

```
[NOTE] Service telnet does not exist in /etc/inetd.conf.
```

You can control notice messages through the `JASS_LOG_NOTICE` environment variable. For more information on this environment variable, see Chapter 6.

logPackageExists and logPackageNotExists

Use these functions to log the messages associated with the success or failure of a check. These functions report on the results of a check that determines if a software package is installed. These functions are used primarily with the `check_packageExists` and `check_packageNotExists` functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the software package to test
- Non-negative integer representing the vulnerability value result
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional)

Example usage:

```
logPackageExists SUNWcsr 0
```

Example output:

```
[PASS] Package SUNWcsr is installed.
```

These functions display either success or failure messages. You can control these messages through the `JASS_LOG_FAILURE` and `JASS_LOG_SUCCESS` environment variables. For more information on these environment variables, see Chapter 6.

logPatchExists and logPatchNotExists

Use these functions to log the messages associated with the success or failure of a check. These functions report on the results of a check that determines if a software patch is installed. These functions are used primarily with the `check_patchExists` and `check_patchNotExists` functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the patch identifier (number) to test
- Non-negative integer representing the vulnerability value result

- String value representing related information that you want displayed for users after a PASS or FAIL message (optional)

Example usage:

```
logPatchExists 123456-01 0
```

Example output:

```
[PASS] Patch ID 123456-01 or higher is installed.
```

These functions display either success or failure messages. You can control these messages through the JASS_LOG_FAILURE and JASS_LOG_SUCCESS environment variables. For more information on these environment variables, see Chapter 6.

logProcessArgsMatch and logProcessArgsNoMatch

Use these functions to log the messages associated with the success or failure of a check. These functions report on the results of a check for runtime process arguments. These functions are used primarily with the check_processArgsMatch and check_processArgsNoMatch functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the process to test
- String value representing the argument search pattern
- Non-negative integer representing the vulnerability value result
- String value representing related information that you want displayed for users after a PASS or FAIL message (optional)

Example usage:

```
logProcessArgsMatch inetd "-t" 0
```

Example output:

```
[PASS] Process inetd found with argument -t.
```

These functions display either success or failure messages. You can control these messages through the JASS_LOG_FAILURE and JASS_LOG_SUCCESS environment variables. For more information on these environment variables, see Chapter 6.

logProcessExists and logProcessNotExists

Use these functions to log the messages associated with the success or failure of a check. These functions report on the results of a check for a process. These functions are used primarily with the check_processExists and check_processNotExists functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the process to test
- Non-negative integer representing the vulnerability value result
- String value representing related information that you want displayed for users after a PASS or FAIL message (optional)

Example usage:

```
logProcessExists nfsd 0
```

Example output:

```
[PASS] Process nfsd was found.
```

These functions display either success or failure messages. You can control these messages through the JASS_LOG_FAILURE and JASS_LOG_SUCCESS environment variables. For more information on these environment variables, see Chapter 6.

logProcessNotFound

Use this function to log a FAIL message for any process that is not found. This function displays "process not found" messages. This function provides a standard message when a designated process cannot be found on a system.

You can supply the following arguments to this function:

- String value representing the name of the process to test

- String value representing related information that you want displayed for users after a PASS or FAIL message (optional)

Example usage:

```
logProcessNotFound inetd
```

Example output:

```
[FAIL] Process inetd was not found.
```

You can control these messages through the JASS_LOG_FAILURE environment variable. For more information on these environment variables, see Chapter 6.

logServiceConfigExists and logServiceConfigNotExists

Use these functions to log the messages associated with the success or failure of a check. These functions report on the results of a check that determines if a configuration file exists. These functions are used primarily with the check_serviceConfigExists and check_serviceConfigNotExists functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the service configuration file to test
- Non-negative integer representing the vulnerability value result
- String value representing related information that you want displayed for users after a PASS or FAIL message (optional)

Example usage:

```
logServiceConfigExists /etc/apache/httpd.conf 0
```

Example output:

```
[PASS] Service Config File /etc/apache/httpd.conf was found.
```

These functions display either success or failure messages. You can control these messages through the JASS_LOG_FAILURE and JASS_LOG_SUCCESS environment variables. For more information on these environment variables, see Chapter 6.

logStartScriptExists and logStartScriptNotExists

Use these functions to log the messages associated with the success or failure of a check. These functions report on the results of a check that determines if a run-control start script exists. These functions are used primarily with the `check_startScriptExists` and `check_startScriptNotExists` functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the start script to test
- Non-negative integer representing the vulnerability value result
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional)

Example usage:

```
logStartScriptExists /etc/rc3.d/S89sshd 0
```

Example output:

```
[PASS] Start Script /etc/rc3.d/S89sshd was found.
```

These functions display either success or failure messages. You can control these messages through the `JASS_LOG_FAILURE` and `JASS_LOG_SUCCESS` environment variables. For more information on these environment variables, see Chapter 6.

logStopScriptExists and logStopScriptNotExists

Use these functions to log the messages associated with the success or failure of a check. These functions report on the results of a check that determines if a run-control stop script exists. These functions are used primarily with the `check_stopScriptExists` and `check_stopScriptNotExists` functions, although they can be used independently if necessary.

You can supply the following arguments to this function:

- String value representing the name of the stop script to test
- Non-negative integer representing the vulnerability value result

- String value representing related information that you want displayed for users after a PASS or FAIL message (optional)

Example usage:

```
logStopScriptExists /etc/rc2.d/K03sshd 0
```

Example output:

```
[PASS] Stop Script /etc/rc2.d/K03sshd was found.
```

These functions display either success or failure messages. You can control these messages through the JASS_LOG_FAILURE and JASS_LOG_SUCCESS environment variables. For more information on these environment variables, see Chapter 6.

logSuccess

Use this function to display success messages. This function accepts a single string value that is displayed as an audit success message. Success messages are those that contain the string "[PASS]."

Example usage:

```
logSuccess "Package SUNWsshdr is installed."
```

Example output:

```
[PASS] Package SUNWsshdr is installed.
```

You can control success messages through the JASS_LOG_SUCCESS environment variable. For more information on this environment variable, see Chapter 6.

logWarning

Use this function to display warning messages. This function accepts a single sting value that is displayed as a warning message. Warning messages are those that contain the string "[WARN]."

Example usage:

```
logWarning "User ${acct} is not listed in ${JASS_PASSWD}."
```

Example output:

```
[WARN] User abc is not listed in /etc/passwd.
```

You can control warning messages through the `JASS_LOG_WARNING` environment variable. For more information on this environment variable, see Chapter 6.

Using Common Miscellaneous Functions

These functions are for common miscellaneous functions that are used within several areas of the Solaris Security Toolkit software and are not specific to functionality provided by other framework functions (files ending with a `.func` suffix). These functions are in the `Drivers` directory in a file called `common_misc.funcs`. Common utility functions such as `isNumeric` and `printPretty` are in this file.

This section describes the common miscellaneous functions.

isNumeric

Use this function to determine if input arguments are positive numbers. It is used throughout the software by helper functions whenever input must be validated to ensure that it consists of a single, positive integer. This function accepts a single string argument and determines if the value is a positive integer. If the value is a positive integer, this function displays a value of 0, otherwise it displays a value of 1.

invalidVulnVal

Use this function to determine if input arguments are positive numbers. This function accepts a single string argument and determines if the value is a positive integer. It logs an error message for each failure. This function is necessary to determine where there may be invalid arguments supplied to a function as a vulnerability value. In all other aspects, this function behaves like its `isNumeric` counterpart. This function applies only to audit operations.

checkLogStatus

Use this function to determine if a given input string is set to `LOG`. This function accepts a single string argument and determines whether the calling function should log its results. If the string evaluated is the string `LOG`, then this function echoes a value of 1, indicating that the calling function should log its results. If the input string contains any other value, this function echoes a value of 0, indicating that no output is logged by the calling function. This function applies only to audit operations.

adjustScore

Use this function to adjust the score outside of the methods provided by the functions defined in the `audit_public.funcs` file. This function accepts a positive integer argument representing the value that is added to the current score for an audit script. For example, there might be times when only the audit script can determine a failure. In those cases, use this function to adjust the score, accounting for the failure. If one is not supplied, it logs an error message and does not adjust the score. This function applies only to audit runs.

printPretty

Use this function to format printed output so that it is easier to read. This function accepts an unformatted input string and processes it. The resulting string is wrapped at 72 characters with each line of output indented by three characters.

printPrettyPath

Use this function to format path names. This function accepts as input an unformatted path name. This function strips any redundant forward slashes from the input string, then displays the result. If the string is empty, then the keyword "<No Value>" is displayed in its place.

extractComments

Use this function to remove comments from a file or script. This function accepts as input a list of tokens (script names, file names, and so on) and removes any text that is commented out. This function defines a comment as any substring of text that begins with a "#" (number) symbol and continues to the end of the line.

clean_path

Use this function to remove redundant "/" (forward slash) characters from a file name. This function accepts as input a single string argument and returns the same value after any duplicate forward slash characters (/) have been removed from the string. This function is used to clean up path names before they are displayed to the user or before they are placed in logs.

strip_path

Use this function to remove the JASS_ROOT_DIR prefix from the file name. This function accepts as input a single string argument and returns the same value after removing the JASS_ROOT_DIR prefix and replacing it with a single "/" (forward slash) character. This function is used with the add_to_manifest function when storing path names in the JASS manifest file.

Using Driver Functions

These functions are for driver functionality. These functions are in the `driver.funcs` file, located in the `Drivers` directory. Functions such as `add_pkg` and `copy_a_file` are in this file.

When customizing or creating scripts, use the following functions to perform standard operations.

`add_patch`

Use this function to add Solaris OS patches to the system. By default, this function expects that the patches installed are located in the `JASS_PATCH_DIR` directory. TABLE 1-2 lists the options for this function.

TABLE 1-2 Options for `add_patch` Finish Script Function

Option	Description
`-o` *options*	Options to be passed on
`-M` *patchdir*	The fully qualified path to the source directory
patchlist	List of patches or name of file containing a list of patches to apply

Example usage:

```
add_patch 123456-01
```

```
add_patch -M ${JASS_PATCH_DIR}/OtherPatches patch_list.txt
```

add_pkg

Use this function to add Solaris OS packages to the system. By default, this function expects that the packages are located in the JASS_PACKAGE_DIR directory and that these packages are in one of the standard Sun formats, spooled directories or package stream files. This function automatically adds the necessary manifest entries to permit this operation to be reversed during an undo run. During an undo run, packages added using this function are removed from the system. TABLE 1-3 lists the options for this function.

TABLE 1-3 Options for add_pkg Function

Option	Description
-a *ask_file*	The pkgadd ask file name. By default, the pkgadd ask file, noask_pkgadd, is used if no other file is specified.
-d *src_loc*	The fully qualified path to the source package (streams or directory) to be installed
-o *options*	The pkgadd command options
package	The package to be installed

Example usage:

```
add_pkg ABCtest
```

```
add_pkg -d ${JASS_ROOT_DIR}/${JASS_PACKAGE_DIR}/SUNWjass.pkg SUNWjass
```

add_to_manifest

Use this function to manually insert entries into a manifest file during hardening runs without calling one of the helper functions. This approach is most often done when a command must be executed for the undo operation to complete. Use this option with care to protect the integrity of the system and the Solaris Security Toolkit repository.

Caution – Exercise extreme caution when using the `X` manifest option. The commands specified by this operation are executed during an undo run of the Solaris Security Toolkit as the `root` user. If you are not careful, you could cause data loss or render a target system unstable. For example, an `X` manifest entry of `rm -rf/` would delete the system's root partition during an undo run.

The `add_to_manifest` command uses the following syntax:

```
add_to_manifest operation src dst args
```

This command puts an entry in the `JASS_RUN_MANIFEST` file in `JASS_REPOSITORY/jass-manifest.txt`, which is critical to the ability to undo the changes made by a finish script.

Note – Not all of the operations used by the Solaris Security Toolkit support each of these arguments. Also, the meaning of the options for `src`, `dst`, and `args` can differ based on the operation selected, as discussed in TABLE 1-4.

The operations supported by the add_to_manifest function are listed in TABLE 1-4. This table includes a sample resulting manifest entry after each option.

TABLE 1-4 add_to_manifest Options and Sample Manifest Entries

Option	Description

C	Indicates a file was copied. In this case, the src and dst parameters represent the original and copied file names, respectively. No other arguments are used.
	`install-templates.fin /etc/syslog.conf /etc/ \`
	`syslog.conf.JASS.20020823230626`

D	Indicates a directory was created. In this case, the src parameter represents the name of the newly created directory. No other arguments are used. `disable-lp.fin /var/spool/cron/crontabs.JASS`

J	Indicates a new file was created on the system. This operation is used only when the file specified by the src parameter does not exist on the system. During an undo run, files tagged with this operation code are removed. This operation uses both the src and dst parameters to represent the original name of the file and its saved file name (which must include the JASS_SUFFIX).
	`disable-power-mgmt.fin /noautoshutdown \`
	`/noautoshutdown.JASS.20020823230629`

M	Indicates a file was moved. In this case, the src and dst parameters represent the original and moved file names, respectively. No other arguments are used.
	`disable-ldap-client.fin /etc/rcS.d/K41ldap.client \`
	`/etc/rcS.d/_K41ldap.client.JASS.20020823230628`

R	Indicates a file was removed from the system. In this case, the src parameter represents the name of the file that was removed. Files marked with this operation code cannot be restored using the Solaris Security Toolkit undo command.

S	Indicates a symbolic link was created. In this case, the src and dst parameters represent the source and target file names, respectively. During an undo run, the symbolic links for files tagged with this operation are removed from the system.
	`install-templates.fin ../init.d/nddconfig /etc/rc2.d/ \`
	`S70nddconfig`

X	Indicates a command was defined that should be run when the Solaris Security Toolkit processes a manifest entry that has this operation code. A special operation, this one is most often used to execute complex commands that go beyond the standard operations. For example, in the install-fix-modes.fin finish script, the following manifest entry is added to instruct the software to undo changes made by the Fix Modes program: `/opt/FixModes/fix-modes -u` This command instructs the software to run the fix-modes program with the -u option. Note that all commands processed by this operation code should be specified using an absolute path to the program.

backup_file

Use this function to back up an existing file system object. This function backs up the original file using a standard naming convention. The convention appends JASS_SUFFIX to the original file name. This function automatically adds the necessary manifest entries to permit this operation to be reversed during an undo run.

The JASS_SAVE_BACKUP variable specifies if the Solaris Security Toolkit software saves or does not save backup copies of files modified during a run. If this environment variable is set to 0, then this function does not save backup files on the system. If files are not saved, then the run cannot be reversed by using the undo command.

Example usage:

```
backup_file /etc/motd
```

check_os_min_version

Use this function to detect functionality that exists in multiple releases of the Solaris OS. This function takes only one argument, indicating the minimal OS release version. If the actual release of the OS on the target platform is greater than or equal to the argument, then the function returns 0, otherwise this function returns 1. If an error is encountered, then this function returns 255.

For example, this function can be used as shown in CODE EXAMPLE 1-3.

CODE EXAMPLE 1-3 Detecting Functionality That Exists in Multiple OS Releases

```
if check_os_min_revision 5.6 ; then
   if [ "${JASS_KILL_SCRIPT_DISABLE}" = "1" ]; then
      disable_rc_file ${JASS_ROOT_DIR}/etc/rcS.d K10dtlogin
      disable_rc_file ${JASS_ROOT_DIR}/etc/rc0.d K10dtlogin
      disable_rc_file ${JASS_ROOT_DIR}/etc/rc1.d K10dtlogin
   fi
   disable_rc_file ${JASS_ROOT_DIR}/etc/rc2.d S99dtlogin
else
   logInvalidOSRevision "5.6 and later"
fi
```

In this example, the Common Desktop Environment (CDE) was not included in the Solaris OS until version 2.6, and this script checks to ensure that the version is at least 5.6, before attempting to disable the run-control scripts listed.

check_os_revision

Use this function to check for a specific OS revision or range of values. This function can take either one or two arguments. If one argument is supplied, then the script returns 0 only if the target operating system revision is the same as the argument, otherwise it returns 1.

Similarly, if two arguments are provided, the target operating system revision must be between the two values inclusively for the result to be 0. In either case, if an error is encountered, this function returns a value of 255.

For example, this function can be used as shown in CODE EXAMPLE 1-4.

CODE EXAMPLE 1-4 Checking for a Specific OS Revision or Range

```
if check_os_revision 5.5.1 5.8; then
   if [ "${JASS_DISABLE_MODE}" = "conf" ]; then
      disable_conf_file ${JASS_ROOT_DIR}/etc asppp.cf
   elif [ "${JASS_DISABLE_MODE}" = "script" ]; then
      if [ "${JASS_KILL_SCRIPT_DISABLE}" = "1" ]; then
         disable_rc_file ${JASS_ROOT_DIR}/etc/rcS.d K50asppp
         disable_rc_file ${JASS_ROOT_DIR}/etc/rc0.d K47asppp
         disable_rc_file ${JASS_ROOT_DIR}/etc/rc0.d K50asppp
         disable_rc_file ${JASS_ROOT_DIR}/etc/rc1.d K47asppp
         disable_rc_file ${JASS_ROOT_DIR}/etc/rc1.d K50asppp
      fi
      disable_rc_file ${JASS_ROOT_DIR}/etc/rc2.d S47asppp
   fi
else
   logInvalidOSRevision "5.5.1-5.8"
fi
```

In this example, the script disables only its scripts or configuration files, based on the value of JASS_DISABLE_MODE, when the target OS revision is or falls between Solaris OS versions 2.5.1 and 8 inclusively.

checksum

Use this function to calculate the checksum for a file. This function takes a single string value that represents the file for which the checksum is being calculated. This function, which uses the Solaris `cksum` program to calculate the checksum, outputs a value in the format *checksum*:*number of octets*.

CODE EXAMPLE 1-5 Calculating the Checksum

```
checksum foobar
12345:23
```

copy_a_dir

Use this function to recursively copy the contents of a directory. This function takes two arguments, a source directory name and a destination directory name. This function copies the contents of the source directory to the directory specified by the destination parameter. This function creates the new directory if it does not already exist. This function automatically adds the necessary manifest entries to permit this operation to be reversed during an undo run.

Example usage:

```
copy_a_dir /tmp/test1 /tmp/test2
```

copy_a_file

Use this function to copy exactly one regular file. This function takes two arguments: a source file name and a destination file name. This function copies the contents of the source file to the file name specified by the destination parameter. This function automatically adds the necessary manifest entries to permit this operation to be reversed during an undo run.

Example usage:

```
copy_a_file /tmp/test-file-a /tmp/test-file-b
```

copy_a_symlink

Use this function to copy a symbolic link to the target platform. This function takes two arguments: a source link name and a destination file name. This function creates a new symbolic link based on the source link specified using the new file name passed as the destination parameter. This function automatically adds the necessary manifest entries to permit this operation to be reversed during an undo run.

Example usage:

```
copy_a_symlink /tmp/test-link-a /tmp/test-link-b
```

copy_files

Use this function to copy a set of file system objects from the `JASS_HOME_DIR/Files` directory tree to a target system. This function uses the appropriate copy functions listed previously to ensure that the changes made can be reversed during an undo run. This function is capable of copying regular files, directories, and symbolic links.

Example usage:

```
copy_files /etc/init.d/nddconfig
```

```
copy_files "/etc/init.d/nddconfig /etc/motd /etc/issue"
```

This function extends capability by permitting the selective copy of objects based on tags appended to their file names.

The files that are copied by this function are selected by the following criteria:

- `/some/fully/qualified/path/file.${HOST}`

  In this option, the software copies the object only if the name of the target platform matches the tag specified by `${HOST}`. This host environment variable uses the same naming format as the `JASS_HOSTNAME` environment variable.

  For example: `/etc/issue.jordan`

  If the software cannot find this file, the software continues to search for the general file (the option described next).

- `/some/fully/qualified/path/file+${OS}`

In this option, the software only copies the object if the OS revision of the target platform matches the tag specified by `${OS}`. The OS parameter uses the same naming format as the `JASS_OS_REVISION` environment variable. So, a file to be used only on the Solaris 8 OS is denoted as "`filename+5.8`".

For example: `/etc/issue+5.10`

If the software cannot find this file, the software looks for the file associated with the host name (the option described next).

- `/some/fully/qualified/path/file`

  In this option, the software copies the file to a target system.

  For example: `/etc/issue`

Note – When the file length/size is zero, the file is not copied to the system.

The order of precedence used to match a file is listed. For example, if a host-specific and general file both exist, the host-specific file is used if the name of a target system matches the host name defined by the host-specific file.

Note – The `copy_files` function silently ignores any objects listed that are not found in the `JASS_HOME_DIR/Files` directory tree.

---

create_a_file

Use this function to create an empty file on a target system. This function uses a combination of the `touch`, `chown`, and `chmod` commands to create an empty file with a specific owner, group, and set of permissions.

Note – This function does not adjust permissions or ownerships on a file that exists.

This function creates a file with specific permissions.

Example usage:

```
create_a_file -o guppy:staff -m 750 /usr/local/testing
```

In this example, a file called `testing` is created in the `/usr/local` directory, owned by `guppy` and group of `staff`, with permissions `750`. This function accepts the options listed in TABLE 1-5.

TABLE 1-5 `create_a_file` Command Options

Option	Valid Input
`[-o user[:group]]`	Follows syntax of `chown`(1) and accepts `user` and `user:group`
`[-m perms]`	Follows syntax of `chmod`(1) and accepts `perms`
`/some/fully/qualified/path/file`	The fully qualified path to the file

Example usage:

```
create_a_file /usr/local/testing
```

```
create_a_file -o root /usr/local/testing
```

```
create_a_file -o root:sys /usr/local/testing
```

```
create_a_file -o root -m 0750 /usr/local/testing
```

create_file_timestamp

Use this function to create a unique timestamp value for a given file and for all file backup operations. This function is useful for creating a backup of an already backed-up file when a unique suffix value is needed. The timestamp value created is in the same format as `JASS_TIMESTAMP`. The resulting timestamp value created by this function is stored in the `JASS_SUFFIX` environment variable. For more information, see Chapter 6, "`JASS_TIMESTAMP`" on page 181.

Example usage:

```
create_file_timestamp /usr/local/testing
```

disable_conf_file

Use this function to disable service configuration files. This function accepts two string values representing the directory name in which the file is located and the service configuration file name. This function disables the service configuration file by prepending a prefix of "_" (underscore) to the file name, thereby preventing its execution.

Example usage:

```
disable_conf_file /etc/dfs dfstab
```

This example renames a file from `/etc/dfs/dfstab` to `/etc/dfs/_dfstab.JASS.<timestamp>`. This function automatically adds the necessary manifest entries to permit this operation to be reversed during an undo run.

disable_file

Use this function to disable files that cannot be stored in their original directory. For example, the `/var/spool/cron/crontabs` directory contains individual user `crontab` files. If a disabled or backed-up copy of a `crontab` file were stored in the `crontabs` directory, then the `cron` service would indicate an error, because there would be no user name that matched the names of the disabled or backed-up files.

To address this issue, this function creates a mirror directory with a `.JASS` suffix within which to store any of the disabled files. For example, if the file to be disabled is located in the `/var/spool/cron/crontabs` directory, this function creates a `/var/spool/cron/crontabs.JASS` directory into which the disabled file is moved.

The file to be disabled, as with the other disable functions, has a suffix of `.JASS.<timestamp>`. The difference with this function is that the disabled file is not stored in the same directory as the original file.

Example usage:

```
disable_file /var/spool/cron/crontabs/uucp
```

In this example, the file `/var/spool/cron/crontabs/uucp` is moved to the `/var/spool/cron/crontabs.JASS` directory and renamed as `uucp.JASS.<timestamp>`. This function automatically adds the necessary manifest entries to permit this operation to be reversed during an undo run.

disable_rc_file

Use this function to disable the execution of a run-control file. This function accepts two string values representing the directory name in which the script is located and the run-control script name. This function disables the script by prepending a prefix of "_" (underscore) to the file name, thereby preventing its execution by run-control framework. To be executed, a script name must begin with either an `S` or a `K` depending on its purpose as a start or kill run-control script. In addition, a suffix of `.JASS.<timestamp>` is appended to the disabled file.

Example usage:

```
disable_rc_file /etc/rc2.d S71rpc
```

This example renames a file from `/etc/rc2.d/S71rpc` to `/etc/rc2.d/_S71rpc.JASS.<timestamp>`. This function automatically adds the necessary manifest entries to permit this operation to be reversed during an undo run.

is_patch_applied and is_patch_not_applied

Use these functions to determine if a patch is or is not applied to a system. These functions accept a single string value representing the patch number to check.

This value can be specified in one of two ways:

- You can specify the patch number as in "`123456`." These functions display a value of 0 if the patch is installed on a target system. If the patch is not installed, they display a value of 1.

  Example usage:

```
is_patch_applied 123456
```

- You can specify the patch number and revision number as in "`123456-03`." These functions display a value of 0 if the patch is on the system and has at a minimum the same revision as specified. If the patch is not on the system, a value of 1 is displayed. If the patch is installed, however, and its revision is not at least the value specified, then these functions display a value of 2.

Example usage:

```
is_patch_applied 123456-02
```

mkdir_dashp

Use this function to create a new directory on a target system. This function accepts a single string value representing the name of the directory to create. This function uses the `-p` option to `mkdir` so that no error is reported if the target directory exists. This function automatically adds the necessary manifest entries to permit this operation to be reversed during an undo run.

Example usage:

```
mkdir_dashp /usr/local
```

move_a_file

Use this function to move a file from one name to another. This function requires two entries: a source file name and a destination file name. This function moves, or renames, the source file to the file name specified by the destination parameter. This function automatically adds the necessary manifest entries to permit this operation to be reversed during an undo run.

Example usage:

```
move_a_file /tmp/test-file-a /tmp/test-file-b
```

rm_pkg

Use this function to remove Solaris OS packages from a system. The operations performed by this function are final and cannot be reversed during an undo run. The options for this function are listed in TABLE 1-6.

TABLE 1-6 rm_pkg Function Options

Option	Description
-a *ask_file*	The pkgrm ask file name. By default, the pkgrm ask file, noask_pkgrm, is used if no other file is specified.
-o *options*	The pkgrm command options
package	The package to be removed

Example usage:

```
rm_pkg SUNWadmr
```

Using Audit Functions

Two types of audit functions are in the software: private and public. The functions defined in the audit_private.funcs file are private and not for public use. Never use the private scripts defined in this file. Only use the public scripts defined in the audit_public.funcs file.

The public functions define audit functions used in audit scripts, which are located in JASS_AUDIT_DIR. Functions defined in this file are public and can be freely used in both standard and custom audit scripts. Note that in many cases, the functions defined in this file are stubs that call functions defined in the audit_private.funcs file. These stubs were implemented to allow users to code their scripts to these public interfaces without needing to care if the underlying code will be modified or enhanced in newer releases.

Use these functions as part of audit scripts to assess components of the system's stored and runtime configurations. The following functions are public interfaces to the Solaris Security Toolkit software's audit framework.

When customizing or creating audit scripts, use the following functions to perform standard operations.

- "check_fileContentsExist and check_fileContentsNotExist" on page 38

check_fileContentsExist and check_fileContentsNotExist

Use these functions to determine if a designated file has content matching a supplied search string. These functions search a designated file to match its content with a search string. The search string can be in the form of a regular expression. These functions display a 0 for success, 1 for failure, and 255 for error condition.

You can supply the following arguments to this function:

- String value representing the name of the file or files to test.
- String value representing the search pattern.
- Non-negative integer representing the vulnerability value to be used if the audit check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results are logged automatically by either the `log_FileContentsExist` or the `log_FileContentsNotExist` functions. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the environment variable is set to "`LOG`."

Example usage:

```
check_fileContentsExist /etc/default/inetinit "TCP_STRONG_ISS=2" 1 LOG
```

check_fileExists and check_fileNotExists

Use these functions to determine if a file exists on a target system. These functions display a status of 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the file or files to test.
- Non-negative integer representing the vulnerability value to be used if the audit check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the environment variable is set to `LOG`.

Example usage:

```
check_fileExists /etc/inet/inetd.conf 1 LOG
```

check_fileGroupMatch and check_fileGroupNoMatch

Use these functions to determine if a file belongs to a group on a target system. These functions display a status of 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the file or files to test.
- String value representing the group to check. The group value can be a name or a group identifier (GID). If a group name is numeric and does not appear in a name service table, it is taken as a GID.

- Non-negative integer representing the vulnerability value to be used if the audit check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the environment variable is set to `LOG`.

Example usage:

```
check_fileGroupMatch /etc/passwd sys 1 LOG

check_fileGroupMatch /etc/passwd 3 1 LOG
```

`check_fileModeMatch` and `check_fileModeNoMatch`

Use these functions to determine if a file has the permissions specified on a target system. These functions display a status of 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the file or files to test.
- String value representing the mode or permissions to check. The permissions value can be either a symbolic or octal value. This function accepts the same values for this environment variable as does the `find`(1) command's `perm` option.
- Non-negative integer representing the vulnerability value to be used if the audit check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the environment variable is set to `LOG`.

Example usage:

```
check_fileModeMatch /etc/passwd "0444" 1 LOG

check_fileModeMatch /etc/passwd "ugo=r" 1 LOG
```

check_fileOwnerMatch and check_fileOwnerNoMatch

Use these functions to determine if a file belongs to a specific user on a target system. These functions display a status of 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the file or files to test.
- String value representing the user to check. The user value can be either a name or a user identifier.
- Non-negative integer representing the vulnerability value to use if the audit check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied for this argument, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the above environment variable is set to `LOG`.

Example usage:

```
check_fileOwnerMatch /etc/passwd root 1 LOG

check_fileOwnerMatch /etc/passwd 0 1 LOG
```

check_fileTemplate

Use this function to determine if a file template defined by the Solaris Security Toolkit software matches its counterpart installed on a target system. For example, if you were to use this function to check the file template `/etc/motd`, this function would compare the contents of `JASS_FILES_DIR/etc/motd` with `/etc/motd` to

determine if they were the same. If they were identical, this function would display 0 for success, 1 for failure, or 255 for any error condition. If you specify more than one file, they all must pass to get a display code of 0.

You can supply the following arguments to this function:

- String value representing the name or a list of files separated by spaces (for example, `a b c`) to test.
- Non-negative integer representing the vulnerability value to be used if the check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the environment variable is set to `LOG`.

Example usage:

```
check_fileTemplate /etc/motd 1 LOG
```

`check_fileTypeMatch` and `check_fileTypeNoMatch`

Use these functions to determine if a file system object is a specific object type on a target system. These functions display a 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the file or files to test.
- String value representing the file type to check. For more information on available types, see "`logFileTypeMatch` and `logFileTypeNoMatch`" on page 10.

TABLE 1-7 lists the types detected by the software:

TABLE 1-7 File Types Detected by Using the `check_fileTemplate` Function

File Type	Description
`b`	Block special file
`c`	Character special file
`d`	Directory
`D`	Door
`f`	Regular file
`l`	Symbolic link
`p`	Named pipe (fifo)
`s`	Socket

- Non-negative integer representing the vulnerability value to be used if the check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used this information is simply passed to the logging function if the environment variable is set to `LOG`.

Example usage:

```
check_fileTypeMatch /etc/passwd f 1 LOG

check_fileTypeMatch /etc d 1 LOG
```

check_minimized

Use this function when a package check should only be performed on a minimized platform. This function is similar to the `check_packagesNotExist` function, except that its behavior is controlled by the `JASS_CHECK_MINIMIZED` environment variable. If a target system is not minimized, then the `JASS_CHECK_MINIMIZED` environment variable should be set to 0. In this case, this function does not perform any of its checks and simply displays a value of 0 with a notice indicating that a

check was not run. Otherwise, this function behaves exactly as the `check_packageNotExists` function and displays a 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the package or packages to test.
- Non-negative integer representing the vulnerability value to be used if the check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the above environment variable is set to `LOG`.

Example usage:

```
check_minimized SUNWatfsu 1 LOG
```

`check_packageExists` and `check_packageNotExists`

Use these functions to determine if a software package is installed on a target system. These functions display a 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the package or packages to test.
- Non-negative integer representing the vulnerability value to be used if the audit check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the environment variable is set to `LOG`.

Example usage:

```
check_packageExists SUNWsshdu 1 LOG
```

check_patchExists and check_patchNotExists

Use these functions to determine if a software patch is installed on a target system. These functions display a 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the patch or patches to test.
- Non-negative integer representing the vulnerability value to be used if the check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the environment variable is set to `LOG`.

Example usage:

```
check_patchExists 123456 1 LOG

check_patchExists 123456-01 1 LOG
```

Note – You can specify a patch revision. If you do, then any installed revision must be equal to or greater than the revision specified. If you do not specify a revision, then this function indicates success if any version of the patch is installed.

---

check_processArgsMatch and check_processArgsNoMatch

Use these functions to determine if a process is running on the system with specific runtime arguments. These functions display a 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the process or processes to test.
- String value representing the runtime arguments to check.
- Non-negative integer representing the vulnerability value to be used if the check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the above environment variable is set to `LOG`.

Example usage:

```
check_processArgsMatch /usr/sbin/syslogd "-t" 1 LOG
```

check_processExists and check_processNotExists

Use these functions to determine if a process is running on a target system. These functions display a 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the process or processes to test.
- Non-negative integer representing the vulnerability value to be used if the check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.

- String value representing related information that you want displayed for users after a PASS or FAIL message (optional). If used, this information is simply passed to the logging function if the above environment variable is set to LOG.

Example usage:

```
check_processExists sshd 1 LOG
```

check_serviceConfigExists and check_serviceConfigNotExists

Use these functions to determine if a service configuration file exists on a target system. These functions display a 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the service configuration file or files to test.
- Non-negative integer representing the vulnerability value to be used if the check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value LOG, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a PASS or FAIL message (optional). If used, this information is simply passed to the logging function if the above environment variable is set to LOG.

Example usage:

```
check_serviceConfigExists /etc/ssh/sshd_config 1 LOG
```

check_startScriptExists and check_startScriptNotExists

Use these functions to determine if a run-control start script exists on a target system. These functions display a 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the run-control start script or scripts to test.
- Non-negative integer representing the vulnerability value to be used if the check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the above environment variable is set to `LOG`.

Example usage:

```
check_startScriptExists /etc/rc3.d/S89sshd 1 LOG
```

check_stopScriptExists and check_stopScriptNotExists

Use these functions to determine if a run-control stop script exists on a target system. These functions display a 0 for success, 1 for failure, and 255 for any error condition.

You can supply the following arguments to this function:

- String value representing the name of the run-control stop script or scripts to test.
- Non-negative integer representing the vulnerability value to be used if the check fails.
- String value representing the logging status of the function. If this value is equivalent to the string value `LOG`, then the results of this function are logged automatically. If any other string keyword is supplied, logging is not automatic, and the calling program code has to log any status messages.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is simply passed to the logging function if the above environment variable is set to `LOG`.

Example usage:

```
check_stopScriptExists /etc/rc2.d/K03sshd 1 LOG
```

finish_audit

Use this function to signal that a check script has completed all of its processing and that a score for the script must be computed. This function is typically the last entry in a check script. If you want to display a message indicating a script's termination, then pass a single string argument to this function.

Example usage:

```
finish_audit
```

```
finish_audit "End of script"
```

start_audit

Use this function to call an audit script. This function is typically the first instruction in an audit script, not including comments or variable declaration. This function defines the name of the script, displays the banners, and resets the score to 0.

You can supply the following arguments to this function:

- String value representing the name of the audit script.
- String value representing a description of the audit script. This description can be multiple lines and is formatted using the `logFormattedMessage` function.
- String value representing related information that you want displayed for users after a `PASS` or `FAIL` message (optional). If used, this information is formatted using the `logFormattedMessage` function.

Example usage:

```
start_audit disable-apache.aud "Apache" "Description of Check"
```

Example output:

```
#----------------------------------------------------------------
# Apache
#
# Description of Check
#----------------------------------------------------------------
```

CHAPTER 2

File Templates

This chapter provides reference information about how to use, modify, and customize the file templates included in the Solaris Security Toolkit software. Also, this chapter describes how drivers process functions and other information that is stored in file templates.

This chapter contains the following topics:

Customizing File Templates

File templates are an integral part of the Solaris Security Toolkit software. These files provide a mechanism for you to customize and distribute scripts easily through environment variables, OS version numbers, and client host names. You can leverage the contents of the `Files` directory in combination with finish scripts to isolate related changes, depending on the design of your security profile (driver).

This section provides instructions and recommendations for customizing file templates, including instructions for creating new files in the `Files` directory.

For information about customizing drivers, finish scripts, and audit scripts, see the following chapters:

- To customize drivers, see Chapter 3.
- To customize finish scripts, see Chapter 4.
- To customize audit scripts, see Chapter 5.

Note – Consider submitting a request for enhancement if you think that your customized files could benefit a wider audience. The Solaris Security Toolkit development team is always looking for ways to improve the software to benefit users.

▼ To Customize a File Template

Use the following steps to customize file templates (files) so that your custom versions are available and not overwritten if newer versions of software are released and installed on your systems.

1. **Copy the files and any related files that you want to customize.**
2. **Rename the copies with names that identify the files as custom files.**

   For recommendations, refer to "Configuring and Customizing the Solaris Security Toolkit Software", Chapter 1, *Solaris Security Toolkit 4.1 Administration Guide*.
3. **If applicable, modify your custom drivers to call the uniquely named files appropriately.**

   The following code sample shows a modification to the `JASS_FILES` environment variable that customizes which files are copied to a particular host.

```
JASS_FILES="
[...]
       /etc/init.d/nddconfig
       /etc/rc2.d/S70nddconfig
[...]
"
```

   In this case, a customized hardening driver called `abccorp-starfire_ssp-hardening.driver` uses a custom `nddconfig` file. Instead of modifying the `nddconfig` original file, which could be overwritten with an updated Solaris Security Toolkit software release, create a custom `nddconfig` script by appending

the host name of the destination system to the file name in the `Files` directory. The following example shows a custom `nddconfig` script that has the host name of the destination system in the script file name

```
# pwd
/opt/jass-n.n
# find Files -name "*nddconfig*"
Files/etc/init.d/nddconfig
Files/etc/init.d/nddconfig.ssp-db-serv
Files/etc/rc2.d/S70nddconfig -> ../init.d/nddconfig
```

Note – Be advised that in some cases a script name cannot be changed because a specific name is required by the software. In these cases, use a suffix, as described in this chapter. Or, create a finish script that makes the copies and renames the files as appropriate. If you use this latter option, make sure that the copy and rename operations are compatible with reversing the changes through an undo run. For more information about customizing files, drivers, and scripts so that changes can be reversed, refer to Chapter 4, *Solaris Security Toolkit 4.1 Administration Guide*.

Understanding Rules for How Files Are Copied

Files are copied automatically by the software from the `JASS_HOME_DIR/Files` directory based on the way you define the `JASS_FILES` and `JASS_FILE_`*OS version* environment variables. For information about these environment variables, see Chapter 6.

The Solaris Security Toolkit software differentiates between multiple files in the `JASS_HOME_DIR/Files` directory and the definitions in the `JASS_FILES` and `JASS_FILE_`*OS version* environment variables.

The files that are copied by the `copy_files` function are selected by the following criteria:

- `/some/fully/qualified/path/file.${HOST}`

  In this option, the software copies the object only if the name of the target platform matches the value specified by `${HOST}`. This host environment variable uses the same naming format as the `JASS_HOSTNAME` environment variable.

  For example: `/etc/issue.jordan`

If the software cannot find this file, the software continues to search for the general file (the option described next).

- `/some/fully/qualified/path/file+${OS}`

  In this option, the software only copies the object if the OS revision of the target platform matches the value specified by `${OS}`. The OS parameter uses the same naming format as the `JASS_OS_REVISION` environment variable. So, a file to be used only on the Solaris 8 OS is denoted as "`filename+5.8`".

  For example: `/etc/issue+5.9`

  If the software cannot find this file, the software looks for the file associated with the host name (the option described next).

- `/some/fully/qualified/path/file`

  In this option, the software copies the file to a target system.

  For example: `/etc/issue`

Note – When the file length or size is zero, the file is not copied to the system.

The order of precedence used to match a file is listed. For example, if a host-specific and general file both exist, the host-specific file is used if the name of a target system matches the host name defined by the host-specific file.

Using Configuration Files

You can configure the Solaris Security Toolkit software by editing configuration files that reference environment variables. This feature allows you to use the Solaris Security Toolkit software drivers in different environments, without modifying finish or audit scripts directly.

All Solaris Security Toolkit environment variables are maintained in a set of configuration files. These configuration files are imported by drivers, which make the variables available to finish and audit scripts as they are called by the drivers.

The Solaris Security Toolkit software has three primary configuration files, all of which are stored in the Drivers directory:

- `driver.init`
- `finish.init`
- `user.init.SAMPLE`

driver.init

This file contains environment variables that define aspects of the Solaris Security Toolkit software framework and overall operation.

Note – Do not alter the `driver.init` file, because it is overwritten when you upgrade to subsequent versions of the Solaris Security Toolkit software.

Core environment variables such as `JASS_VERSION` and `JASS_ROOT_DIR` are in the `driver.init` script.

This script loads the `user.init` script, thereby incorporating any user variables or environment variable overrides. Also, this script loads the contents of the `finish.init` file to set any finish script variables that might not have been defined. This script serves as the public interface used by drivers to load all of the variables used by the Solaris Security Toolkit software. None of the other initialization functions are supposed to be directly accessed by any of the driver, finish, or audit scripts.

This file contains the following environment variables:

- `JASS_AUDIT_DIR`
- `JASS_CHECK_MINIMIZED`
- `JASS_DISABLE_MODE`
- `JASS_FILES_DIR`
- `JASS_FINISH_DIR`
- `JASS_HOME_DIR`
- `JASS_HOSTNAME`
- `JASS_ISA_CAPABILITY`
- `JASS_MODE`
- `JASS_NOVICE_USER`
- `JASS_OS_REVISION`
- `JASS_OS_TYPE`
- `JASS_PACKAGE_DIR`
- `JASS_PATCH_DIR`
- `JASS_PKG`
- `JASS_REPOSITORY`
- `JASS_ROOT_DIR`
- `JASS_RUN_AUDIT_LOG`
- `JASS_RUN_CHECKSUM`
- `JASS_RUN_INSTALL_LOG`
- `JASS_RUN_MANIFEST`
- `JASS_RUN_SCRIPT_LIST`
- `JASS_RUN_UNDO_LOG`
- `JASS_RUN_VERSION`
- `JASS_SAVE_BACKUP`

- JASS_STANDALONE
- JASS_SUFFIX
- JASS_TIMESTAMP
- JASS_USER_DIR
- JASS_VERBOSITY
- JASS_VERSION

Each of these environment variables is described in Chapter 6.

finish.init

This file contains environment variables that define the behavior of the individual finish scripts. The two factors that contribute to how a system is hardened are as follows:

- The driver selected contains the list of finish scripts to execute and files to install.
- The `finish.init` file defines how the executed finish scripts act.

Note – Do not alter the `finish.init` file, because it is overwritten when you upgrade to subsequent versions of the Solaris Security Toolkit software.

This file contains the following environment variables:

- JASS_ACCT_DISABLE
- JASS_ACCT_REMOVE
- JASS_AGING_MAXWEEKS
- JASS_AGING_MINWEEKS
- JASS_AGING_WARNWEEKS
- JASS_AT_ALLOW
- JASS_AT_DENY
- JASS_BANNER_DTLOGIN
- JASS_BANNER_FTPD
- JASS_BANNER_SENDMAIL
- JASS_BANNER_SSHD
- JASS_BANNER_TELNETD
- JASS_CORE_PATTERN
- JASS_CPR_MGT_USER
- JASS_CRON_ALLOW
- JASS_CRON_DENY
- JASS_CRON_LOG_SIZE
- JASS_FIXMODES_DIR
- JASS_FIXMODES_OPTIONS
- JASS_FTPD_UMASK
- JASS_FTPUSERS
- JASS_KILL_SCRIPT_DISABLE

- JASS_LOGIN_RETRIES
- JASS_MD5_DIR
- JASS_PASSWD
- JASS_PASS_LENGTH
- JASS_POWER_MGT_USER
- JASS_OS_REVISION
- JASS_REC_PATCH_OPTIONS
- JASS_RHOSTS_FILE
- JASS_ROOT_GROUP
- JASS_ROOT_PASSWORD
- JASS_SADMIND_OPTIONS
- JASS_SENDMAIL_MODE
- JASS_SGID_FILE
- JASS_SHELLS
- JASS_SHELL_DISABLE
- JASS_SUID_FILE
- JASS_SUSPEND_PERMS
- JASS_SVCS_DISABLE
- JASS_SVCS_ENABLE
- JASS_TMPFS_SIZE
- JASS_UMASK
- JASS_UNOWNED_FILE
- JASS_WRITABLE_FILE

Each of these environment variables is described in Chapter 6.

user.init.SAMPLE

This file is for adding user-defined variables. You can override variables defined in the `driver.init` and `finish.init` files by defining the variables in the `user.init` file. This feature allows administrators to customize the Solaris Security Toolkit software to suit their site needs and requirements.

A `user.init.SAMPLE` is included to provide an example of what must be defined for the software to function properly. Copy and modify the copy of `user.init.SAMPLE` to fit your environment, then rename the file to `user.init`. Because a `user.init` file is not included with the software, you can create and customize it without it being overwritten during subsequent software upgrades.

The `user.init` file provides default values for the following environment variables:

- JASS_PACKAGE_MOUNT
- JASS_PATCH_MOUNT

The default values for these two variables are *JumpStart-server-IP address*`/jumpstart/Packages` and *JumpStart-server-IP address*`/jumpstart/Patches`, respectively. These are the recommendations made in Chapter 5, *Solaris Security Toolkit 4.1 Administration Guide* and in the Sun BluePrints book *JumpStart Technology: Effective Use in the Solaris Operating Environment*. If you follow the recommendations made in these other sources, then no changes are required in the `user.init.SAMPLE` file. Simply copy this file to `user.init`.

However, if you move the JumpStart environment from one site to another, verify these variables, as they can be modified to reference your JumpStart server and directory paths. Each of these environment variables is described in Chapter 6.

You can also make modifications to the `JASS_SVCS_ENABLE` and `JASS_SVCS_DISABLE` variables and other environment variables through the `user.init` file. However, because variables might already be used in specific drivers, care must be taken when modifying the behavior of the Solaris Security Toolkit software.

For example, the `suncluster3x-secure.driver` uses `JASS_SVCS_ENABLE` to leave certain services enabled in the `/etc/inetd.conf` file. If you want other services enabled, create and customize a version of the `suncluster3x` driver file, comment out the definition of `JASS_SVCS_ENABLE`, and add a new `JASS_SVCS_ENABLE` definition to the `user.init` file.

You can also add environment variables to the `user.init` and `user.run` scripts. To add a new variable, add the variable declaration with its default value and export it in the `user.init` file. This process provides a global, default value that you can subsequently change as needed by overriding it within a security profile (driver).

In the CODE EXAMPLE 2-1, the code adds a new variable `JASS_ACCT_DISABLE` to the `user.init` file to disable a list of user accounts. These accounts are disabled when finish scripts are run.

CODE EXAMPLE 2-1 Adding a User-Defined Variable

```
if [ -z "${JASS_ACCT_DISABLE}" ]; then
   if [ -f ${JASS_HOME_DIR}/Drivers/finish.init ]; then
      . ${JASS_HOME_DIR}/Drivers/finish.init
   fi

   JASS_ACCT_DISABLE="${JASS_ACCT_DISABLE} newuser"
   export JASS_ACCT_DISABLE
fi
```

Note – If you remove `SUNWjass` using the `pkgrm` command, the `user.init` and `user.run` files, if created, are not removed. However, the `Files` directory and `sysidcfg` files exist in the current distribution of the Solaris Security Toolkit software, and would, therefore, be removed.

Using File Templates

The software uses the `Files` directory with the `JASS_FILES` environment variable and the `copy_files` function. This directory stores file templates that are copied to a JumpStart client during a hardening run.

The following file templates are in the `Files` directory:

- `.cshrc`
- `.profile`
- `etc/default/sendmail`
- `etc/dt/config/Xaccess`
- `etc/hosts.allow`
- `etc/hosts.deny`
- `etc/init.d/klmmod`
- `etc/init.d/nddconfig`
- `etc/init.d/set-tmp-permissions`
- `etc/init.d/sms_arpconfig`
- `etc/issue`
- `etc/motd`
- `etc/notrouter`
- `etc/rc2.d/S00set-tmp-permissions`
- `etc/rc2.d/S07set-tmp-permissions`
- `etc/rc2.d/S70nddconfig`
- `etc/rc2.d/S73sms_arpconfig`
- `etc/rc2.d/S73swapadd`
- `etc/rc2.d/S77klmmod`
- `etc/security/audit_class`
- `etc/security/audit_control`
- `etc/security/audit_event`
- `etc/sms_domain_arp`
- `etc/sms_sc_arp`
- `etc/syslog.conf`

The following subsections describe each of these files.

.cshrc

This configuration file is provided as a sample. It provides some base-level configuration for `csh` users by setting some common `csh` variables such as file completion and history. In addition, it sets the kill and erase terminal options, as well as a command line prompt that includes the path to the current working directory. This file is not required for the software to function properly and can be modified or replaced as appropriate for your environment.

By default, this file is copied by the `config.driver` to the system being hardened.

.profile

This configuration file is provided as a sample. As distributed with the software, this configuration only defines a `UMASK`, the `PATH`, and `MANPATH` for any `root sh` started shells.

This file is not required for the software to function properly and can be modified or replaced as appropriate for your environment.

By default, this file is copied by the `config.driver` to the system being hardened.

etc/default/sendmail

With the release of Solaris 8 OS, a `sendmail` configuration file can be used to run `sendmail` in queue processing mode only. This file is copied only onto Solaris 8 OS systems being hardened by the `disable-sendmail.fin` script.

The `disable-sendmail.fin` script is OS-version aware and modifies the behavior of sendmail based on the OS being hardened. For more information, refer to the Sun BluePrints OnLine article titled "Solaris Operating Environment Security: Updated for Solaris 9 OE."

By default, this file is copied by the `disable-sendmail.fin` to the system being hardened.

etc/dt/config/Xaccess

This file disables all remote access, whether direct or broadcast, to any X server running on the system. Depending on the X support requirements and the environment the Solaris Security Toolkit software is used in, this file might not be appropriate.

By default, this file is copied by the `hardening.driver` to the system being hardened.

`etc/hosts.allow` and `etc/hosts.deny`

These two files are installed on Solaris 9 OS systems by the finish script `enable-tcpwrappers.fin`. After installing the `hosts.allow` and `hosts.deny` files, the finish script enables Transmission Control Protocol (TCP) wrappers by modifying the `/etc/default/inetd` configuration file.

The `hosts.allow` and `hosts.deny` files are samples to customize for your security profile based on local policies, procedures, and requirements. The default configuration of the `hosts.allow` defines permitted Solaris Secure Shell (SSH) access to be `LOCAL`, which means that SSH connections are only permitted from the subnet to which the system is connected. The default configuration of the `hosts.deny` file is to deny all connection attempts not permitted in the `hosts.allow`.

By default, this file is copied by the `enable-tcpwrappers.fin` to the system being hardened.

`etc/init.d/klmmod` and `etc/rc2.d/S77klmmod`

These files are used on Sun Fire High-End Systems domains to ensure that the `misc/klmmod` kernel module is loaded. For further information, refer to the Sun Blueprints OnLine article, “Securing the Sun Fire 12K and 15K Domains.”

By default, these files are copied by the `s15K-install-klmmod-loader.fin` to the domain being hardened.

`etc/init.d/nddconfig`

This file copies over the `nddconfig` startup script required to implement network settings, which improves security. For information about configuring network settings for security, refer to the Sun BluePrints OnLine article titled “Solaris Operating Environment Network Settings for Security: Updated for the Solaris 9 Operating Environment.”

By default, this file is copied by the `hardening.driver` to the system being hardened.

etc/init.d/set-tmp-permissions

This file sets the correct permissions on the `/tmp` and `/var/tmp` directories when a system is rebooted. If an inconsistency is found, it is displayed to standard output and logged via `SYSLOG`. This file is installed in `/etc/rc2.d` twice to permit this check to be performed both before and after the `mountall` command is run from `S01MOUNTFSYS`. This check helps ensure that both the mount point and the mounted file system have the correct permissions and ownership.

By default, this file is copied by the `hardening.driver` to the system being hardened.

etc/init.d/sms_arpconfig

This file, in combination with the `/etc/rc2.d/S73sms_arpconfig`, `/etc/sms_domain_arp`, and `/etc/sms_sc_arp` is for use on Sun Fire™ High-End Systems to implement static Address Resolution Protocol (ARP) on the internal IP-based management network for additional security. For information about how to use these capabilities, refer to the Sun BluePrints OnLine articles titled "Securing the Sun Fire 12K and 15K System Controllers" and "Securing the Sun Fire 12K and 15K Domains."

By default, this file is copied by the `s15k-static-arp.fin` to the system being hardened.

etc/issue and /etc/motd

These files are based on United States (U.S.) government recommendations and provide legal notice that user activities could be monitored. If an organization has specific legal banners, they can be installed into these files.

These files are provided as default templates. Have your legal counsel provide or review notices that apply to your organization.

By default, this file is copied by the `hardening.driver` to the system being hardened.

etc/notrouter

This file disables IP forwarding between interfaces on a system by creating an `/etc/notrouter` file. The client no longer functions as a router regardless of the number of network interfaces.

By default, this file is copied by the `hardening.driver` to the system being hardened.

etc/rc2.d/S00set-tmp-permissions and etc/rc2.d/S07set-tmp-permissions

---

Note – These files are symbolic links to `/etc/init.d/set-tmp-permissions`.

These files set the correct permissions on the `/tmp` and `/var/tmp` directories when a system is rebooted. If an inconsistency is found, it is displayed to standard output and logged via `SYSLOG`. These scripts are installed into `/etc/rc2.d` twice to permit this check to be performed both before and after the `mountall` command is run from `S01MOUNTFSYS`. This check helps ensure that both the mount point and the mounted file system have the correct permissions and ownership.

By default, these files are copied by the `hardening.driver` to the system being hardened.

etc/rc2.d/S70nddconfig

Note – This file is a symbolic link to `/etc/init.d/nddconfig`.

This file copies over the `S70nddconfig` startup script required to implement network settings, which improves security. Refer to the Sun BluePrints OnLine article titled “Solaris Operating Environment Network Settings for Security: Updated for Solaris 9 Operating Environment.”

By default, this file is copied by the `hardening.driver` to the system being hardened.

etc/rc2.d/S73sms_arpconfig

Note – This file is a symbolic link to `/etc/init.d/sms_arpconfig`.

This file in conjunction with the `/etc/init.d/sms_arpconfig`, `/etc/sms_domain_arp`, and `/etc/sms_sc_arp` files is for use on Sun Fire High-End Systems to implement static Address Resolution Protocol (ARP) on the internal IP-based management network for additional security. For information about how to use these capabilities, refer to the Sun BluePrints OnLine articles titled "Securing the Sun Fire 12K and 15K System Controllers" and "Securing the Sun Fire 12K and 15K Domains."

By default, this file is copied by the `s15k-static-arp.fin` to the system being hardened.

etc/rc2.d/S77swapadd

This file is installed when `disable-nfs-client.fin` runs. As `disable-nfs-client.fin` normally starts the swap space, this run control script is added by the Solaris Security Toolkit software to perform this task.

etc/security/audit_class, etc/security/audit_control, and etc/security/audit_event

These are configuration files for the Solaris OS auditing subsystem, also referred to as the Solaris Basic Security Module, released in February 2001. If you add these three files to a Solaris 8 OS system, it configures the auditing subsystem.

These files are only installed by the Solaris Security Toolkit software on Solaris 8 OS systems. For more information, refer to the Sun BluePrints OnLine article titled "Auditing in the Solaris 8 Operating Environment."

By default, these files are copied by the `enable-bsm.fin` to the system being hardened.

etc/sms_domain_arp and /etc/sms_sc_arp

These files in combination with the `/etc/init.d/sms_arpconfig` and `/etc/S70sms_arpconfig` files are for use on Sun Fire High-End Systems to implement static Address Resolution Protocol (ARP) on the internal IP-based management network for additional security. For information about how to use these capabilities, refer to the Sun BluePrints OnLine articles titled “Securing the Sun Fire 12K and 15K System Controllers” and “Securing the Sun Fire 12K and 15K Domains.”

By default, these files are copied by the `s15k-static-arp.fin` to the system being hardened.

etc/syslog.conf

This file performs additional logging. It serves as a placeholder for organizations to add their own centralized log server (or servers) so that proactive log analysis can be done.

By default, this file is copied by the `hardening.driver` to the system being hardened.

CHAPTER 3

Drivers

This chapter provides reference information about using, adding, modifying, and removing drivers. This chapter describes the drivers used by the Solaris Security Toolkit software to harden, minimize, and audit Solaris OS systems. A series of drivers and related files make up a security profile.

The default driver (`secure.driver`) in the Solaris Security Toolkit software disables all services, including network services, not required for the OS to function. This action might not be appropriate for your environment. Evaluate which security modifications are required for your system, then make adjustments by using the information in this chapter and related chapters.

This chapter contains the following topics:

Understanding Driver Functions and Processes

The core processing for hardening and audit runs are defined by the functions in the `driver.run` script. During these operations, the driver in use calls the `driver.run` script after the security profile is configured. That is, after the `driver.init` file is called and the `JASS_FILES` and `JASS_SCRIPTS` environment variables are defined, the driver calls the `driver.run` script functions. This script processes each of the entries contained in the `JASS_FILES` and `JASS_SCRIPTS` environment variables in both the hardening and audit operations.

The high-level processing flow of this script is as follows:

1. Load functionality (`.funcs`) files
2. Perform basic checks
3. Load user functionality overrides
4. Mount file systems to JumpStart client (JumpStart mode only)
5. Copy or audit files specified by the `JASS_FILES` environment variable (optional)
6. Execute scripts specified by the `JASS_SCRIPTS` environment variable (optional)
7. Compute total score for the run (audit operation only)
8. Unmount file systems from JumpStart client (JumpStart mode only)

Each of these functions are described in detail in the following subsections.

Load Functionality Files

The first task of the `driver.run` script is to load the functionality files. Loading these files at this stage allows the `driver.run` script to take advantage of the functionality in each of the files. Also, any scripts that are executed can take advantage of the common functions. The functionality files loaded during this task are the following:

- `common_misc.funcs`
- `common_log.funcs`
- `driver.funcs`
- `audit_public.funcs`

Perform Basic Checks

The Solaris Security Toolkit software checks to determine if core environment variables are set. This check ensures that the software is properly executed. If any of the checks fail, the software reports an error and exits. The software checks to ensure the following:

- The `JASS_OS_REVISION` environment variable is defined. If this environment variable was not defined, it is possible that either the `driver.init` script was not called or the environment variable was improperly modified.
- For JumpStart mode, the `JASS_PACKAGE_MOUNT` environment variable is defined. If this environment variable is not properly defined, then the software might not be able to locate the `Packages` directory during a JumpStart installation.

- For JumpStart mode, the `JASS_PATCH_MOUNT` environment variable is defined. If this environment variable is not properly defined, then the software might not be able to locate the `Patches` directory during a JumpStart installation.

Load User Functionality Overrides

Before continuing to process the current profile, the Solaris Security Toolkit software loads the `user.run` file, if it exists. This file stores all site or organization-specific functions, including those that override any Solaris Security Toolkit software default functions. By default, this file does not exist and must be manually created by the user if this functionality is needed.

This capability allows you to extend or enhance the functionality of the software by implementing new functions or customizing existing ones to better suit your environment. This file is similar to the `user.init`, except that this file is for functions, whereas the `user.init` file is for environment variables.

Mount File Systems to JumpStart Client

Note – If using a local, bootable CD-ROM for JumpStart installation, modify this functionality to access the directories from the local media. No changes are necessary if accessing the Patches and Packages directory from a remote server using the Network File System (NFS).

In JumpStart mode, the `driver.run` script calls an internal subroutine called `mount_filesystems`. This routine mounts the following directories onto the JumpStart client:

- `JASS_PACKAGE_MOUNT`, which is mounted onto `JASS_PACKAGE_DIR`
- `JASS_PATCH_MOUNT`, which is mounted onto `JASS_PATCH_DIR`

If other file system mount points are required, use the `user.run` script to implement them. This routine is JumpStart mode specific and is not executed during standalone mode runs.

Copy or Audit Files

After the software establishes its foundation by loading common functions, initializing environment variables, and mounting file systems (if needed), it is ready to begin its work. Whether performing a hardening or audit operation, the software assembles a complete list of file templates to be copied to or verified on a target

system. The software does this task by concatenating the entries found in the JASS_FILES global environment variable with entries found in the JASS_FILES_*x_x* OS-version environment variable (for example, JASS_FILES_5_8 for Solaris 8 OS). Note that both the global and OS environment variables are optional, and either or none can be defined. The combined list is stored in the JASS_FILES environment variable. For more information about this variable, see Chapter 6, "JASS_FILES" on page 167.

If the resulting list has at least one entry, the software prepends the JASS_SCRIPTS list with a special finish script called install-templates.fin. In hardening runs, this script takes the contents of the resulting list and copies it to a target system before other finish scripts are run. In audit runs, the install-templates.aud script verifies that the files were successfully copied to a target system.

Execute Scripts

The software executes the scripts defined by the JASS_SCRIPTS environment variable. Whether performing a hardening or audit operation, the software assembles a complete list of file templates to be copied to or verified on a target system. The software does this task by concatenating the entries found in the JASS_SCRIPTS global environment variable with entries found in the JASS_SCRIPTS_*x_x* OS-version environment variable (for example, JASS_SCRIPTS_5_8 for Solaris 8 OS). Note that both the global and OS environment variables are optional, and either or none can be defined. The combined list is stored in the JASS_SCRIPTS environment variable. For more information about this variable, see Chapter 6, "JASS_FINISH_DIR" on page 171.

In hardening runs, each finish script is executed in turn. The finish scripts are stored in the JASS_FINISH_DIR directory.

In audit runs, some additional processing must be done first. Before a script defined by JASS_SCRIPTS executes, it must first be transformed from its finish script name to its audit script counterpart. The Solaris Security Toolkit software automatically changes the file name extension from .fin to .aud. In addition, the software expects the audit script to be in the JASS_AUDIT_DIR. After this alteration is made, the software executes each audit script in turn.

The output of the scripts is processed in one or more of the following ways:

- Logged to the file specified by the jass-execute -o option. If a file is not specified, the output is directed to standard output. This option is only available in standalone mode.

- Logged into the `/var/sadm/system/logs/finish.log` file on the JumpStart client during JumpStart installations. The `/var/sadm/system/logs/finish.log` is the standard log file used by any JumpStart command run on the client. This option is only available in JumpStart mode.
- Logged to the file `JASS_REPOSITORY/`*timestamp*`/jass-install-log.txt` or `jass-audit-log.txt`. The timestamp is a fully qualified time parameter of the form YYYYMMDDHHMMSS. This value is constant for each run of the Solaris Security Toolkit software and represents the time at which the run was started. For example, a run started at 1:30 p.m. on April 1, 2003 would be represented by the value 20030401013000. These log files are generated during every run. In hardening runs, the software creates the `jass-install-log.txt` file. In audit runs, the software creates the `jass-audit-log.txt` file. Do not modify the contents of these files.

Compute Total Score for the Run

In audit runs, after all of operations are completed for a driver, the software calculates the driver's total score. This score denotes the status of the driver and is part of the grand total if multiple drivers are called. If only one driver is used, then this total and the grand total are the same value. The score is zero if all of the checks passed. If any checks fail, the score is a number representing how many checks or subchecks fail.

Unmount File Systems From JumpStart Client

When operating in JumpStart mode, after all operations are completed for a driver, the software unmounts those file systems mounted during the process "Mount File Systems to JumpStart Client" on page 69. This functionality typically marks the end of a JumpStart client's installation. At this point, control returns to the calling driver. The driver can either exit and end the run or it can call other drivers and start new processing.

Customizing Drivers

Modifying the Solaris Security Toolkit drivers is one of the tasks done most often because each organization's policies, standards, and application requirements differ, even if only slightly. For this reason, the Solaris Security Toolkit software supports the ability to customize tasks undertaken by a driver.

If your system or application requires some of the services and daemons that are automatically disabled by the Solaris Security Toolkit software, or if you want to enable any of the inactive scripts, do so before executing the Solaris Security Toolkit software.

Similarly, if there are services that must remain enabled, and the Solaris Security Toolkit software automatically disables them, override the defaults before executing the applicable driver in the Solaris Security Toolkit software. Review the configuration of the software and make all necessary customization before changing the system's configuration. This approach is more effective than discovering that changes must be reversed and reapplied using a different configuration.

There are two primary ways in which services can be disabled using the Solaris Security Toolkit software. The first way involves modifying drivers to comment out or remove any finish scripts defined by the `JASS_SCRIPTS` parameter that should not be run. This approach is one of the most common ways to customize drivers.

For example, if your environment requires NFS-based services, you can leave them enabled. Comment out the `disable-nfs-server.fin` and `disable-rpc.fin` scripts by prepending a `#` sign before them in your local copy of the `hardening.driver`. Alternatively, you can remove them entirely from the file. As a general rule, it is recommended that any entries that are commented out or removed should be documented in the file header, including information such as:

- Name of the script that is disabled
- Name of the person who disabled the script
- Timestamp indicating when the change was made
- Brief description for why this change was necessary

Including this information can be very helpful in maintaining drivers over time, particularly when they must be updated for newer versions of the software.

Note – Never make changes directly to the drivers distributed with the Solaris Security Toolkit software. Always modify local copies of drivers so that the changes made are not impacted by the removing or upgrading of the Solaris Security Toolkit software.

The other method for disabling services is to customize environment variables. This approach is typically done in either the driver or the `user.init` file. Make changes in the `user.init` file only if the changes are global in nature and used by all of the drivers. Otherwise, localize the change to just the drivers requiring the change.

For example, to enable or disable services started by the `inetd` daemon, use the `JASS_SVCS_ENABLE` and `JASS_SVCS_DISABLE` environment variables. See Chapter 6 for detailed information about using variables. Also, see "Customizing and Assigning Variables" on page 157 in Chapter 6.

▼ To Customize a Driver

Use the following steps to customize a driver so that newer versions of the original files do not overwrite your customized versions. Furthermore, this step should be taken to help ensure that customized files are not accidentally deleted during software upgrades or removal.

1. **Copy the driver and any related files that you want to customize.**

   For example, if you want to create a `secure.driver` specific to your organization, copy the following drivers located in the `Drivers` directory:

   - `secure.driver`
   - `config.driver`
   - `hardening.driver`

   The `config.driver` and `hardening.driver` must be copied because they are called by the `secure.driver`. If the driver you are customizing does not call or use other drivers, copy only the driver.

2. **Rename the copies with names that identify the files as custom drivers.**

   For example, using your company's name, your files would look like:

   - `abccorp-secure.driver`
   - `abccorp-config.driver`
   - `abccorp-hardening.driver`

   For more information, refer to "Configuring and Customizing the Solaris Security Toolkit Software", Chapter 1, *Solaris Security Toolkit 4.1 Administration Guide*.

3. **Modify your custom** *prefix*`-secure.driver` **to call the new related** *prefix*`-config.driver` **and** *prefix*`-hardening.driver` **files accordingly.**

   This step is necessary to prevent the new *prefix*`-secure.driver` from calling the original `config.driver` and `hardening.driver`. This step is not necessary if the drivers being customized do not call or use other drivers.

4. **To copy, add, or remove files from a driver, modify the `JASS_FILES` environment variable.**

   For detailed information about this variable, see Chapter 6.

   The following code example is taken from the `Drivers/config.driver` file. This security profile performs basic configuration tasks on a platform. The security profile provides clear samples of how both file templates and finish scripts are used.

   In the following example, the driver is configured to copy the `/.cshrc` and `/.profile` files from the `JASS_HOME_DIR/Files/` directory onto the target platform when the `driver.run` function is called.

```
JASS_FILES="
/.cshrc
/.profile
"
```

   a. **To change the contents of either of these files, modify the copies of the files located in the `JASS_HOME_DIR/Files/` directory.**

   b. **If you only need to add or remove file templates, adjust the `JASS_FILES` variable accordingly.**

   c. **If you want to define the Solaris OS version, append the major and minor operating system version to the end of the `JASS_FILES` variable, separated by underscores.**

      The Solaris Security Toolkit software supports operating system-version specific file lists. These file lists are added to the contents of the general file list only when the Solaris Security Toolkit software is run on a defined version of the Solaris OS. For example, Solaris 9 OS would be specified

```
JASS_FILES_5_9
```

5. **To add or remove scripts from a driver, modify the `JASS_SCRIPTS` variable.**

   For detailed information about this variable, see Chapter 6.

6. **To call other drivers, create a nested or hierarchical security profile.**

   This technique is often useful when attempting to enforce standards across the majority of platforms while still providing for platform or application-specific differences.

   CODE EXAMPLE 3-1 comes from the `secure.driver` file. This file is used as a wrapper to call both configuration and hardening drivers that, in this case, implement the actual functionality of the security profile. Although this is often the

model used, it should be noted that this need not be the case. In fact, each driver supports the JASS_FILES and JASS_SCRIPTS convention, even if it is not always used (as is the case in CODE EXAMPLE 3-1).

CODE EXAMPLE 3-1 Creating a Nested or Hierarchical Security Profile

```
DIR="`/bin/dirname $0`"
export DIR

. ${DIR}/driver.init
. ${DIR}/config.driver
. ${DIR}/hardening.driver
```

CODE EXAMPLE 3-2 illustrates a slightly more complex configuration where the driver not only calls other foundational drivers, but also implements its own functionality. In this case, this new security profile installs the /etc/named.conf file and runs the configure-dns.fin script after it runs the config.driver and hardening.driver drivers.

CODE EXAMPLE 3-2 Having a Driver Implement Its Own Functionality

```
DIR="`/bin/dirname $0`"
export DIR

. ${DIR}/driver.init
. ${DIR}/config.driver
. ${DIR}//hardening.driver

JASS_FILES="
/etc/named.conf
"

JASS_SCRIPTS="
configure-dns.fin
"

. ${DIR}/driver.run
```

Note – CODE EXAMPLE 3-2 shows a sample of how you can nest drivers to provide various levels of functionality and coverage. The /etc/named.conf and configure-dns.fin references are for example purposes only. Those files are not supplied by default with the Solaris Security Toolkit software.

---

7. **When finished customizing your driver, save it in the** Drivers **directory.**
8. **Test the driver to ensure that it functions properly.**

Using Standard Drivers

This section describes the following drivers, which are supplied by default in the `Drivers` directory:

- "`config.driver`" on page 76
- "`hardening.driver`" on page 77
- "`secure.driver`" on page 79
- "`undo.driver`" on page 80

In addition to these standard drivers, product-specific drivers are also included with the Solaris Security Toolkit distribution. For a list of product-specific drivers, see "Using Product-Specific Drivers" on page 80.

`config.driver`

This driver is called by the `secure.driver` and is responsible for implementing tasks associated with that driver set. By grouping related functions into a single driver, you can create common functions and use them as building blocks to assemble more complex configurations. In the following example, machines with different security requirements can share the same base Solaris OS configuration driver because similar tasks are separated into their own driver.

CODE EXAMPLE 3-3 shows sample output from the `config.driver`.

CODE EXAMPLE 3-3 Sample Output from `config.driver`

```
DIR="`/bin/dirname $0`"
export DIR

. ${DIR}/driver.init

JASS_FILES="
/.cshrc
"

JASS_SCRIPTS="
set-root-password.fin
set-term-type.fin
"

. ${DIR}/driver.run
```

This driver performs several tasks. It calls the `driver.init` to initialize the Solaris Security Toolkit framework and to configure its runtime environment. Then, it sets both the `JASS_FILES` and `JASS_SCRIPTS` environment variables. These variables define the actual configuration changes that are undertaken by this driver. After these variables are set, the `driver.run` script is called. The `driver.run` script completes the installation of the files and executes all configuration-specific scripts.

In the example, the `.cshrc` file contained in `JASS_HOME_DIR/Files` directory is copied to `/.cshrc` and the finish scripts (`set-root-password.fin` and `set-term-type.fin`) are run on the system.

hardening.driver

Most of the security-specific scripts included in the Solaris Security Toolkit software are listed in the `hardening.driver`. This driver builds upon those changes by implementing additional security enhancements that are not included in the `hardening.driver`. This driver, similar to the `config.driver`, defines scripts to be run by the `driver.run` script.

The following scripts are listed in this driver:

- `disable-ab2.fin`
- `disable-apache.fin`
- `disable-asppp.fin`
- `disable-automount.fin`
- `disable-dhcpd.fin`
- `disable-directory.fin`
- `disable-dmi.fin`
- `disable-dtlogin.fin`
- `disable-ipv6.fin`
- `disable-kdc.fin`
- `disable-keyserv-uid-nobody.fin`
- `disable-ldap-client.fin`
- `disable-lp.fin`
- `disable-mipagent.fin`
- `disable-named.fin`
- `disable-nfs-client.fin`
- `disable-nfs-server.fin`
- `disable-nscd-caching.fin`
- `disable-ppp.fin`
- `disable-preserve.fin`
- `disable-power-mgmt.fin`
- `disable-remote-root-login.fin`
- `disable-rhosts.fin`
- `disable-rpc.fin`
- `disable-samba.fin`

- disable-sendmail.fin
- disable-ssh-root-login.fin
- disable-slp.fin
- disable-sma.fin
- disable-snmp.fin
- disable-spc.fin
- disable-syslogd-listen.fin
- disable-system-accounts.fin
- disable-uucp.fin
- disable-vold.fin
- disable-xserver-listen.fin
- disable-wbem.fin
- enable-coreadm.fin
- enable-ftpaccess.fin
- enable-ftp-syslog.fin
- enable-inetd-syslog.fin
- enable-priv-nfs-ports.fin
- enable-process-accounting.fin
- enable-rfc1948.fin
- enable-stack-protection.fin
- install-at-allow.fin
- install-ftpusers.fin
- install-loginlog.fin
- install-newaliases.fin
- install-sadmind-options.fin
- install-security-mode.fin
- install-shells.fin
- install-sulog.fin
- remove-unneeded-accounts.fin
- set-banner-dtlogin.fin
- set-banner-ftpd.fin
- set-banner-sendmail.fin
- set-banner-sshd.fin
- set-banner-telnetd.fin
- set-ftpd-umask.fin
- set-login-retries.fin
- set-power-restrictions.fin
- set-root-group.fin
- set-rmmount-nosuid.fin
- set-sys-suspend-restrictions.fin
- set-system-umask.fin
- set-tmpfs-limit.fin
- set-user-password-reqs.fin
- set-user-umask.fin
- update-at-deny.fin
- update-cron-allow.fin
- update-cron-deny.fin

- update-cron-log-size.fin
- update-inetd-conf.fin
- install-md5.fin
- install-fix-modes.fin

Note – By default, all changes made by the finish scripts provided are reversible, except for changes made by the `install-strong-permissions.fin` script. The changes made by this script must be manually reversed in the event that the changes are no longer wanted.

In addition, the following scripts are listed in the `hardening.driver`, but are commented out by default:

- disable-autoinst.fin
- disable-keyboard-abort.fin
- disable-picld.fin
- enable-tcpwrappers.fin
- enable-bsm.fin
- install-strong-permissions.fin

For descriptions of these scripts, see Chapter 4.

secure.driver

The `secure.driver` is the default driver used in the `rules` file for client installation. This driver is a ready-to-use driver that implements *all* the hardening functionality in the Solaris Security Toolkit software. This driver performs the initialization tasks required, then calls the `config.driver` and `hardening.driver` drivers to configure the system and perform all the hardening tasks.

CODE EXAMPLE 3-4 lists the contents of the `secure.driver`.

CODE EXAMPLE 3-4 `secure.driver` Contents

```
DIR="`/bin/dirname $0`"
export DIR

. ${DIR}/driver.init

. ${DIR}/config.driver

. ${DIR}/hardening.driver
```

undo.driver

Caution – Never call or modify this driver directly.

This driver provides the undo functionality during an undo run. This driver is called when you invoke the `jass-execute` command with the `-u` option. This driver is quite straightforward and contains the contents listed in CODE EXAMPLE 3-5.

CODE EXAMPLE 3-5 `undo.driver` Contents

```
DIR="`/bin/dirname $0`"
export DIR

. ${DIR}/driver.init

. ${DIR}/undo.run
```

When called by `./jass-execute -u`, this driver initializes itself much the same way as any other driver by calling `driver.init`, then passing control to a different driver, `undo.driver` in this case.

Using Product-Specific Drivers

This section lists product-specific drivers, which are used to harden specific Sun products or configurations. These drivers are included with the Solaris Security Toolkit in the Drivers directory. TABLE 3-1 lists product specific drivers.

New drivers are released periodically to harden new and updated Sun products. Newer versions of the Solaris Security Toolkit software may offer new and revised drivers.

TABLE 3-1 Product-Specific Drivers

Product	Driver Name

Desktop systems	`desktop-secure.driver`
	`desktop-config.driver`
	`desktop-hardening.driver`

Sun Java System Web Servers[1]	`install-Sun_ONE-WS.driver`

JumpStart technology	`jumpstart-secure.driver`
	`jumpstart-config.driver`
	`jumpstart-hardening.driver`

Sun Cluster 3.x Software	`suncluster3x-secure.driver`
	`suncluster3x-config.driver`
	`suncluster3x-hardening.driver`

Sun Fire Midrange Systems System Controller	`sunfire_mf_msp-secure.driver`
	`sunfire_mf_msp-config.driver`
	`sunfire_mf_msp-hardening.driver`

Sun Enterprise 10000 System Service Processors	`starfire_ssp-secure.driver`
	`starfire_ssp-config.driver`
	`starfire_ssp-hardening.driver`

Sun Fire High-End Systems Domains	`sunfire_15k_domain-secure.driver`
	`sunfire_15k_domain-config.driver`
	`sunfire_15k_domain-hardening.driver`

Sun Fire High-End Systems System Controllers	`sunfire_15k_sc-secure.driver`
	`sunfire_15k_sc-config.driver`
	`sunfire_15k_sc-hardening.driver`

1 Sun Java System servers were formerly referred to as Sun ONE servers and before that as iPlanet servers.

`desktop-secure.driver`

This driver is provided as an example, based on the `secure.driver`, to highlight what changes may be necessary to secure a desktop system. This script is a guide; therefore, you may need to customize it, depending on your environment. The differences between this and the `secure.driver` are as follows:

- The following `inetd` services are not disabled: Telnet, FTP, `dtspc` (CDE subprocess control service), `rstatd` (Kernel statistics server), and `rpc.smserverd` (used for volume management in Solaris OS 9).
- The following file templates are not used: `/etc/dt/config/Xaccess`, and `/etc/syslog.conf`.
- The following finish scripts are not used: `disable-dtlogin.fin`, `disable-automount.fin`, `disable-lp.fin`, `disable-nfs-client.fin`, `disable-rpc.fin`, `disable-vold.fin`, and `disable-xserver-listen.fin`.

install-Sun_ONE-WS.driver

Applicable only to JumpStart mode, this driver calls the `minimize-Sun_ONE-WS.fin` script so that the Solaris Security Toolkit software can install the Sun Java™ System Web Server (formerly Sun ONE Web Server and before that iPlanet™ Web Server) software. The script removes all Solaris OS packages not required to successfully install and run the Sun Java System Web Server software. In addition, this driver calls the `hardening.driver` to harden the platform after it is minimized.

CODE EXAMPLE 3-6 lists the contents of the `install-Sun_ONE-WS.driver`.

CODE EXAMPLE 3-6 `install-Sun_ONE-WS.driver` Contents

```
DIR="`/bin/dirname $0`"
export DIR

. ${DIR}/driver.init

. ${DIR}/config.driver

JASS_SCRIPTS="
minimize-Sun_ONE-WS.fin
install-Sun_ONE-WS.fin
"
. ${DIR}/driver.run

. ${DIR}/hardening.driver
```

If you build a JumpStart client using this driver, then you must include this driver in the `rules` file list. This driver performs all the actions specified by the `config.driver` and `hardening.driver`, in addition to the functionality in the `minimize-Sun_ONE-WS.fin` and `install-Sun_ONE-WS.fin` scripts.

For more information about this driver, refer to the Sun BluePrints OnLine article titled "Minimizing the Solaris Operating Environment."

jumpstart-secure.driver

This driver is provided as an example, based on the `secure.driver`, to highlight what changes might be necessary to secure a JumpStart server. This driver is a guide, and you might need to customize it, depending on your environment. The differences between this and the `secure.driver` are as follows:

- Trivial FTP (`tftp`) `inetd` service is not disabled
- Following finish scripts are not used: `disable-nfs-server.fin` and `disable-rpc.fin`

suncluster3x-secure.driver

This driver provides a baseline configuration for hardening SunPlex™, formerly Sun Cluster 3.*x*, software releases. You can modify the driver to remove Solaris OS functionality being disabled; however, do not alter enabled services that are required for the Sun Cluster software to work properly. For more information, refer to the Sun BluePrints OnLine article titled "Securing the Sun Cluster 3.x Software."

sunfire_mf_msp-secure.driver

This driver is for hardening the midframe service processor (MSP) when building secured Sun Fire midframe environments. This driver automates and simplifies building a secure MSP, but one that still has all of the required services enabled. For more information, refer to the Sun BluePrints OnLine article titled "Securing the Sun Fire Midframe System Controller."

starfire_ssp-secure.driver

This driver is for creating supported and hardened Sun Enterprise™ 10000 system service processors (SSP). It is strongly recommended that the SSPs always be hardened, due to their ability to impact the reliability, availability, and serviceability of Sun Enterprise 10000 systems. For more information, refer to the Sun BluePrints OnLine article titled "Securing the Sun Enterprise 10000 System Service Processors."

sunfire_15k_domain-secure.driver

This driver provides a baseline for developing hardened Sun Fire High-End Systems domains. The configuration implemented by this driver disables all services not required by Sun Fire High-End Systems, while enabling optional Solaris OS security features disabled in default configurations. For more information, refer to the Sun BluePrints OnLine article titled "Securing Sun Fire 12K and 15K Domains."

sunfire_15k_sc-secure.driver

This driver is the only supported mechanism by which Sun Fire High-End Systems system controllers (SC) can be secured. All services not required by the SC are disabled by this driver. If some of the disabled services are required, you can modify the driver to not disable them. For more information, refer to the Sun BluePrints OnLine article titled "Securing the Sun Fire 12K and 15K System Controllers."

CHAPTER 4

Finish Scripts

This chapter provides reference information about using, adding, modifying, and removing finish scripts. This chapter describes the scripts used by the Solaris Security Toolkit software to harden and minimize Solaris OS systems.

The default scripts in the Solaris Security Toolkit software disable all services, including network services, not required for the OS to function. This action might not be appropriate for your environment. Evaluate which security modifications are required for your system, then make adjustments by using the information in this chapter.

This chapter contains the following topics:

Customizing Finish Scripts

Finish scripts serve as the heart of the Solaris Security Toolkit software. These scripts collectively implement the majority of security modifications. The finish scripts isolate related changes into single files that can be combined and grouped in any number of ways, depending on the design of the security profile (driver).

This section provides instructions and recommendations for customizing existing finish scripts and creating new finish scripts. Also, it provides guidelines for using finish script functions.

Note – Consider submitting a bug report or request for enhancement if you think that the change could benefit a wider audience. The Solaris Security Toolkit development team is always looking for ways to improve the software to better support its users.

Customize Existing Finish Scripts

Just as with Solaris Security Toolkit drivers, you can customize finish scripts. Use great care when modifying scripts that are supplied with the Solaris Security Toolkit software. Always modify a copy of the finish script and not the original script directly. Failure to do so might result in a loss of changes upon Solaris Security Toolkit software upgrade or removal. Also, wherever possible, try to minimize and document the modifications made to scripts.

Customize finish scripts by using environment variables. The behavior of most finish scripts can be tailored using this technique, thereby eliminating the need to modify the actual script. If this is not possible, then you might find it necessary to modify the code.

For a list of all environment variables and guidelines for defining them, see Chapter 6.

Note – When you install the Solaris Security Toolkit software on a JumpStart server, the finish scripts run from a memory resident mini-`root` running on the JumpStart client. The mini-`root` contains almost all of the Solaris OS functions. If you create finish scripts, it is sometimes necessary to execute commands using the `chroot` command, because the client disk is mounted on `/a`. This limitation is not present during a standalone mode Solaris Security Toolkit software installation.

▼ To Customize a Finish Script

Use the following steps to customize a finish script so that new versions of the original files do not overwrite your customized versions. Furthermore, these files are not removed if the software is removed using the `pkgrm` command.

1. **Copy the script and the related files that you want to customize.**
2. **Rename the copies with names that identify the files as custom scripts and files.**

   For naming guidelines, refer to "Configuring and Customizing the Solaris Security Toolkit Software", Chapter 1, *Solaris Security Toolkit 4.1 Administration Guide*.

3. **Modify your custom script and files accordingly.**

CODE EXAMPLE 4-1 shows how to automate software installation using `install-openssh.fin`. In this example, the code expects the version of OpenSSH to be "2.5.2p2," however, the current version of OpenSSH is "3.5p1." Obviously, the version to install varies depending on when the software is installed. This script can also be altered to support a commercial version of the Secure Shell product.

CODE EXAMPLE 4-1 Sample `install-openssh.fin` Script

```
#!/bin/sh
# NOTE: This script is not intended to be used for Solaris 9+.
  logMessage "Installing OpenSSH software.\n"
if check_os_revision 5.5.1 5.8 ; then
   OPENSSH_VERSION="2.5.2p2"
   OPENSSH_NAME="OBSDssh"
   OPENSSH_PKG_SRC="${OPENSSH_NAME}-${OPENSSH_VERSION}-`uname -p`
`uname -m`-`uname -r`.pkg"
   OPENSSH_PKG_DIR="${JASS_ROOT_DIR}/${JASS_PACKAGE_DIR}"
# Install the OpenSSH package onto the client
   if [ "${JASS_STANDALONE}" = "1" ]; then
      logNotice "This script cannot be used in standalone mode due
to the potential for overwriting the local OBSHssh installation."
   else
logMessage "Installing ${OPENSSH_NAME} from
${OPENSSH_PKG_DIR}/${OPENSSH_PKG_SRC}"
      if [ -f ${OPENSSH_PKG_DIR}/${OPENSSH_PKG_SRC} ]; then
         add_pkg -d ${OPENSSH_PKG_DIR}/${OPENSSH_PKG_SRC}
${OPENSSH_NAME} add_to_manifest X "pkgrm ${OPENSSH_NAME}"
      else
      logFileNotFound "${OPENSSH_NAME}"
[...]
```

In this case, the only way to adjust this script to support a different version of OpenSSH is to modify it directly. After completing the changes, be sure to change the security profile that uses this script, to account for its new name.

Note – As noted previously, this method of modifying a script directly should rarely be necessary, because most of the Solaris Security Toolkit software's functionality can be customized through variables.

Prevent `kill` Scripts From Being Disabled

Finish scripts that begin with the keyword `disable` are typically responsible for disabling services. Many of these scripts modify shell scripts that are located in the run-control directories (`/etc/rc*.d`). In most cases, run-control scripts are of two flavors: `start` and `kill` scripts. As their name implies, `start` scripts start services and `kill` scripts stop services. The `start` scripts begin with the capital letter `S` and `kill` scripts begin with the capital letter `K`.

`Kill` scripts are most often used to prepare a system for shutting down or rebooting. These scripts shut down services in a logical order so that changes are not lost and the system state is maintained. Typically, both `start` and `kill` scripts are hard links to files in the `/etc/init.d` directory, although this is not always the case.

The default action of the Solaris Security Toolkit software is to disable both `start` and `kill` scripts. This behavior can be altered using the `JASS_KILL_SCRIPT_DISABLE` environment variable. By default, this variable is set to 1, instructing the Solaris Security Toolkit software to disable both start and kill scripts.

There are times when this action is not preferred. For example, `kill` scripts are often used to stop services that were manually started by an administrator. If these scripts are disabled by the Solaris Security Toolkit software, then these services might not be stopped properly or in the correct sequence. To prevent `kill` scripts from being disabled, simply set the `JASS_KILL_SCRIPT_DISABLE` environment variable to `0` in the `user.init` file or in the relevant driver.

Create New Finish Scripts

You can create new finish scripts and integrate them into your deployment of the Solaris Security Toolkit software. Because finish scripts must be developed in Bourne shell, it is relatively easy to add new functionality. For those who are less experienced in UNIX shell scripting, examine existing finish scripts that perform similar functions to gain an understanding of how to accomplish a given task and to understand the correct sequence of actions.

Consider the following conventions when developing new finish scripts. Understanding these conventions ensures that the scripts are functional in standalone mode, JumpStart mode, and undo operations.

Whenever adding new finish scripts, be sure to consider adding a companion audit script. Audit scripts are used to determine the state of changes made on an existing system. For more information, see Chapter 5.

- Ensure that the finish script understands the relative `root` directory.

The scripts must not be configured to rely on the fact that the "/" directory is the actual root directory of the system. Incorrect configuration prevents the script from working in JumpStart mode when the target's actual root directory is "/a." This convention is easily implemented using the JASS_ROOT_DIR environment variable. For more information about this and other environment variables, see Chapter 6.

In some cases, the program used in a finish script might not support a relocated root directory. In these cases, it might be necessary to use the chroot(1M) command to force the command to run within a relative root directory, such as that described previously. For example, the usermod(1M) command does not allow the user to specify an alternate root directory. In this case, it is necessary to use the chroot(1M) command as follows.

```
chroot ${JASS_ROOT_DIR} /usr/sbin/usermod ...arguments...
```

The Solaris Security Toolkit software automatically detects the location of the platform's real root directory and assigns that value to the JASS_ROOT_DIR variable. Use this variable in place of hard-coding a specific path for the root file system. For example, in place of using /etc/inet/inetd.conf within the finish script, use JASS_ROOT_DIR/etc/inet/inetd.conf.

- Where possible, use the Solaris Security Toolkit software's framework when creating new directories, copying files, or backing up existing files.

  Using the framework functions ensures that the changes made by a new script are consistent with those done elsewhere, and that they can be safely undone. For a list of framework functions, see Chapter 1.

  Examples of framework functions that are compatible with undo are as follows:

  - disable_rc_file
  - disable_conf_file
  - backup_file
  - create_a_file

- Wherever possible, attempt to use standard, supportable ways to configure or tune a system.

  For example, programs like usermod(1M) are preferred over directly modifying the /etc/passwd file. This preference is necessary to make the software as flexible as possible and to make the resulting finish scripts as OS-version independent as possible. Also, complicated or obscure ways of configuring a system could actually be harder to debug or maintain over the life of a script. For an example of methods on supportable ways in which changes can be made, refer to the Sun BluePrints OnLine article titled "Solaris Operating Environment Security: Updated for Solaris Operating Environment 9."

- Make sure that new finish scripts are OS-version aware.

If a particular function is not needed on a version of the OS, then do not attempt to use it. This approach helps to make the software backward compatible with existing releases and more likely to support future releases. Furthermore, by making finish scripts OS-version aware, the number of warning and error messages can be dramatically reduced. The Solaris Security Toolkit software's finish directory contains example scripts that are aware of the OS on which they are being used and that only make changes when necessary. Some sample scripts that use this capability are as follows:

- `enable-rfc1948.fin`
- `install-ftpusers.fin`

To make this process simpler for software developers, the framework includes the following two functions:

- `check_os_min_revision`
- `check_os_revision`

For detailed information about these functions, see Chapter 1.

- A final consideration when developing or customizing finish scripts is that the Solaris Security Toolkit software could be run more than once on a single platform.

  The finish scripts must be able to detect whether a change actually needs to be made.

  For example, the `enable-rfc1948.fin` script checks to see if the `/etc/default/inetinit` script already has the setting `TCP_STRONG_ISS=2`. If this setting is present, there is no need to back up files or make other changes.

```
if [ `grep -c "TCP_STRONG_ISS=2" ${INETINIT}` = 0 ]; then
# The following command will remove any exiting TCP_STRONG_ISS
# value and then insert a new one where TCP_STRONG_ISS is set
# to 2.  This value corresponds to enabling RFC 1948
# unique-per-connection ID sequence number generation.
logMessage "\nSetting 'TCP_STRONG_ISS' to '2' in ${INETINIT}.\n"
backup_file ${INETINIT}
cat ${INETINIT}.${JASS_SUFFIX} |\
sed '/TCP_STRONG_ISS=/d' > ${INETINIT}
echo "TCP_STRONG_ISS=2" >> ${INETINIT}
fi
```

This technique not only reduces the number of unnecessary backup files, but it helps prevent errors and confusion resulting from multiple, redundant changes made in the same files. Also, by implementing this functionality, you are well on your way toward developing the code necessary to implement the finish script's companion audit script.

Using Standard Finish Scripts

Finish scripts perform system modifications and updates during hardening runs. These scripts are not used in any other runs or operations of the software.

The `finish.init` handles all finish script configuration variables. You can override the default variables by modifying the `user.init` file. This file is heavily commented to explain each variable, its impact, and its use in finish scripts. Additionally, see Chapter 6 for a description of each variable.

Using variables found in the `finish.init` script, you can customize most of the finish scripts to suit your organization's security policy and requirements. You can customize nearly every aspect of the Solaris Security Toolkit software through variables, without needing to alter the source code. The use of this script is strongly recommended so as to minimize migration issues with new Solaris Security Toolkit software releases.

This section describes the standard finish scripts, which are in the `Finish` directory. Each of the scripts in the `Finish` directory is organized into the following categories:

- Disable
- Enable
- Install
- Minimize
- Print
- Remove
- Set
- Update

In addition to these standard finish scripts, the Solaris Security Toolkit software provides product-specific finish scripts. For a list of product-specific finish scripts, see "Using Product-Specific Finish Scripts" on page 120.

Disable Finish Scripts

The following disable finish scripts are described in this section:

disable-ab2.fin

This script prevents the AnswerBook2™ (ab2) server from starting. The ab2 server software is distributed on the Documentation CD in the Solaris OS Server pack. This script applies only to systems running Solaris OS versions 2.5.1 through 8, because the ab2 software is no longer used in Solaris OS version 9.

disable-apache.fin

This script prevents the Apache Web Server, shipped with Solaris OS versions 8 and 9, from starting. This script disables only the Apache services included in the Solaris OS Distribution package. This script does not impact other Apache distributions installed on the system. For more information on this service, refer to the `apache`(1M) manual page.

disable-asppp.fin

This script disables the Asynchronous Point-to-Point Protocol (ASPPP) service from starting. This service implements the functionality described in Remote Function Call (RFC) 1331, The Point-to-Point Protocol (PPP) for the transmission of multi-protocol datagrams over Point-to-Point links. This script applies only to Solaris OS versions 2.5.1 through 8. For the Solaris 9 OS, this service has been replaced with the PPP service and is disabled using the `disable-ppp.fin` finish script. For more information on this functionality, refer to the `aspppd`(1M) manual page.

disable-autoinst.fin

Caution – Do not use the `disable-autoinst.fin` script if there might be a need to use the functionality provided by the `sys-unconfig`(1M) program to restore a system's configuration to an as-manufactured state.

Caution – If you are using a a JumpStart environment, disable the run-control, or startup, scripts mentioned in the following paragraph to help prevent an intruder from reconfiguring the system.These run-control scripts are never used in a JumpStart environment.

This script prevents a system from being re-installed, by disabling the run-control scripts associated with automatic configuration. These scripts are used only if the `/etc/.UNCONFIGURED` or `/AUTOINSTALL` files are created. After initial installation and configuration, there is generally little reason for these scripts to remain available.

disable-automount.fin

Note – Because this service relies on the Remote Procedure Call (RPC) port mapper, if `disable-automount.fin` is *not* used, then the `disable-rpc.fin` script should *not* be used either.

This script disables the NFS automount service. The automount service answers file system mount and unmount requests from the `autofs` file system. When this script is used, the NFS automount service is disabled and all forms of automount maps are affected. For more information on this functionality, refer to the `automountd`(1M) manual page.

disable-dhcp.fin

This script disables the Dynamic Host Configuration Protocol (DHCP) Server included in Solaris OS versions 8 and 9. For more information on this server, refer to the `dhcpd`(1M) manual page.

disable-directory.fin

This script prevents the Sun Java System Directory Server, formerly the Sun ONE Directory Server, (bundled with the Solaris 9 OS) from starting. Note that this script is for use only with the Sun Java System Directory Server. This script does not affect either the unbundled product or the Sun Java System Directory Server software provided with other Solaris OS versions. By default, the Solaris Security Toolkit software disables only the services supplied with the Solaris OS. For more information on this server, refer to the `directoryserver`(1M) manual page.

disable-dmi.fin

This script prevents the Desktop Management Interface (DMI) from starting. This script applies only to systems running Solaris OS versions 2.6 through 9. For more information on this service, refer to the `dmispd`(1M) and `snmpXdmid`(1M) manual pages.

disable-dtlogin.fin

Note – Because this service relies on the RPC port mapper, if `disable-rpc.fin` is *not* used, then the `disable-dtlogin.fin` script should *not* be used either.

This script prevents any windowing environment from being started at boot time, for example, the Common Desktop Environment (CDE) service. However, this script does not prevent a windowing environment from being started at a later time (for example, after a system is booted). This script applies only to systems running the Solaris OS versions 2.6 through 9. For more information on this service, refer to the `dtlogin`(1X) and `dtconfig`(1) manual pages.

disable-ipv6.fin

This script disables the use of IPv6 on specific network interfaces by removing the the associated host name files in `/etc/hostname6.*`. Also, this mechanism prevents the `in.ndpd` service from running. This script applies only to systems running the Solaris OS versions 8 and 9. This script should not be used if IPv6 functionality is required on the system.

disable-kdc.fin

This script prevents the Kerberos Key Distribution Center (KDC) service from starting. Note that if `JASS_DISABLE_MODE` is set to `conf`, the `kdc.conf` file is disabled, thus impacting the ability to act as a Kerberos client. This script should not be used in that manner if the system must act as a Kerberos client. This script applies only to systems running the Solaris OS version 9. For more information on this service, refer to the `krb5kdc`(1M) and `kdc.conf`(4) manual pages.

disable-keyboard-abort.fin

Note – Some systems feature key switches with a secure position. On these systems, setting the key switch to the secure position overrides any software default set with this command.

This script configures the system ignore keyboard abort sequences. Typically, when a keyboard abort sequence is initiated, the operating system is suspended and the console enters the OpenBoot™ PROM monitor or debugger. Using this script prevents the system from being suspended. For more information on this capability, refer to the `kbd`(1) manual page. This script is used only in the Solaris OS versions 2.6 through 9.

disable-keyserv-uid-nobody.fin

This script disables the nobody UID access to secure RPC. In Solaris 9 OS, access is disabled by setting the `ENABLE_NOBODY_KEYS` variable in the `/etc/init.d/rpc` to `NO`. For versions earlier than Solaris OS version 9, access is disabled by adding the `-d` option to the `keyserv` command in the `/etc/init.d/rpc` run-control file. For more information on this service, refer to the `keyserv`(1M) manual page.

disable-ldap-client.fin

This script prevents the Lightweight Directory Access Protocol (LDAP) client daemons from starting on the system. This service provides the directory lookup capability for the system. If the system is acting as an LDAP client or requires the directory lookup capability, then this script should not be used. This script applies to Solaris OS versions 8 and 9. For more information on this service, refer to the `ldap_cachemgr`(1M) and `ldapclient`(1M) manual pages.

disable-lp.fin

This script prevents the line printer (`lp`) service from starting. Note that in addition to disabling the service, this script removes the `lp` user's access to the `cron` subsystem by adding `lp` to the `/etc/cron.d/cron.deny` file, and removing all `lp` commands in the `/var/spool/cron/crontabs` directory.

This functionality is distinct from the `update-cron-deny.fin` script, because the `lp` packages might or might not be installed on a system. In addition, the `lp` subsystem might be necessary, while the functions removed by the `cron-deny-update.fin` script are not.

disable-mipagent.fin

This script prevents the Mobile Internet Protocol (MIP) agents from starting. This service implements the MIP home agent and foreign agent functionality described in RFC 2002, IP Mobility Support. This script applies only to Solaris OS versions 8 and 9. For more information on this service, refer to the `mipagent`(1M) manual page.

disable-named.fin

Note – Disabling this service does *not* affect the ability of the system to act as a Domain Name System (DNS) client.

This script prevents the DNS server from starting using the `named`(1M) command. Note that this script is intended to be used only with the DNS service shipped with the Solaris OS.

`disable-nfs-client.fin`

Note – If this service is required, then this script should *not* be used. Further, because this service relies on the RPC service, the `disable-rpc.fin` script also should *not* be used.

This script prevents the NFS client service from starting. Also, this disables the network status monitor (`statd`) and lock manager (`lockd`) daemons. Note that an administrator can still mount remote file systems onto the system, even if this script is used. Those file systems, however, do not take advantage of the status monitor or lock manager daemons. For more information on this service, refer to the `statd`(1M) and `lockd`(1M) manual pages.

`disable-nfs-server.fin`

This script prevents the NFS service from starting. Also, this script disables the daemons that provide support for NFS logging, mounting, access checks, and client service. Do not use this script if the system must share its file systems with remote clients. For more information on this service, refer to the `nfsd`(1M), `mountd`(1M), and `dfstab`(4) manual pages.

Note – If this service is required, then this script should *not* be used. Further, because this service relies on the RPC service, the `disable-rpc.fin` script also should *not* be used.

`disable-nscd-caching.fin`

Caution – There might be a performance impact on systems that use name services intensively.

This script disables caching for `passwd`, `group`, `hosts`, and `ipnodes` entries by the Name Service Cache Daemon (NSCD). For the Solaris 8 OS, patch 110386 version 02 at minimum must be applied to fix a bug in the Role-Based Access Control (RBAC) facility, otherwise the Solaris Security Toolkit software generates an error message.

The NSCD provides caching for name service requests. It exists to provide a performance boost to pending requests and reduce name service network traffic. The `nscd` maintains cache entries for databases such as `passwd`, `group`, and `hosts`. It does not cache the shadow password file for security reasons. All name service requests made through system library calls are routed to `nscd`. With the addition of IPv6 and RBAC in Solaris 8 OS, the `nscd` caching capability was expanded to address additional name service databases.

Because caching name service data makes spoofing attacks easier, it is recommended that the configuration of `nscd` be modified to cache as little data as possible. This task is accomplished by setting the positive time-to-live (`ttl`) to zero in the `/etc/nscd.conf` file for the name service requests deemed vulnerable to spoofing attacks. In particular, the configuration should be modified so that `passwd`, `group`, and Solaris 8 and 9 OS RBAC information has a positive and negative `ttl` of zero.

The `nscd -g` option can be used to view the current `nscd` configuration on a server and is a helpful resource when tuning `nscd`.

Disabling `nscd` entirely is not recommended because applications make name service calls directly, which exposes various bugs in applications and name service backends.

disable-picld.fin

This script prevents the Platform Information and Control Library (PICL) service from starting. Disabling this service could impact the ability of the system to monitor environmental conditions and should, therefore, be used with care. This script applies only to systems running Solaris OS versions 8 and 9. For more information on this service, refer to the `picld`(1M) manual page.

disable-power-mgmt.fin

This script prevents the power management service from starting. (This service allows the system to power down monitors, spin down disks, and even power off the system itself.) Using this script disables the power management functionality. Additionally, a `noautoshutdown` file is created to prevent a system administrator from being asked about the state of power management during an automated JumpStart mode installation. This script applies only to systems running Solaris OS versions 2.6 through 9. For more information on this service, refer to the `powerd`(1M), `pmconfig`(1M), and `power.conf`(4) manual pages.

disable-ppp.fin

This script prevents the Point-to-Point Protocol (PPP) service from starting. This service was introduced in the Solaris 8 OS (7/01) and supplements the older Asynchronous PPP (ASPPP) service. This service provides a method for transmitting datagrams over serial point-to-point links. This script applies only to systems running the Solaris OS versions 8 and 9. For more information on this service, refer to the `pppd`(1M) and `pppoed`(1M) manual pages.

disable-preserve.fin

This script prevents the moving of saved files (that were previously edited) to `/usr/preserve` when a system is rebooted. These files are typically created by editors that are abruptly terminated due to a system crash or loss of a session. These files are normally located in `/var/tmp` with names beginning with "`Ex`".

disable-remote-root-login.fin

This script changes the `CONSOLE` variable in the `/etc/default/login` file to prevent direct remote `root` logins. Although this was the default behavior for the Solaris OS since the final update of 2.5.1, it is included to ensure that this setting has not been altered. Note that this setting has no impact on programs, such as Secure Shell, that can be configured to not use the `/bin/login` program to grant access to a system. For more information on this capability, refer to the `login`(1) manual page.

disable-rhosts.fin

This script disables `rhosts` authentication for `rlogin` and `rsh` by modifying the Pluggable Authentication Module (PAM) configuration in `/etc/pam.conf`.

The `disable-rlogin-rhosts.fin` finish script was renamed `disable-rhosts.fin` to be more indicative of its actions. In addition, both `rsh` and `rlogin` entries are commented out in the `/etc/pam.conf` file to ensure that `rhosts` authentication is not enabled for either service.

This script applies only to Solaris OS versions 2.6 through 9. For more information on this capability, refer to the `in.rshd`(1M), `in.rlogind`(1M) and `pam.conf`(4) manual pages.

disable-rpc.fin

Note – The RPC port mapper function should not be disabled if any of the following services are used on the system: automount, NFS, Network Information Services (NIS), NIS+, CDE, and volume management (Solaris OS version 9 *only*).

This script prevents the remote procedure call (RPC) service from starting. Note that disabling this service impacts bundled services such as NFS and CDE, and unbundled services such as Sun Cluster. Also, some third-party software packages expect that this service is available. Before disabling this service, verify that no services or tools require RPC services. For more information on this service, refer to the `rpcbind`(1M) manual page.

disable-samba.fin

This script prevents the Samba file and print sharing service from starting. This script disables only the Samba services included in the Solaris OS distribution. This script does not impact other Samba distributions installed on the system. For more information on this service, refer to the `smbd`(1M), `nmbd`(1M), and `smb.conf`(4) manual pages.

disable-sendmail.fin

Note – The Solaris Security Toolkit software modifications only prevent a Solaris OS from receiving email. Outgoing email is still processed normally.

This script disables the `sendmail` daemon startup and shutdown scripts, and adds an entry to the `cron` subsystem, which executes `sendmail` once an hour for Solaris OS versions 2.5.1, 2.6, and 7.

For Solaris 8 OS, the `/etc/default/sendmail` file is installed, which implements similar functionality. This method of purging outgoing mail is more secure than having the daemon run continually.

Solaris 9 OS implements another `sendmail` option in which the daemon only listens on the loopback interface. For more information, refer to the Sun BluePrints OnLine article titled "Solaris Operating Environment Security: Updated for Solaris Operating Environment 9."

disable-slp.fin

This script prevents the Service Location Protocol (SLP) service from starting. This service provides common server functionality for the SLP versions 1 and 2, as defined by the Internet Engineering Task Force (IETF) in RFC 2165 and RFC 2608. SLP provides a scalable framework for the discovery and selection of network services. This script applies only to systems running the Solaris OS versions 8 and 9. For more information on this service, refer to the `slpd`(1M) manual page.

disable-sma.fin

This script prevents the System Management Agent (SMA, based on NET-SNMP) service from starting.

disable-snmp.fin

This script prevents the Simple Network Management Protocol (SNMP) service from starting. This script does not prevent third-party SNMP agents from functioning on the system. This script only affects the SNMP agent provided in the Solaris OS. This script applies only to systems running the Solaris OS versions 2.6 through 9. For more information on this service, refer to the `snmpdx`(1M) and `mibiisa`(1M) manual pages.

disable-spc.fin

This script disables all SunSoft™ Print Client startup scripts. This script applies only to systems running the Solaris OS versions 2.6 through 9.

disable-ssh-root-login.fin

This script configures the Secure Shell service distributed in the Solaris 9 OS to restrict remote access to the `root` account. By default, remote `root` access is denied using the version of Secure Shell shipped with the Solaris 9 OS. This script verifies that functionality, thereby implementing a mechanism similar to that of the `disable-remote-root-login.fin` script. The script sets the `PermitRootLogin` parameter in `/etc/ssh/sshd_config` to `no`. For more information on this capability, refer to the `sshd_config`(4) manual page.

disable-syslogd-listen.fin

Note – Do no use this script on a SYSLOG server, because a SYSLOG server's function is to log remotely.

This script prevents the log system messages (syslogd) service from accepting remote log messages. For Solaris OS versions prior to Solaris 9 OS, this scripts adds the -t option to the syslogd(1M) command line. For Solaris 9 OS, this script sets the LOG_FROM_REMOTE variable to NO in the /etc/default/syslogd file. Note that this script prevents the daemon from listening on User Diagram Protocol (UDP) port 514. This script is useful for systems that use system log services and do not need to receive system log messages from remote systems.

disable-system-accounts.fin

This script disables specific unused system accounts other than root. The list of accounts that will be disabled on the system are explicitly enumerated in the JASS_ACCT_DISABLE variable.

disable-uucp.fin

This script disables the UNIX-to-UNIX Copy (UUCP) startup script. In addition, the nuucp system account is removed with the uucp crontab entries in the /var/spool/cron/crontabs directory. For more information on this service, refer to the uucp(1C) and uucico(1M) manual pages.

disable-vold.fin

Note – Do *not* use this script if you need the automatic mounting and unmounting of removable media (such as diskettes and CD-ROMs).

Note – Do *not* use this script if this service is required in Solaris OS version 9. Further, because this service relies on both the RPC and the rpc.smserverd services, do *not* disable them either. To prevent the RPC service from being disabled, do not use the disable-rpc.fin script. Similarly, to prevent the rpc.smserverd service from being disabled, add 100155, which represents the rpc.smserverd service, to the JASS_SVCS_ENABLE parameter.

This script prevents the Volume Management Daemon (VOLD) from starting. The `vold` creates and maintains a file system image rooted at `/vol`, by default, that contains symbolic names for diskettes, CD-ROMs, and other removable media devices. For more information on this service, refer to the `vold`(1M) manual page.

disable-wbem.fin

Note – If this service is required, then do *not* use this script. Also, because this service relies on the RPC service, the `disable-rpc.fin` script should *not* be used.

This script prevents the Web-Based Enterprise Management (WBEM) service from starting. The WBEM is a set of management and Internet-related technologies that unify management of enterprise computing environments. Developed by the Distributed Management Task Force (DMTF), the WBEM enables organizations to deliver an integrated set of standards-based management tools that support and promote World Wide Web technology. Do not use this script if the use of Solaris Management Console is needed. This script applies only to systems running the Solaris OS versions 8 and 9. For more information on this service, refer to the `wbem`(5) manual page.

disable-xserver.listen.fin

This script disables the X11 server's ability to listen to and accept requests over TCP on port 6000. This script adds the option `-nolisten` TCP to the X server configuration line in the `/etc/dt/config/Xservers` file. If this file does not exist, it is copied from the master location at `/usr/dt/config/Xservers`. This script is applicable only to the Solaris 9 OS. For more information on this capability, refer to the `Xserver`(1) manual page.

Enable Finish Scripts

The following enable finish scripts are described in this section:

enable-32bit-kernel.fin

This script sets the `boot-file` variable in the EEPROM of Sun SPARC systems to the value of `/kernel/unix`. This setting forces the system to boot using a 32-bit kernel. It is useful for products that can run on the Solaris OS versions 7, 8, and 9, but must run in 32-bit-only mode. This script applies only to Sun4U™ systems. This script is included as a convenience for environments using applications that support only 32-bit mode OS.

enable-bsm.fin

This script enables the SunSHIELD™ Solaris Basic Security Module (BSM) auditing service. Additionally, this script installs a default audit configuration that is described in the Sun BluePrints OnLine article titled "Auditing in the Solaris 8 Operating Environment." An `audit_warn` alias is added, if necessary, and assigned to the `root` account. And, the abort disable code is overridden to permit abort sequences. This setting is most often used in a lights-out data center environment, where physical access to the platform is not always possible. After the system is rebooted, the Solaris BSM subsystem is enabled and auditing begins. For more information on this service, refer to the `bsmconv`(1M) manual page.

enable-coreadm.fin

This script configures the `coreadm` functionality that is present in the Solaris OS versions 7, 8, and 9. It configures the system to store generated core files under the directory specified by `JASS_CORE_DIR`. Further, each of the core files are tagged with a specification denoted by the `JASS_CORE_PATTERN` so that information about

the core files can be collected. Typically, the information collected includes the process identifier, effective user identifier, and effective group identifiers of the process, as well as name of the process executable and time the core file was generated. For more information on this capability, refer to the `coreadm`(1M) manual page.

enable-ftp-syslog.fin

This script forces the `in.ftpd` daemon to log all File Transfer Protocol (FTP) access attempts through the `SYSLOG` subsystem. This option is enabled by adding the `-l` option to the `in.ftpd` command in the `/etc/inetd/inetd.conf` file. For more information, refer to the `in.ftpd`(1M) manual page.

enable-ftpaccess.fin

This script enables the `ftpaccess` functionality for the FTP service in Solaris 9 OS. This functionality is necessary so that security modifications made by the `set-banner-ftp.fin` and `set-ftpd-umask.fin` scripts are used. For example, modifications to set the default greeting, file creation mask, and other parameters documented in `ftpaccess`(4) manual pages. This script adds the `-a` argument to the `in.ftpd` entry in the `/etc/inet/inetd.conf` file. This script applies only to systems running the Solaris 9 OS.

enable-inetd-syslog.fin

This script configures the Internet services daemon (INETD) to log all incoming TCP connection requests. That is, a log entry occurs via SYSLOG if a connection is made to any TCP service for which the `inetd` daemon is listening. For Solaris OS versions prior to Solaris 9 OS, this script enables logging by adding the `-t` option to the `inetd` command line. In Solaris 9 OS, the script sets the `ENABLE_CONNECTION_LOGGING` variable in the `/etc/default/inetd` file to `YES`.

enable-priv-nfs-ports.fin

Note – If the key word value pair is already defined in the `/etc/system` file, the value is rewritten in the file to verify that it is set properly. Otherwise, the keyword value pair is appended to the file.

This script modifies the `/etc/system` file to enable restricted NFS port access. After setting the variable, only NFS requests originating from ports less than 1024 are accepted.

enable-process-accounting.fin

If the required Solaris OS packages (currently SUNWaccr and SUNWaccu) are installed on the system, this script enables Solaris OS process accounting. For more information on this service, refer to the acct(1M) manual page.

enable-rfc1948.fin

This script creates or modifies the /etc/default/inetinit file to enable support of RFC 1948. (This RFC defines unique-per-connection ID sequence number generation.) The script sets the variable TCP_STRONG_ISS to 2 in the /etc/default/inetinit file. For more information, refer to http://ietf.org/rfc1948.html. This script applies only to systems running the Solaris OS versions 2.6 through 9.

enable-stack-protection.fin

Note – After the system is rebooted with these variables set, the system denies attempts to execute the stack directly, and logs any stack execution attempt through SYSLOG. This facility is enabled to protect the system against common buffer overflow attacks.

For SPARC systems only, this script modifies the /etc/system file to enable stack protections and exception logging. These options are enabled by adding the noexec_user_stack and noexec_user_stack_log to the /etc/system file.

If the key word value pairs are already defined in the /etc/system file, their values are rewritten in the file to verify that they are set properly. Otherwise, the keyword value pairs are appended to the file. This script applies only to systems running the Solaris OS versions 2.6 through 9. Enabling this feature makes the system noncompliant with the SPARC version 8 Application Binary Interface (ABI), therefore, it is possible that some applications might fail.

In Solaris OS version 9, many of the core Solaris executables are linked against a map file (/usr/lib/ld/map.noexstk). This map file provides functionality similar to the script by making the program's stack non-executable. Using the script is still recommended, however, because its changes are global to the system.

enable-tcpwrappers.fin

Note – The sample hosts.allow and hosts.deny files should be customized prior to their use to ensure that their configuration is appropriate for your organization. File templates are available in JASS_ROOT_DIR/Files/etc.

This script configures the system to use TCP wrappers. Included with the Solaris 9 OS, TCP wrappers allow an administrator to restrict access to TCP services. By default, all services in /etc/inet/inetd.conf that are defined as stream, nowait are protected. This script configures the /etc/default/inetd file to set the ENABLE_TCPWRAPPERS parameter to YES. Further, this script installs sample /etc/hosts.allow and /etc/hosts.deny files that control access to services protected by TCP wrappers.

Install Finish Scripts

The following install finish scripts are described in this section:

install-at-allow.fin

This script restricts the at command execution by creating an at.allow file in /etc/cron.d. The file is then populated with the list of users defined in the JASS_AT_ALLOW variable. All users who require at access must be added to the

at.allow file. This script should be used with the update-at-deny.fin script to determine access to the at and batch facilities. For more information on this capability, refer to the at(1) manual page.

install-fix-modes.fin

Note – Although the changes implemented by the FixModes software are integrated into the Solaris 9 OS, the use of FixModes is still recommended because many unbundled and third-party applications benefit from its use.

This script both copies the fix-modes software from the JASS_PACKAGE_DIR directory to the client, then executes the program. (The FixModes software was created by Casper Dik; see "Related Resources" on page xxix.) Use the FixModes software to tighten permissions of a Solaris system.

install-ftpusers.fin

This script creates or modifies the ftpusers file that is used to restrict access to the FTP service. This script adds users listed in the JASS_FTPUSERS variable to the ftpusers file. This script only adds a user to the file if the user's name is not already in the file.

A default ftpusers file is included with Solaris OS versions 8 and 9. The path to the file varies. For Solaris OS versions 8 and lower, the file path is /etc. For Solaris 9 OS, the path is /etc/ftpd. All accounts *not* allowed to use the incoming FTP service should be specified in this file. At a minimum, this should include all system accounts (for example, bin, uucp, smtp, sys, and so forth) in addition to the root account. These accounts are often targets of intruders and individuals attempting to gain unauthorized access. Frequently, root access to a server over Telnet is disabled and root FTP access is not. This configuration provides a back door for intruders who might modify the system's configuration by uploading modified configuration files.

install-Sun_ONE-WS.fin

This script installs the Sun Java System Web Server software onto the target platform. This script is provided as a sample of how software installation can be automated using JumpStart technology. For additional information, refer to the Sun BluePrints OnLine article titled "Minimizing the Solaris Operating Environment."

install-jass.fin

This script automates the installation of the Solaris Security Toolkit software onto a JumpStart client when the Solaris Security Toolkit software is being run. Use this approach so that the Solaris Security Toolkit software is available to be run after patch installations on the client. The installation is performed by installing the Solaris Security Toolkit software package distribution with the Solaris OS command `pkgadd`. This script expects the Solaris Security Toolkit software to be installed in the `JASS_PACKAGE_DIR` directory. The Solaris Security Toolkit software package is installed by default in `/opt/SUNWjass`.

install-loginlog.fin

This script creates the `/var/adm/loginlog` file used by the system to log unsuccessful login attempts. The failed log ins are logged after the maximum number of failed log ins is exceeded. This number is specified in the `RETRIES` variable, set in the `/etc/default/login` configuration file. See also the `set-login-retries.fin` script. For more information, refer to the `loginlog`(4) manual page.

install-md5.fin

This script automates the installation of the message-digest 5 (MD5) algorithm software. This software is used for creating digital fingerprints of file system objects and is referenced in the Sun BluePrints OnLine article titled "The Solaris Fingerprint Database - A Security Tool for Solaris Software and Files." By default, the MD5 software is installed in the directory specified by the `JASS_MD5_DIR` parameter.

install-nddconfig.fin

This script installs the `nddconfig` file that is used to set more secure values for various networking parameters, based on the Sun BluePrints OnLine article, "Solaris Operating Environment Network Settings for Security."

install-newaliases.fin

This script adds the `newaliases` symbolic link to the `/usr/lib/sendmail` program. This link is required in some cases of minimized installations if the `SUNWnisu` package is not installed or is removed. This link is necessary for systems running the Solaris OS versions 2.5.1 through 8, where the `newaliases` was a part of the `SUNWnisu` package.

install-openssh.fin

Note – Solaris 9 OS includes a version of the Secure Shell software, therefore this script is not used if you install Solaris 9 OS.

This script installs the OpenBSD version of OpenSSH into `/opt/OBSDssh`. The distribution for which this script is written is based on the Sun BluePrints OnLine article titled "Configuring OpenSSH for the Solaris Operating Environment." This script does not overwrite host keys if they exist.

The installation is based on having a Solaris OS, stream-formatted package called `OBSDssh-3.5p1-sparc-sun4u-5.8.pkg` in the `JASS_PACKAGE_DIR` directory.

install-recommended-patches.fin

This script installs applicable patches from the `JASS_HOME_DIR/Patches` directory on the JumpStart server. The appropriate Recommended and Security Patch Clusters must be downloaded and extracted to the `JASS_HOME_DIR/Patches` directory for the script to execute properly.

install-sadmind-options.fin

This script adds the options specified in the `JASS_SADMIND_OPTIONS` environment variable to the `sadmind` daemon entry in `/etc/inet/inetd.conf`. For more information on this service, refer to the `sadmind`(1M) manual page.

install-security-mode.fin

This script displays the current status of the OpenBoot PROM security mode. This script does not set the EEPROM password directly; it is not possible to script the setting of the EEPROM password during a JumpStart installation. The output of the script provides instructions on how to set the EEPROM password from the command line. This script applies only to systems based on SPARC technology. For more information on this capability, refer to the `eeprom`(1M) manual page.

install-shells.fin

Note – This script only adds a shell to the `/etc/shells` file if the shell exists on the system, is executable, and is not in the file.

This script adds the user shells specified in the JASS_SHELLS environment variable to the /etc/shells file. The Solaris OS function getusershell(3C) is the primary user that the /etc/shells file uses to determine valid shells on a system. For more information, refer to the shells(4) manual page. For more information about the JASS_SHELLS environment variable, see "JASS_SHELLS" on page 194.

install-strong-permissions.fin

Caution – Exercise care when using this script, because its changes cannot be undone automatically by the Solaris Security Toolkit software. Always ensure that the permissions set by this script are appropriate for your environment and applications.

This script changes a variety of permissions and ownerships to enhance security by restricting group and user access on a system.

install-sulog.fin

This script creates the /var/adm/sulog file, which enables logging of all superuser (su) attempts. For more information on this capability, refer to the sulog(4) manual page.

install-templates.fin

This special purpose script should not be called directly by any driver. This script is automatically called by the driver.run program if the JASS_FILES parameter or any of its OS-specific values is not empty. This script automates the copying of file templates onto a target system. This functionality was originally in the driver.run script, but was separated to better support the verification of file templates. If needed, based on the contents of the JASS_FILES parameter, this script is the first finish script to run.

Minimize Finish Script

The minimize-Sun_ONE-WS.fin script is provided as an example of how the Solaris OS minimization procedure can be implemented. In this case, this script is used to minimize a system that is used as a web server running the Sun Java System Web Server software.

Print Finish Scripts

The following print finish scripts are described in this section:

print-jass-environment.fin

This script prints out all the environment variables used in the Solaris Security Toolkit software. This script is provided for diagnostic purposes and is often called at the beginning of a driver so that the state of the environment variables can be recorded prior to their use.

print-jumpstart-environment.fin

This script prints out all the environment variables used by a JumpStart installation. This script is provided for diagnostic purposes to aid in debugging problems encountered during a JumpStart installation.

print-rhosts.fin

This script lists all the `.rhosts` and `hosts.equiv` files contained in any directory under the `JASS_ROOT_DIR` directory. The results are displayed on standard output unless the `JASS_RHOSTS_FILE` variable is defined. If this variable is defined, then all of the results are written to that file.

print-sgid-files.fin

This script prints all files in any directory under the `JASS_ROOT_DIR` directory with set group ID permissions. The results are displayed on standard output unless the `JASS_SGID_FILE` variable is defined. If this variable is defined, all of the results are written to that file.

print-suid-files.fin

This script prints all files in any directory under the JASS_ROOT_DIR directory with set user ID permissions. The results are displayed on standard output unless the JASS_SUID_FILE variable is defined. If this variable is defined, all of the results are written to that file.

print-unowned-objects.fin

This script lists all files, directories, and other objects on a system, starting from JASS_ROOT_DIR, that do not have valid users or groups assigned to them. The results are displayed on standard output unless the JASS_UNOWNED_FILE variable is defined.

print-world-writable-objets.fin

This script lists all world-writable objects on a system, starting from JASS_ROOT_DIR. The results are displayed on standard output unless the JASS_WRITABLE_FILE variable is defined. If this variable is defined, then all of the results are written to that file.

Remove Finish Script

The remove-unneeded-accounts.fin script removes unused Solaris OS accounts from the /etc/passwd and /etc/shadow files using the passmgmt command. This script removes those accounts defined by the JASS_ACCT_REMOVE variable.

Set Finish Scripts

The following set finish scripts are described in this section:

set-banner-dtlogin.fin

This script installs a service banner for the `dtlogin` service. This banner is presented to a user after successfully authenticating to a system using a graphical interface, such as is provided by the Common Desktop Environment (CDE). This script configures the system to display the contents of a file specified by the file template `JASS_ROOT_DIR/etc/dt/config/Xsession.d/0050.warning`. By default the contents of the `/etc/motd` file are displayed. This script applies only to systems running the Solaris OS versions 2.6 through 9.

set-banner-ftpd.fin

This script installs the File Transfer Protocol (FTP) service banner defined by the variable `JASS_BANNER_FTPD`. For Solaris OS 8 and earlier versions, this banner is defined using the `BANNER` variable in the `/etc/default/ftpd` file. For the Solaris OS version 9, this banner is defined using the `/etc/ftpd/banner.msg` file. For more information, refer to the `in.ftpd`(1M) or `ftpaccess`(4) (for Solaris 9 OS) manual pages. This script applies only to systems running the Solaris OS versions 2.6 through 9.

If the `install-ftpaccess.fin` script is not used, then the change made by this script on a Solaris OS version 9 system does not take effect.

set-banner-telnet.fin

This script installs the Telnet service banner defined by the variable `JASS_BANNER_TELNET`. This banner is defined using the `BANNER` variable in the `/etc/default/telnetd` file. For more information, refer to the `in.telnetd`(1M) manual page. This script applies only to systems running the Solaris OS versions 2.6 through 9.

set-banner-sendmail.fin

This script installs the Sendmail service banner defined by the variable JASS_BANNER_SENDMAIL. This banner is defined using the SmtpGreetingMessage or De parameter in the /etc/mail/sendmail.cf file. For Solaris OS versions 7, 8, and 9, the SmtpGreetingMessage parameter is used. For earlier releases, the De parameter is used to implement this functionality. For more information, refer to the sendmail(1M) manual page.

set-banner-sshd.fin

This script installs the Secure Shell service banner by configuring the Secure Shell service to display the contents of /etc/issue to the user prior to authenticating to the system. This task is accomplished by setting the Banner parameter to /etc/issue in the /etc/ssh/sshd_config file. For more information on this functionality, refer to the sshd_config(4) manual page. This script is used only for systems running the Solaris OS version 9 .

set-ftpd-umask.fin

This script sets the default file creation mask for the FTP service. In versions prior to Solaris 9 OS, the script sets the default file creation mask by adding a UMASK value, defined by the JASS_FTPD_UMASK variable, to the /etc/default/ftpd file. For Solaris 9 OS, the script sets the defumask parameter defined in the /etc/ftpd/ftpaccess file. For more information, refer to the in.ftpd(1M) or ftpaccess(4) (for Solaris 9 OS) manual pages. This script applies only to systems running the Solaris OS versions 2.6 through 9.

If the install-ftpaccess.fin script is not used, then the change made by this script on a Solaris OS version 9 system does not take effect.

set-login-retries.fin

This script sets the RETRIES variable in the /etc/default/login file to the value defined by the JASS_LOGIN_RETRIES variable. By reducing the logging threshold, additional information might be gained. The install-loginlog.fin script enables the logging of failed login attempts. For more information on this capability, refer to the login(1) manual page.

set-power-restrictions.fin

This script alters the configuration of `/etc/default/power` to restrict user access to power management functions using the `JASS_POWER_MGT_USER` and `JASS_CPR_MGT_USER` variables. As a result, access to the system's power management and suspend/resume functionality is controlled. This script applies only to systems running the Solaris OS versions 2.6 through 9. This script works only on software controllable power supplies, for example, power off at PROM prompt.

set-rmmount-nosuid.fin

Note – Solaris OS versions 8 and 9 are configured to mount removable media with the `nosuid` option by default. This script performs the necessary checks regardless of the default settings.

This script adds two entries to the `/etc/rmmount.conf` file to disable mounting of Set-UID files. It is important to disable mounting, because someone with access to a system could insert a diskette or CD-ROM and load Set-UID binaries, thereby compromising the system. For more information on this capability, refer to the `rmmount.conf`(4) manual page.

set-root-group.fin

This script changes the `root` user's primary group to `JASS_ROOT_GROUP` from group identifier #1 (`GID 1`, `other`). This script prevents the `root` user from sharing a common group with non-privileged users.

set-root-password.fin

Note – This script executes only during a JumpStart software installation. It does not execute when the Solaris Security Toolkit software is invoked from the command line.

This script automates setting the `root` password by setting the password to an initial value as defined by `JASS_ROOT_PASSWORD`. The password used in this script should only be used during the installation and must be changed immediately after the JumpStart installation process has successfully completed. By default, the password used by the `JASS_ROOT_PASSWORD` parameter is `t00lk1t`.

set-sys-suspend-restrictions.fin

This script alters the configuration of /etc/default/sys-suspend to restrict user access to suspend and resume functionality based on the JASS_SUSPEND_PERMS variable. This script applies only to systems running the Solaris OS versions 2.6 through 9. For more information, refer to the sys-suspend(1M) manual page.

set-system-umask.fin

This script ensures that all of the run-control scripts execute with a safe file creation mask based on the setting of JASS_UMASK. This setting is important because using a poorly chosen file creation mask could leave critical files writable by any user.

For versions prior to Solaris 8 OS, this script creates startup scripts at each run level, thereby setting the file creation mask to JASS_UMASK. For Solaris 8 and 9 OS versions, the CMASK variable in /etc/default/init is set to JASS_UMASK. For more information on this capability, refer to the init(1M) manual page.

set-term-type.fin

This script sets a default terminal type of vt100 to avoid issues with systems not recognizing dtterm. This script is mainly for use on systems that do not have graphical consoles and are generally accessed over a terminal console or other serial link. This script is provided as a convenience only and does not impact the security of the system.

set-tmpfs-limit.fin

This script installs a limit on the disk space that can be used as part of a tmpfs file system. This limit can help prevent memory exhaustion. The usable space is limited by default in this script to the value defined by JASS_TMPFS_LIMIT. The set-tmpfs-limit.fin script does not run under Solaris OS version 2.5.1, where this functionality is unsupported. For more information on this capability, refer to the mount_tmpfs(1M) manual page.

set-user-password-reqs.fin

The changes implemented by this script configure the password policy of a system for the next time that passwords are changed on a system. This profile might need to be further tuned to ensure that applications and operational functions are not adversely impacted by the hardening process.

This script enables more strict password requirements by enabling:

- Password aging
- Minimum intervals between password changes
- Minimum password length

This script accomplishes the requirements by using the values defined by the JASS_AGING_MINWEEKS, JASS_AGING_MAXWEEKS, JASS_AGING_WARNWEEKS, and JASS_PASSLENGTH variables to set the appropriate entries in the /etc/default/passwd file. This script is especially recommended for systems with nonprivileged user access.

This script modifies only the settings in the /etc/default/passwd file. It does not enable password aging for any user. The password aging requirements are implemented for each user upon the next password change. To enable password aging for a user without waiting for a password change event, use the passwd(1) command.

set-user-umask.fin

This script sets the default file creation mask (UMASK) to the value defined by JASS_UMASK for the following user startup files: /etc/.login, /etc/profile, /etc/skel/local.cshrc, /etc/skel/local.login, /etc/skel/local.profile, and /etc/default/login.

Update Finish Scripts

The following update finish scripts are described in this section:

update-at-deny.fin

This script adds the accounts listed in JASS_AT_DENY to the /etc/cron.d/at.deny file. This script prevents those users from using at and batch facilities. This script is used with the install-at-allow.fin file to determine access to at and batch facilities. For more information on this capability, refer to the at(1) manual page.

update-cron-allow.fin

This script adds the accounts listed in JASS_CRON_ALLOW to the /etc/cron.d/cron.allow file. This script allows those users to use the cron facility. This script is used with the update-cron-deny.fin script to determine access to the cron facility. For more information on this capability, refer to the crontab(1) manual page.

update-cron-deny.fin

This script adds the accounts listed in JASS_CRON_DENY to the /etc/cron.d/cron.deny file. This script prevents those users from accessing the cron facility. This script is used with the update-cron-allow.fin script to determine access to the cron facility. This script does not disable access for the root account user. For more information on this capability, refer to the crontab(1) manual page.

update-cron-log-size.fin

This script adjusts the maximum limit used for storing cron log information. For Solaris OS versions prior to Solaris 9 OS, this script adjusts the LIMIT variable in the /etc/cron.d/logchecker script. For Solaris 9 OS, this script adjusts the -s parameter in the /etc/logadm.conf file (for the /var/cron/log entry).

The size limit used by this script is determined by the JASS_CRON_LOG_SIZE environment variable. By default, the limit defined by the Solaris OS is only 0.5 megabytes.

update-inetd-conf.fin

This script disables all services, started from the inetd, that are defined by the JASS_SVCS_DISABLE variable. This script enables the services listed by the JASS_SVCS_ENABLE variable. If the same service is in both variables, the service is enabled. The JASS_SVCS_ENABLE variable takes precedence.

All services, including common services such as in.telnetd, in.ftpd, and in.rshd, in the base OS are disabled by default in Solaris OS versions 2.5.1 through 9. The services are disabled after the script inserts a # at the start of each line for service entries in the /etc/inet/inetd.conf file. Additional services installed by unbundled or third-party software are not disabled.

Using Product-Specific Finish Scripts

This section lists product-specific finish scripts, which are for hardening specific Sun products. These scripts are in the Finish directory. TABLE 4-1 lists product-specific finish scripts.

New finish scripts are released periodically to harden new and updated Sun products. For the latest list of scripts, refer to the Security Web site:

`http://www.sun.com/security/jass`

TABLE 4-1 Product-Specific Finish Scripts

Product	Driver Name
Sun Cluster 3.x Software	`suncluster3x-set-nsswitch-conf.fin`
Sun Fire High-End Systems Domains	`s15k-static-arp.fin` `s15k-install-klmmod-loader.fin`

Sun Fire High-End Systems System Controllers	`s15k-static-arp.fin`
	`s15k-exclude-domains.fin`
	`s15k-sms-secure-failover.fin`

suncluster3x-set-nsswitch-conf.fin

This script automates the configuration of a system as a Sun Cluster 3.x node. This script installs the cluster keyword into the `/etc/nsswitch.conf` file to simplify deploying Sun Cluster 3.x systems. The keyword should be located in the hosts field. This script applies only to Sun Cluster 3.x systems and does not execute on other systems.

For more information, refer to the Sun BluePrints OnLine article titled “Securing Sun Cluster 3.x Software.”

s15k-static-arp.fin

Note – This script applies only to Sun Fire High-End Systems SCs and domains and does not execute on other systems.

This script enables static ARP addresses on the I1 MAN network. The I1 MAN network is a network internal to the Sun Fire High-End Systems chassis, which is used for TCP/IP-based communication between the SCs and domains. By using static ARP instead of dynamic ARP, several ARP-based attacks against the SC no longer have any effect.

The following four files are used by the Sun Fire High-End Systems optional `s15k-static-arp.fin` script:

- `/etc/sms_sc_arp`
- `/etc/sms_domain_arp`
- `/etc/rc2.d/S73sms_arpconfig`
- `/etc/init.d/sms_arpconfig`

For more information, refer to the Sun BluePrints OnLine article titled "Securing the Sun Fire 12K and 15K System Controller" and the article titled "Securing the Sun Fire 12K and 15K Domains."

s15k-exclude-domains.fin

This script disables TCP/IP connectivity between the SC and one or more domains. For more information, refer to the Sun BluePrints OnLine article titled "Securing the Sun Fire 12K and 15K System Controller."

s15k-install-klmmod-loader.fin

This domain-only script adds a runtime control script, `klmmod`, which loads the `misc/klmmod` kernel module during startup of a Sun Fire High-End Systems domain.

s15k-sms-secure-failover.fin

This script automates enabling the use of Secure Shell by the failover daemon `fomd`. This script automates much of the Secure Shell configuration, in addition to disabling the use of legacy `r*` services. This script applies only to Sun Fire High-End Systems SCs and does not execute on other systems.

For more information, refer to the Sun BluePrints OnLine article titled "Securing the Sun Fire 12K and 15K System Controller."

CHAPTER 5

Audit Scripts

This chapter provides reference information on using, adding, modifying, and removing audit scripts. Audit scripts provide an easy method for periodically checking the security posture of a system. Check your systems regularly to make sure that their security matches your security profile.

The standard audit scripts confirm that modifications controlled by finish scripts were made to the system, and they report any discrepancies that occurred since the hardening run. Audit scripts use the same name as their correlating finish script, except they have a different suffix. Audit scripts use the `.aud` suffix instead of `.fin`.

This chapter contains the following topics:

Customizing Audit Scripts

This section provides instructions and recommendations for customizing existing audit scripts or creating new audit scripts. In addition, guidelines are provided for using audit script functions.

Customize Standard Audit Scripts

Just as with Solaris Security Toolkit drivers and finish scripts, you can customize audit scripts. Be careful when modifying scripts that are supplied with the Solaris Security Toolkit software. Always modify a copy of the script and not the original.

Failure to do so may result in a loss of functionality during Solaris Security Toolkit software upgrade or removal. Also, make as few changes as necessary to the code whenever possible and document those changes.

Use environment variables to customize audit scripts. The behavior of most scripts can be tailored by using environment variables, which eliminates the need to modify the script directly. If this is not possible, you may find it necessary to modify the code. For a list of all environment variables and guidelines for defining them, see Chapter 6.

Caution – Whenever you customize the standard finish scripts or develop new ones, be sure to make the corresponding changes to related audit scripts.

Note – Consider submitting a bug report or request for enhancement if you think that the change could benefit a wider audience. The Solaris Security Toolkit development team is always looking for ways to improve the software to better support its users.

▼ To Customize An Audit Script

Use the following steps to customize a standard audit script for your system and environment. Use these instructions so that newer versions of the original files do not overwrite your customized versions. Note that these files are not removed if you use the `pkgrm` command tor remove the software.

1. **Copy the audit script and related files that you want to customize.**

   Refer to Chapter 6 in the *Solaris Security Toolkit 4.1 Administration Guide* for information about audit scripts and their related files.

2. **Rename the copies with names that identify the files as custom scripts and files.**

   For naming guidelines, refer to “Guidelines”, Chapter 1, *Solaris Security Toolkit 4.1 Administration Guide*.

3. **Modify your custom script and files accordingly.**

   The `finish.init` file provides all audit script configuration variables. You can override the default variables by modifying the `user.init` file. This file is heavily commented to explain each variable, its impact, and its use in audit scripts. For more information about this file and modifying its variables, see Chapter 2. Or, if you want the change to be localized rather than to apply to all drivers, modify the driver.

   When you customize audit scripts, it is critical to the accuracy of the audit functionality that both finish and audit scripts are able to access your customization. This goal is most easily and effectively achieved by modifying environment variables in the `user.init` script instead of modifying other init files or modifying scripts directly.

   CODE EXAMPLE 5-1 shows how to customize the `install-openssh.aud` script to validate software installation. In this example, these checks ensure that the software package is installed, configured, and set up to run whenever the system reboots.

CODE EXAMPLE 5-1 Sample `install-openssh.aud` Script

```
#
#!/bin/sh
# Copyright (c) 2002 by Sun Microsystems, Inc.
# All rights reserved.
#
#ident  "@(#)install-openssh.aud       1.3     02/12/03       SMI"
#
# ****************************************************************
# Service definition section.
# ****************************************************************
#---------------------------------------------------------------
service="OpenSSH"
servfil="install-openssh.aud"
servhdr_txt="
#Rationale for Verification Check:
#This script will attempt to determine if the OpenSSH software is
#installed, configured and running on the system. Note that this
#script expects the OpenSSH software to be installed in package
#form in accordance with the install-openssh.fin Finish script.

#Determination of Compliance:

#It indicates a failure if the OpenSSH package is not installed,
#configured, or running on the system.
"

#---------------------------------------------------------------

servpkg="
   OBSDssh
```

CODE EXAMPLE 5-1 Sample `install-openssh.aud` Script *(Continued)*

```
#
"

#-----------------------------------------------------------------

servsrc="
   ${JASS_ROOT_DIR}/etc/rc3.d/S25openssh.server
"

#-----------------------------------------------------------------

servcfg="
   ${JASS_ROOT_DIR}/etc/sshd_config
"

#-----------------------------------------------------------------

servcmd="
   /opt/OBSDssh/sbin/sshd
"

#
****************************************************************
# Check processing section.
#
****************************************************************

start_audit "${servfil}" "${service}" "${servhdr_txt}"

logMessage "${JASS_MSG_SOFTWARE_INSTALLED}"

if check_packageExists "${servpkg}" 1 LOG ; then
 pkgName="`pkgparam -R ${JASS_ROOT_DIR} ${servpkg} NAME`"
 pkgVersion="`pkgparam -R ${JASS_ROOT_DIR} ${servpkg} VERSION`"
 pkgBaseDir="`pkgparam -R ${JASS_ROOT_DIR} ${servpkg} BASEDIR`"
 pkgContact="`pkgparam -R ${JASS_ROOT_DIR} ${servpkg} EMAIL`"

 logNotice "Package has description '${pkgName}'"
 logNotice "Package has version '${pkgVersion}'"
 logNotice "Package has base directory '${pkgBaseDir}'"
 logNotice "Package has contact '${pkgContact}'"

 logMessage "\n${JASS_MSG_SOFTWARE_CONFIGURED}"
 check_startScriptExists "${servsrc}" 1 LOG
 check_serviceConfigExists "${servcfg}" 1 LOG

 logMessage "\n${JASS_MSG_SOFTWARE_RUNNING}"
```

CODE EXAMPLE 5-1 Sample `install-openssh.aud` Script *(Continued)*

```
#
 check_processExists "${servcmd}" 1 LOG
fi

finish_audit
```

Create New Audit Scripts

You can create new audit scripts and integrate them into your deployment of the Solaris Security Toolkit software. Because scripts are commonly developed in Bourne shell, it is relatively easy to add new functionality. For those who are less experienced in UNIX shell scripting, examine existing audit scripts that perform similar functions to gain an understanding of how to accomplish a given task and to understand the correct sequence of actions.

The same conventions for developing new finish scripts apply to developing new audit scripts. For these conventions, see "Customizing Finish Scripts" on page 85.

Audit and finish scripts work together. Whenever you add new audit scripts, be sure to add their companion finish scripts.

Using Standard Audit Scripts

Audit scripts provide an automated way within the Solaris Security Toolkit software to validate a security posture by comparing it to a predefined security profile. Use audit scripts to validate that security modifications were made correctly, and to obtain reports on any discrepancies between a system's security posture and your security profile. For details on using audit scripts to validate system security, refer to Chapter 6 in the *Solaris Security Toolkit 4.1 Administration Guide*.

This section describes the standard audit scripts, which are in the Audit directory. Only the functionality performed by the audit scripts is described.

Each of the scripts in the Audit directory is organized into the following categories, which mirror those of the finish scripts in the Finish directory:

- Disable
- Enable
- Install
- Minimize
- Print
- Remove

- Set
- Update

In addition to these standard audit scripts, Solaris Security Toolkit software provides product-specific audit scripts. For a list of product-specific audit scripts, see "Using Product-Specific Audit Scripts" on page 154.

Disable Audit Scripts

The following disable audit scripts are described in this section:

- "disable-system-accounts.aud" on page 138
- "disable-uucp.aud" on page 138
- "disable-vold.aud" on page 138
- "disable-wbem.aud" on page 138
- "disable-xserver.listen.aud" on page 139

disable-ab2.aud

Note – This script is necessary only for systems running the Solaris OS versions 2.5.1 through 8, because the AnswerBook2 software is no longer used in Solaris OS version 9.

This script determines if the AnswerBook2 service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run, or running on the system.

disable-apache.aud

Note – Note that this script checks only for the Apache Web Server that was packaged by Sun and shipped as part of Solaris OS versions 8 and 9.

This script determines if the Apache Web Server is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-asppp.aud

Note – This script applies only to Solaris OS versions 2.5.1 through 8. For Solaris 9 OS, this service was replaced with the PPP service and is verified using the `disable-ppp.aud` script.

This script determines if the ASPPP service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-autoinst.aud

This script determines if automated installation functionality is installed or enabled on the system. It indicates a failure if the software is installed or configured to run (via a run-control script).

disable-automount.aud

Note – If this service is required, then do not use this script. Also, because this service relies on the RPC service, the `disable-rpc.aud` script also should also *not* be used.

This script determines if the automount service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-dhcpd.aud

Note – This script applies only to the DHCP server included in Solaris OS versions 8 and 9.

This script determines if the DHCP service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-directory.aud

Note – This audit script checks only for the Solaris 9 OS-bundled Sun Java System Directory Server. This script does not audit either the unbundled product or the Sun Java System Directory Server software provided with other Solaris OS versions.

This script determines if the Sun Java System Directory service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-dmi.aud

Note – This script is appropriate only for systems running Solaris OS versions 2.6 through 9.

This script determines if the DMI service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-dtlogin.aud

Note – This script is necessary only for systems running Solaris OS versions 2.6 through 9.

This script determines if the CDE login server, or `dtlogin`, is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-ipv6.aud

Note – This script is necessary only for systems running the Solaris OS versions 8 and 9.

This script checks for the absence of the IPv6 host name files, `/etc/hostname6.*`, that cause IPv6 interfaces to be plumbed. This script checks if the `in.ndpd` service is started. It indicates a failure if any IPv6 interfaces are configured, plumbed, or if the service is running.

disable-kdc.aud

Caution – If `JASS_DISABLE_MODE` is set to `conf`, the `kdc.conf` file is disabled, thus determining the ability of the system to act as both a Kerberos client and KDC server. Do not use this script in that manner if the system must act as a Kerberos client.

Note – This script is necessary only for systems running the Solaris 9 OS.

This script determines if the KDC service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-keyboard-abort.aud

Note – This script is used only in Solaris OS versions 2.6 through 9.

Note – Some systems feature key switches with a secure position. On these systems, setting the key switch to the secure position overrides any software default set with the `kdb` command.

This script determines if the system is configured to ignore keyboard abort sequences. Typically, when a keyboard abort sequence is initiated, the operating system is suspended and the console enters the OpenBoot PROM monitor or debugger. This script determines if the system can be suspended in this way.

disable-keyserv-uid-nobody.aud

This script determines if the `keyserv` service is not configured to prevent the use of default keys for the user `nobody`. This script indicates a failure if the `keyserv` process is not running with the `-d` flag and the `ENABLE_NOBODY_KEYS` parameter is not set to `NO` (for Solaris OS version 9).

disable-ldap-client.aud

Note – This script applies to Solaris OS versions 8 and 9.

This script determines if the LDAP client service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-lp.aud

This script determines if the line printer (`lp`) service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system. This script indicates a failure if the `lp` user is permitted to use the `cron` facility or has a `crontab` file installed.

disable-mipagent.aud

Note – This script is necessary only for Solaris OS versions 8 and 9.

This script determines if the Mobile IP service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-named.aud

Note – Disabling this service does not affect the ability of the system to act as a Domain Name System (DNS) client.

This script determines if the DNS is installed, configured, or running on the system. This script indicates a failure if the software is installed, configured to run (through a configuration file), or actually running on the system.

This script checks only for the DNS server that was packaged by Sun Microsystems and shipped as part of the Solaris OS.

disable-nfs-client.aud

Caution – If this service is required, then do not use this script. Also, because this service relies on the RPC service, the `disable-rpc.aud` script should *not* be used.

This script determines if the NFS client service is configured or running on the system. It indicates a failure if the software is configured to run or is running on the system.

disable-nfs-server.aud

Caution – If this service is required, then do not use this script. Also, because this service relies on the RPC service, the `disable-rpc.aud` script should *not* be used.

This script determines if the NFS service is configured or running on the system. It indicates a failure if the software is configured to run or is running on the system.

disable-nscd-caching.aud

This script determines if any of the `passwd`, `group`, `host`, or `ipnodes` services have a positive-time-to-live or negative-time-to-live value that is not set to `0`. It indicates a failure if the value is not `0`.

disable-picld.aud

Note – This script is necessary only for systems running Solaris OS versions 8 and 9.

This script determines if the PICL service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-power-mgmt.aud

Note – This script is necessary only for systems running Solaris OS versions 2.6 through 9.

This script determines if the power management service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-ppp.aud

Note – This service was introduced in Solaris 8 OS (7/01) and supplements the older ASPPP service. This script is necessary only for systems running Solaris OS versions 8 and 9.

This script determines if the PPP service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-preserve.aud

This script determines if the preserve functionality is enabled through its run-control script. If enabled, a failure is indicated.

disable-remote-root-login.aud

Note – Other mechanisms to access systems, such as the use of Solaris Secure Shell, that do not use `/bin/login` might still provide direct `root` access, even if the system passes this test.

This script determines if a `root` user is permitted to directly log in to or execute commands on a system remotely through programs using `/bin/login` such as Telnet. It indicates a failure if this is correct.

disable-rhosts.aud

Note – This script is necessary only for Solaris OS versions 2.6 through 9.

This script determines if the `rhosts` and `hosts.equiv` functionality is enabled through PAM configuration in `/etc/pam.conf`. It indicates a failure if this functionality is enabled using the `pam_rhosts_auth.so.1` module in the `/etc/pam.conf` file.

disable-rpc.aud

Caution – The RPC port mapper function should not be disabled if any of the following services are used on the system: automount, NFS, NIS, NIS+, CDE, and volume management (Solaris 9 OS only).

This script determines if the RPC service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system. In addition, this script indicates a failure for each service registered with the rpcbind port mapper.

disable-samba.aud

This script determines if the Samba service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system. Only Samba services included in the Solaris OS distribution are verified as being disabled. This script does not impact other Samba distributions installed on the system.

disable-sendmail.aud

Note – The Solaris Security Toolkit software modifications verify only that a Solaris OS system is not configured to receive email. Outgoing email is still processed normally.

By default, the sendmail service is configured to both forward local mail and to receive incoming mail from remote sources. If a system is not intended to be a mail server, then the sendmail service can be configured not to accept incoming messages. This script checks that the sendmail service is configured in such a manner.

This check is performed in a variety of ways depending on the version of the Solaris OS used. For Solaris OS version 9, this script checks for the existence of the following in the `/etc/mail/sendmail.cf` file:

```
Name=NoMTA4, Family=inet, Addr=127.0.0.1
```

For Solaris 8 OS, this script checks the `/etc/default/sendmail` file to determine if the `MODE` parameter is set to "" (nothing). For earlier versions of the Solaris OS, this script determines if the sendmail run-control scripts are disabled and an entry added to the `root` user's `crontab` file to automate the processing of queued mail.

This script indicates a failure if the sendmail service is not disabled in accordance with the checks unique to the Solaris OS version.

disable-slp.aud

Note – This script is necessary only for systems running Solaris OS versions 8 and 9.

This script determines if the SLP service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-sma.aud

This script determines if the SMA service is installed, configured, or running on the system. This script indicates a failure if the software is called, configured to run (through a run-control script), or actually running on the system.

disable-snmp.aud

Note – This script checks only the SNMP agent provided in Solaris OS versions 2.6 through 9.

This script determines if the SNMP service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system. This script verifies whether third-party SNMP agents are functioning on the system.

disable-spc.aud

Note – This script is necessary only for systems running Solaris OS versions 2.6 through 9.

This script determines if the SPC service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-ssh-root-login.aud

This script indicates a failure if the Solaris Secure Shell service distributed in the Solaris OS version 9 does not restrict access to the `root` account. This script is necessary only for systems running at minimum Solaris OS version 9 with the Solaris Secure Shell packages installed and enabled.

disable-syslogd-listen.aud

Note – Do not use this script on a `SYSLOG` server, because a `SYSLOG` server's function is to log remotely.

The script sets options to disallow the remote logging functionality of the `syslogd` process. This script determines if the `SYSLOG` service is configured to accept remote log connections. It indicates a failure if the `syslogd` process is not running with the `-t` flag and the `LOG_FROM_REMOTE` parameter is not set to `NO` (for the Solaris OS version 9).

disable-system-accounts.aud

For each account name listed in the JASS_ACCT_DISABLE environment variable, this script indicates a failure for each account that is not configured to use the shell defined by the JASS_SHELL_DISABLE variable. Also, this script indicates a failure if the shell program listed in the JASS_SHELL_DISABLE variable does not exist on the system.

Note that this script only checks accounts that are listed in the /etc/passwd file. It does not check for accounts listed in any other naming service (NIS, NIS+, or LDAP).

disable-uucp.aud

This script determines if the UUCP service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system. Also, it indicates a failure if the nuucp user exists, if in.uucpd exists in /etc/inetd.conf, or if a uucp crontab file is installed.

disable-vold.aud

Note – Do not use this script if automatic mounting and unmounting of removable media (such as diskettes and CD-ROMs) is needed.

This script determines if the VOLD service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or is running on the system.

disable-wbem.aud

Note – If this service is required, then do not use this script. Also, because this service relies on the RPC service, the disable-rpc.fin script should not be used either. Do not use this script if you use Solaris Management Console.

Note – This script is necessary only for systems running Solaris OS versions 8 and 9.

This script determines if the WBEM service is installed, configured, or running on the system. It indicates a failure if the software is installed, configured to run (via a run-control script), or running on the system.

disable-xserver.listen.aud

Note – This script is applicable only to the Solaris OS version 9.

It indicates a failure if the X11 server is configured to accept client connections using the TCP transport. In addition, it indicates a failure if the X11 server is running in a configuration that permits use of the TCP transport.

Enable Audit Scripts

The following enable audit scripts are described in this section:

enable-32bit-kernel.aud

This script determines if the system is configured to run in 32-bit mode or is running in 32-bit mode. It is valid only for Solaris OS versions 7, 8, and 9. Note that this script gives a failure if the system is running in 32-bit mode and is also running Solaris OS versions 7, 8, or 9. For Solaris OS versions 2.5.1 and 2.6, note that all systems run in 32-bit mode only.

enable-bsm.aud

This script determines if the SunSHIELD Solaris Basic Security Module (Solaris BSM) auditing functionality is enabled and running on the system, if the service is loaded in the `/etc/system` file, and if the `audit_warn` alias is defined in `/etc/mail/aliases`. If one or more of these checks fail, then the script indicates a failure.

enable-coreadm.aud

This script verifies that the system stores generated core files under the directory specified by `JASS_CORE_DIR`. It indicates a failure if the `coreadm` functionality present in the Solaris OS versions 7, 8, or 9 is not configured. Also, an error condition is generated if core files are not tagged with the specification denoted by `JASS_CORE_PATTERN`.

enable-ftp-syslog.aud

This script determines if the FTP service is not configured to log session and connection information. A failure is indicated if the FTP service logging is not enabled.

enable-ftpaccess.aud

Note – This script is necessary only on systems running the Solaris OS version 9.

This script determines if the FTP service is configured to use the `/etc/ftpd/ftpaccess` file. A failure is indicated if FTP is not configured properly.

enable-inetd-syslog.aud

This script determines if the Internet services daemon (`inetd`) service is configured to log session and connection information. Note that for Solaris OS version 9, this script checks that the `-t` option was added to the inetd command line and that the `ENABLE_CONNECTION_LOGGING` variable in the `/etc/default/inetd` file is set to `YES`. A failure is indicated if either of these checks fail.

enable-priv-nfs-ports.aud

This script determines if the NFS service is configured to accept only client communication that originates from a port in the privileged range below 1024. A failure is indicated if the NFS service is not configured properly.

enable-process-accounting.aud

This script determines if the processing accounting software is installed, enabled, or running on the system. A failure is indicated if this is not true.

enable-rfc1948.aud

Note – This script is necessary only on systems running Solaris OS versions 2.6 through 9.

This script determines if the system is configured to use RFC 1948 for its TCP sequence number generation. This script checks both the stored configuration and the actual runtime setting. A failure is displayed if the system is not configured to use RFC 1948–compliant TCP sequence number generation.

enable-stack-protection.aud

Note – This script is necessary only on systems running the Solaris OS versions 2.6 through 9.

This script determines if the `noexec_user_stack` and `noexec_user_stack_log` options are set in the `/etc/system` file to enable stack protections and exception logging. If these options are not enabled, a failure is reported.

enable-tcpwrappers.aud

Note – This script applies only to Solaris OS version 9 using the bundled TCP wrapper packages.

This script determines if TCP wrappers are not installed or configured using the `hosts.allow|deny` templates included with the Solaris Security Toolkit software or enabled by using the `ENABLE_TCPWRAPPERS` variable. A failure is reported if the system is not using TCP wrappers.

Install Audit Scripts

The following install audit scripts are described in this section:

install-at-allow.aud

This script determines if a user name is listed in the `JASS_AT_ALLOW` variable and does not exist in the `/etc/cron.d/at.allow` file. The list of user names defined by `JASS_AT_ALLOW` is, by default, empty. To pass this check, each user name must exist in both the `/etc/passwd` file and the `/etc/cron.d/at.allow` file. Furthermore, a user name should not be in the `/etc/cron.d/at.deny` file. A failure is displayed if a user name is not listed in both files.

install-fix-modes.aud

This script determines if the Fix Modes program was installed and run on the system. It indicates a failure if the software is not installed or has not been run. Further, this script uses Fix Modes in debug mode to determine if any additional file system objects should be adjusted.

install-ftpusers.aud

This script determines if a user name listed in the JASS_FTPUSERS parameter does not exist in the ftpusers file.

install-jass.aud

This script determines if the Solaris Security Toolkit (SUNWjass) package is installed on the system. A failure is reported if this package is not installed.

install-loginlog.aud

This script checks for the existence and proper ownership and permissions for the /var/adm/loginlog file. It indicates a failure if the file does not exist, has invalid permissions, or is not owned by the root account.

install-md5.aud

This script determines if the MD5 software is installed on the system. A failure is reported if the software is not installed.

install-nddconfig.aud

This script determines if the nddconfig run-control script files identified in the Sun BluePrints OnLine article, *Solaris Operating Environment Network Settings for Security* and included with the Solaris Security Toolkit, have been copied to and their settings made active on the target system.

The script performs the following checks per object:

1. Test to ensure that the source and target file types (regular file, symbolic link, or directory) match
2. Test to ensure that the source and target file type contents are the same

This script also verifies that the settings defined by the nddconfig script are actually in place on the running system. This script uses its own copy of the nddconfig script in the Solaris Security Toolkit to provide more accurate reporting of results, especially in cases where the script name has changed or where other scripts are used to implement the same effects.

This script gives a failure when any of the checks described above are found to be false.

install-newaliases.aud

This script checks for the existence of the `/usr/bin/newaliases` program. It indicates a failure if this file does not exist or is not a symbolic link.

install-openssh.aud

Note – Solaris 9 OS includes a version of the Secure Shell software; therefore, this script is not used if you install Solaris 9 OS.

This script determines if the OpenSSH package specified by the script is installed and configured. A failure is reported if the package is not installed.

install-recommended-patches.aud

This script determines if the patches listed in the Recommended and Security Patch Cluster file are installed on the system. The patch information is collected from `JASS_HOME_DIR/Patches` directory, based on Solaris OS version of the system being tested. A failure is displayed if one or more of these patches are not installed.

Note that this script indicates success if the version of the patch installed is equal to or greater than the version listed in the patch order file.

install-sadmind-options.aud

This script determines if the `sadmind` service exists in the `/etc/inet/inetd.conf file`. If it does, this script checks to ensure that options are set to those defined by the `JASS_SADMIND_OPTIONS` variable. The default setting is `-S 2`.

install-security-mode.aud

This script checks the status of the EEPROM security mode. It displays a warning if the mode is not command or full. In addition, this script checks the PROM failed login counter and displays a warning if it is not zero.

Note that because the `install-security-mode.fin` script cannot change the security mode of the system, this script only indicates a warning for noncompliance rather than reporting a failure.

install-shells.aud

This script determines if any shell defined by the JASS_SHELLS parameter is not listed in the shells file. TABLE 5-1 lists the shells defined by JASS_SHELLS.

TABLE 5-1 List of shells Defined by JASS_SHELLS

/usr/bin/sh	/usr/bin/csh
/usr/bin/ksh	/usr/bin/jsh
/bin/sh	/bin/csh
/bin/ksh	/bin/jsh
/sbin/sh	/sbin/jsh
/bin/bash	/bin/pfcsh
/bin/pfksh	/bin/pfsh
/bin/tcsh	/bin/zsh
/usr/bin/bash	/usr/bin/pfcsh
/usr/bin/pfksh	/usr/bin/pfsh
/usr/bin/tcsh	/usr/bin/zsh

A failure is displayed if any shells listed in JASS_SHELLS are not also listed in the shells file.

install-strong-permissions.aud

This script determines if any of the modifications recommended by the install-strong-permissions.fin script were not implemented. A failure is displayed if any of these modifications were not made.

install-sulog.aud

This script checks for the proper ownership and permissions of the /var/adm/sulog file. It indicates a failure if the file does not exist, has invalid permissions, or is not owned by the root account.

install-Sun_ONE-WS.aud

This script determines if the Sun Java System, formerly Sun ONE, Web Server is installed on the system in accordance with the install-Sun_ONE-WS.fin script. A failure is reported if the software is not installed correctly.

install-templates.aud

This script determines if the files defined by the JASS_FILES variable was successfully copied to the target system. It indicates a failure if either of the two following checks fail: a test to ensure that the source and target file types match (regular file, symbolic link, or directory) and a test to ensure that their contents are the same.

Minimize Audit Script

This script determines if any nonessential packages exist on the system as defined by the minimization finish script, minimize-Sun_ONE-WS.fin. A failure is displayed if any of the nonessential packages are present on the system.

Print Audit Scripts

The following print audit scripts are described in this section:

- "print-jass-environment.aud" on page 146
- "print-jumpstart-environment.aud" on page 146
- "print-rhosts.aud" on page 147
- "print-sgid-files.aud" on page 147
- "print-suid-files.aud" on page 147
- "print-unowned-objects.aud" on page 147
- "print-world-writable-objects.aud" on page 147

These scripts perform the same functions as the print finish scripts, except that they are customized for audit use.

print-jass-environment.aud

This script displays the variables used by the Solaris Security Toolkit. It does not perform any validation or other checks on the content. The variables and their content are displayed.

print-jumpstart-environment.aud

This script is for JumpStart mode only. It is used to print out JumpStart environment variable settings. This script does not perform any audit checks.

print-rhosts.aud

This script displays a notice for any files found with the name of `.rhosts` or `hosts.equiv`. Further, this script displays the contents of those files for further inspection.

print-sgid-files.aud

This script displays a notice for any files that have the set-gid bit set, and it provides a full (long) listing for further review.

print-suid-files.aud

This script displays a notice for any files that have the set-uid bit set, and it provides a full (long) listing for further review.

print-unowned-objects.aud

This script displays a notice for any files that are not assigned to a valid user and group, and it provides a full (long) listing for further review.

print-world-writable-objects.aud

This script displays a notice for any matching files that are world-writable, and it provides a full (long) listing for further review.

Remove Audit Script

The `remove-unneeded-accounts.aud` script validates that unused Solaris OS accounts, defined by the `JASS_ACCT_REMOVE` variable, were removed from the system.

Set Audit Scripts

The following set audit scripts are described in this section:

- "set-banner-dtlogin.aud" on page 148
- "set-banner-ftpd.aud" on page 148
- "set-banner-sendmail.aud" on page 148

set-banner-dtlogin.aud

Note – This script is necessary only for systems running Solaris OS versions 2.6 through 9.

This script verifies that a service banner for the CDE or `dtlogin` service is defined. This script verifies that the system displays the contents of `/etc/motd` by listing it in the file template `JASS_ROOT_DIR/etc/dt/config/Xsession.d/0050.warning`.

set-banner-ftpd.aud

Note – This script is necessary only for systems running the Solaris OS versions 2.6 through 9.

This script checks that the FTP service banner matches the value defined by the `JASS_BANNER_FTPD` variable. It indicates a failure if the service banner does not match. The value of the variable is `Authorized Use Only`.

set-banner-sendmail.aud

This script verifies that the sendmail service is configured to display the service banner as defined by the `JASS_BANNER_SENDMAIL` environment variable. This banner is displayed to all clients connecting to the sendmail service over the network.

set-banner-sshd.aud

Note – This script is used only for systems running Solaris 9 OS.

This script verifies that the Secure Shell service banner is displayed by ensuring that the Secure Shell service displays the contents of `/etc/issue` to the user prior to authenticating access to the system.

set-banner-telnet.aud

Note – This script is necessary only for systems running Solaris OS versions 2.6 through 9.

This script checks that the Telnet service banner matches the value defined by the `JASS_BANNER_TELNETD` variable It indicates a failure if the service banner does not match. The value of the variable is `Authorized Use Only`.

set-ftpd-umask.aud

Note – This script is necessary only for systems running Solaris OS versions 2.6 through 9.

This script checks that the FTP service banner matches the value defined by the `JASS_FTPD_UMASK` variable. It indicates a failure if the file creation mask value does not match. The value of variable is `022`.

set-login-retries.aud

This script determines if the login `RETRIES` parameter is assigned the value defined by the `JASS_LOGIN_RETRIES` variable. The variable default is set to `3`. A failure is displayed if the variable is not set to the default.

set-power-restrictions.aud

Note – This script is necessary only for systems running Solaris OS versions 2.6 through 9.

This script checks the /etc/default/power file and indicates a failure if the PMCHANGEPERM and CPRCHANGEPERM parameters do not have a hyphen "-" as their values.

set-rmmount-nosuid.aud

Note – Solaris OS versions 8 and 9 are configured to mount removable media with the nosuid option by default. This script performs the necessary checks regardless of the default settings.

This script determines if the /etc/rmmount.conf file restricts the mounting of removable UFS or HSFS file systems by enforcing the nosuid parameter. A failure is displayed if this restriction is not defined in the /etc/rmmount.conf file.

set-root-group.aud

This script determines if the root account's primary group is set to the value defined by the JASS_ROOT_GROUP variable. A failure is displayed if it is not defined properly.

set-root-password.aud

This script checks the password of the root account. It indicates a failure if the value is the same as that of the JASS_ROOT_PASSWORD variable. This check is done to encourage users to change the root password from the value defined by JASS_ROOT_PASSWORD as soon as possible.

set-sys-suspend-restrictions.aud

Note – This script is necessary only for systems running Solaris OS versions 2.6 through 9.

This script checks the /etc/default/sys-suspend file. It indicates a failure if the PERMS parameter does not have a hyphen "-" as its value.

set-system-umask.aud

This script determines if the system's default file creation mask is set to the value defined by the JASS_UMASK variable. The default value is set to 022. A failure is displayed if the variable is not properly defined.

set-term-type.aud

This script determines if the /etc/profile and the /etc/login files set the default terminal type to vt100. A failure is displayed if the default terminal type is not defined properly. This script is provided as a convenience only, and a failure does not impact the security of a system.

set-tmpfs-limit.aud

Note – The set-tmpfs-limit.aud script does not run under Solaris OS version 2.5.1, where this functionality is unsupported.

This script determines if any tmpfs file systems are defined in the /etc/vfstab file without their size being limited to the JASS_TMPFS_SIZE variable, which is set to a default of 512 megabytes. A failure is reported if the tmpfs file system size does not comply with the JASS_TMPFS_SIZE value.

set-user-password-reqs.aud

This script reviews the password policy settings on the system as defined previously. It indicates an error if the values do not match the following default values defined by the Solaris Security Toolkit:

- MINWEEKS - "1"
- MAXWEEKS - "8"
- WARNWEEKS - "1"
- PASSLENGTH - "8"

The default values are contained in the following environment variables:

- JASS_AGING_MINWEEKS
- JASS_AGING_MAXWEEKS
- JASS_AGING_WARNWEEKS
- JASS_PASS_LENGTH

set-user-umask.aud

This script determines if any of the following files do not set the `umask` parameter to the value defined by the `JASS_UMASK` variable. The default value is set `022`.

- `/etc/.login`
- `/etc/profile`
- `/etc/skel/local.cshrc`
- `/etc/skel/local.login`
- `/etc/skel/local.profile`
- `/etc/default/login`

A failure is displayed if these files do not set the `umask` parameter appropriately.

Update Audit Scripts

The following update audit scripts are described in this section:

- "update-at-deny.aud" on page 152
- "update-cron-allow.aud" on page 153
- "update-cron-deny.aud" on page 153
- "update-cron-log-size.aud" on page 153
- "update-inetd-conf.aud" on page 153

update-at-deny.aud

This script determines if a user account is listed in the `JASS_AT_DENY` variable and is not listed in the `/etc/cron.d/at.deny` file. The list of user accounts defined by the `JASS_AT_DENY` variable is as follows:

- `root`
- `daemon`
- `bin`
- `sys`
- `adm`
- `lp`
- `uucp`
- `smmsp`
- `nobody`
- `noaccess`

To pass this check, each user account must exist in both the `/etc/passwd` file and the `/etc/cron.d/at.deny` file. The user account must not exist in the `/etc/cron.d/at.allow` file, because it would override the setting (due to precedence). A failure is displayed if any of these checks fail.

update-cron-allow.aud

This script determines if a user account is listed in the JASS_CRON_ALLOW variable and not in /etc/cron.d/cron.allow file. By default, the value is only the root user. A failure is displayed if this check fails.

update-cron-deny.aud

This script determines if a user account is listed in the JASS_CRON_DENY variable and not in the /etc/cron.d/cron.deny file. The list of user accounts defined by the JASS_CRON_DENY variable is as follows:

- daemon
- bin
- sys
- adm
- lp
- uucp
- smmsp
- nobody
- noaccess

To pass this check, each user account must exist in both the /etc/passwd file and the /etc/cron.d/cron.deny file. Furthermore, the user account must not exist in the /etc/cron.d/cron.allow file, because it would override this setting (due to precedence). A failure is displayed if any of these checks fail.

update-cron-log-size.aud

Note – This script is not needed for systems running Solaris OS version 2.5.1 because the functionality is not supported.

This script determines if the cron facility is configured to increase its default size limit for log files. The check method is based on the version of the Solaris OS and the value of the JASS_CRON_LOG_SIZE variable. The size limit defined by the JASS_CRON_LOG_SIZE variable is 20480 kilobytes. A failure is displayed if the size limitation is not correct. This script runs on Solaris OS versions 2.6, 7, 8, or 9.

update-inetd-conf.aud

This script determines if any of the services listed in the JASS_SVCS_DISABLE variable are disabled in /etc/inetd.conf. This script also checks to ensure that services listed in the JASS_SVCS_ENABLE variable are enabled in the

/etc/inetd.conf file. If a service is listed in both variables, then the service is left enabled by the JASS_SVCS_ENABLE variable. A failure is displayed if any of these checks fail.

The JASS_SVCS_DISABLE parameter is populated as shown in TABLE 5-2.

TABLE 5-2 Sample Output of JASS_SVCS_DISABLE

100068	100083	100087	100134	100146	100147
100150	100155	100166	100221	100229	100230
100232	100234	100235	100242	100424	300326
536870916	chargen	comsat	daytime	discard	dtspc
echo	eklogin	exec	finger	fs	ftp
kerbd	klogin	kshell	login	name	netstat
printer	rexd	rquotad	rstatd	rusersd	rwalld
shell	smtp	sprayd	sun-dr	systat	talk
telnet	tftp	time	ufsd	uucp	uuidgen
walld	xaudio				

The JASS_SVCS_ENABLE variable is, by default, empty. Some drivers may use it, such as the suncluster3x-secure.driver.

Using Product-Specific Audit Scripts

TABLE 5-3 lists product-specific audit scripts for specific Sun products. These scripts are in the Audit directory.

New audit scripts are released periodically for new and updated Sun products. For the latest list of scripts, refer to the Security Web site:

http://www.sun.com/security/jass

TABLE 5-3 Product-Specific Audit Scripts

Product	Driver Name
Sun Cluster 3.x Software	`suncluster3x-set-nsswitch-conf.aud`
Sun Fire High-End Systems Domains	`s15k-static-arp.aud` `s15k-install-klmmod-loader.aud`

Sun Fire High-End Systems System Controllers	`s15k-static-arp.aud`
	`s15k-exclude-domains.aud`
	`s15k-sms-secure-failover.aud`

suncluster3x-set-nsswitch-conf.aud

This script determines if the `/etc/nsswitch.conf` file lists the `cluster` keyword as the first source for the host's database. This script applies only to Sun Cluster 3.x systems and should not be executed on other systems. A failure is displayed if this is not true.

For more information, refer to the Sun BluePrints OnLine article titled "Securing Sun Cluster 3.x Software."

s15k-static-arp.aud

For SMS versions 1.2 and newer, this script verifies that the static ARP configuration files are installed on Sun Fire High-End Systems system controllers (SCs) and domains. For system controllers, the file is `/etc/sms_sc_arp`. For domains, the file is `/etc/sms_domain_arp`.

This script checks that all existing domains have Ethernet addresses as listed in the SC static ARP startup script and corresponding data file.

For more information, refer to the Sun BluePrints OnLine article titled "Securing the Sun Fire 12K and 15K System Controller" and "Securing the Sun Fire 12K and 15K Domains."

s15k-exclude-domains.aud

For SMS versions 1.2 and newer, this script determines if the `/etc/opt/SUNWSMS/SMS/config/MAN.cf` file exists. If it does, this script checks to ensure that all the domains listed are excluded from the I1 MAN. The script excludes all domains from the I1 MAN. If the site has altered the script to exclude only a subset of the domains, this script issues a warning about each domain that is still part of the I1 MAN.

For more information, refer to the Sun BluePrints OnLine article titled "Securing the Sun Fire 12K and 15K System Controller."

s15k-install-klmmod-loader.aud

This domain-only script checks whether the `misc/klmmod` kernel module is loaded on a Sun Fire High-End Systems domain.

s15k-sms-secure-failover.aud

For SMS versions 1.2 through 1.4.1, this script determines if the Sun Fire High-End Systems system controller is configured based on the recommendations in the Sun BluePrints OnLine article titled "Securing the Sun Fire 12K and 15K System Controller." It indicates a failure if any of the services listed in the `SMS_SVCS_DISABLE` variable are enabled in `/etc/inet/inetd.conf`.

CHAPTER 6

Environment Variables

This chapter provides reference information about using environment variables. This chapter describes all of the variables used by the Solaris Security Toolkit software and provides tips and techniques for customizing their values.

This chapter contains the following topics:

Customizing and Assigning Variables

The Solaris Security Toolkit software contains environment variables that provide a simple and easy way to customize and direct the behavior of its drivers and scripts. Because they are simply Bourne shell variables, all of the rules that apply to shell variables apply to Solaris Security Toolkit variables. This section provides information and recommendations for customizing and assigning variables.

Within the Solaris Security Toolkit software, there are four categories of environment variables:

- Framework function variables
- Finish and audit script variables
- JumpStart mode variables
- User variables

Note – All of the categories listed can be assigned or customized.

Before customizing variables, it is important that you understand the role of each variable type and its purpose within the Solaris Security Toolkit software. Setting and customizing variables are key to configuring the Solaris Security Toolkit software to suit your system, environment, and security policies. For detailed information about using variables, see "Using Environment Variables" on page 162.

In some cases, you might find that customizing the standard variables, drivers, and scripts does not address your specific needs. In these cases, you might want to develop variables, drivers, and scripts for your environment. For more information about developing variables, see "Creating Environment Variables" on page 161.

This section contains the following topics:

Assign Static Variables

Static variables are those that are assigned a fixed or static value. This value is set before the Solaris Security Toolkit run is initiated and, unless its value is changed by the external factors, remains constant throughout the run. Also, the value of these variables does not change depending on the context or environment in which the software is run.

Static variables are helpful when a policy setting is not dependent on external factors such as the system's type, network settings, or applications installed. For example, password aging is usually defined by a corporate or divisional policy. Assigning a static variable would apply a setting to all systems and devices within the corporation or division. Because password aging is not dependent on external factors, system administrators usually set it as a static variable.

The following is an example of assigning a static variable.

```
JASS_AGING_MAXWEEKS="8"
JASS_AGING_MINWEEKS="1"
```

In this case, user passwords are configured to expire eight weeks after their most recent change. Furthermore, the second variable, also defined as a static variable, restricts user password changes to one per week maximum.

Assign Dynamic Variables

Dynamic variables are those that generally require greater flexibility and whose values are based on the output of commands or the contents of files. In this way, the variable is more aware of the environment in which it is run and is able to adapt to the environment more effectively. The following is an example of assigning a dynamic variable.

```
JASS_AT_DENY="`awk -F: '{ print $1 }' ${JASS_PASSWD}`"
```

In this case, each of the users defined in the JASS_PASSWD (for example, JASS_ROOT_DIR/etc/passwd) file are added to the variable JASS_AT_DENY. The list of users varies depending on the system on which the Solaris Security Toolkit software is run. In this way, the software is more responsive to its environment. Similar constructions can be made to include all users except for some predefined exceptions. The following example illustrates such a case where every user on the system is added to the JASS_CRON_DENY variable with the exception of the root and ORACLE® accounts.

```
JASS_CRON_DENY="`awk -F: '{ print $1 }' ${JASS_PASSWD} |\
   egrep -v '^root|^oracle'`"
```

Assign Complex Substitution Variables

Taking the assigning methods a step further is the notion of complex substitution. Using this technique, more sophisticated values can be assigned to a variable based perhaps on policy, file content, or other mechanisms.

An example of how this is achieved combines assigning both static and dynamic variables. In this example, the JASS_FTPUSERS is assigned a value based both on a static list and the output of the JASS_ROOT_DIR/etc/passwd file.

```
JASS_FTPUSERS="`awk -F: '$1 !~ /^ftp/ { print $1 }' \
${JASS_PASSWD}` guest"
```

In this example, the guest account is always added to the JASS_FTPUSERS variable. In addition, each user listed in JASS_PASSWD whose login name does not begin with the prefix "ftp" is also added to the JASS_FTPUSERS variable. Using combinations of these techniques, almost any configuration can be achieved capable of meeting the needs of most organizations.

Another sophisticated technique is to define a substitution policy based on a shell script or function. For such an example, refer to the declaration of the `JASS_SHELLS` variable in the `Drivers/finish.init` file (CODE EXAMPLE 6-1). In this case, the variable assignment is dependent on the version of the OS.

CODE EXAMPLE 6-1 Variable Assignment Based on OS Version

```
#
if [ -z "${JASS_SHELLS}" ]; then
# These shells are by default found in Solaris 2.5.1 to Solaris 7
JASS_SHELLS="
      /usr/bin/sh     /usr/bin/csh     /usr/bin/ksh
      /usr/bin/jsh    /bin/sh          /bin/csh
      /bin/ksh        /bin/jsh         /sbin/sh
      /sbin/jsh"
# This is to handle special cases by OS.
case ${JASS_OS_REVISION} in
      5.8 | 5.9)
         JASS_SHELLS="${JASS_SHELLS}
            /bin/bash       /bin/pfcsh       /bin/pfksh
            /bin/pfsh       /bin/tcsh        /bin/zsh
            /usr/bin/bash   /usr/bin/pfcsh   /usr/bin/pfksh
            /usr/bin/pfsh   /usr/bin/tcsh    /usr/bin/zsh"
      ;;
esac
fi
export JASS_SHELLS
# This function could be further enhanced, for example, to remove
# those shell entries that do not exist on the system. This
# could be done by adding the following code:
tmpShells="${JASS_SHELLS}"
JASS_SHELLS=""
for shell in ${tmpShells}; do
    if [ -x "${JASS_ROOT_DIR}${shell}" ]; then
        if [ -z "${JASS_SHELLS}" ]; then
            JASS_SHELLS="${JASS_SHELLS}/${shell}"
        fi
    fi
done
```

This type of functionality can be useful on minimized systems where some of the shells are not available, such as `/usr/bin/bash` or `/usr/bin/tcsh`, which exist in the `SUNWbash` and `SUNWtcsh` packages respectively. This technique helps to reduce the number of notice and warning messages generated by the software due to improper assignment of variables.

Assign Global and Profile-Based Variables

Global variables can be assigned to override the default values of many of the Solaris Security Toolkit variables. Customize the `user.init` file to define and assign variables for which default values are to be overridden during each Solaris Security Toolkit software run. This file is read by the `driver.init` program whenever a software run is initiated.

Also, you can assign profile-based variables to override default values. This override occurs within the profile itself, after the call to the `driver.init` file. Assigning variables within a profile allows variables to be updated, extended, and overridden for specific profiles rather than for all of them. For example, the file `desktop-secure.driver` contains the following profile-based variable override:

```
JASS_SVCS_ENABLE="telnet ftp dtspc rstatd 100155"
```

In this case, the `JASS_SVCS_ENABLE` variable is assigned to include entries for Telnet, FTP, `dtspc`, `rstatd`, and `rpc.smserverd` (`100155`) services. This assignment instructs the software to leave these services enabled (or to enable them if they were disabled). Normally, the default behavior of the software is to disable those services, per the `JASS_SVCS_DISABLE` variable.

Creating Environment Variables

Although, typically, the standard Solaris Security Toolkit variables provide what you need and can be customized for your system and environment, occasionally, you might need to develop new variables. Often this requirement occurs when you develop your own scripts. You can create new variables and assign them to support your site-specific or custom scripts. Creating new variables enables you to take advantage of the software's framework and modularity.

To quickly and easily build new functionality or implement additional customization, leverage the existing capabilities of the software. Use the standard variables as samples from which to develop new variables. Whenever possible, customize the standard variables rather than develop new ones. By using the software's framework in this way, you can develop and support less-customized code.

Note – The prefix `JASS_` is reserved for use by the Solaris Security Toolkit software developers. This prefix must not be used when creating new variables. Use a prefix unique to your company or organization.

To simplify portability and configuration issues, the environment variables defined in the various `.init` scripts are used throughout the Solaris Security Toolkit software.

If you require additional variables, add them as environment variables to the `user.init` and `user.run` scripts.

To add a new variable, add the variable declaration with its default value and export it in the `user.init` file. This process provides a global, default value that you can subsequently change as needed by overriding it within a security profile (driver). For example, the following code adds a new variable `ABC_TESTING` with a default value of `0` to the `user.init` file.

```
ABC_TESTING="0"
export ABC_TESTING
```

There are times when the value of the variable should only be set if it is currently undefined. This approach is most useful when permitting an administrator to change values from the login shell. To accomplish this task, you would alter the previous code sample as follows.

```
if [ -z "${ABC_TESTING}" ]; then
   ABC_TESTING="0"
fi
export ABC_TESTING
```

Using Environment Variables

This section provides descriptions of all the standard variables defined by the Solaris Security Toolkit software, listed in alphabetical order. Where applicable, recommendations and other helpful information are provided so that you can use these variables more effectively.

Within the software, the four categories of environment variables are as follows:

- Framework variables
- Finish and audit script variables
- JumpStart mode variables
- User variables

Each of the variables described in this section is defined in one of the following files, depending on its function within the Solaris Security Toolkit software. (As noted previously, the functions are divided into categories based on their purpose.)

- `driver.init` (framework and JumpStart mode variables)
- `finish.init` (finish and audit script variables)
- `user.init` (user variables and global override variables)

For detailed information about these files, see Chapter 2.

To simplify portability and configuration issues, the environment variables defined in the various `.init` scripts are used throughout the Solaris Security Toolkit software.

If you require additional variables, add them as environment variables to the `user.init` and `user.run` scripts. For more information, see "Creating Environment Variables" on page 161.

Note – The default environment variable values used by scripts are defined in the `finish.init` script.

This section presents the variables in the following organization:

Define Framework Variables

Framework variables are those that are defined and used by the Solaris Security Toolkit software to either maintain configuration state or to provide core variables that are used by the software. These variables are typically global and are in the software framework, its core functions, and scripts.

You can dramatically change the behavior of the software by changing these variables; therefore, change them only when absolutely necessary. Also, changes should be made only by experienced administrators who clearly understand the impact of the changes and can resolve any resulting problems.

Note – Not all framework variables can be modified. This limitation was done to promote consistency between Solaris Security Toolkit software deployments and to aid in supporting those configurations.

Caution – Never attempt to directly change any framework variables that cannot otherwise be overridden.

This section describes the following framework variables:

JASS_AUDIT_DIR

Note – Normally, this variable should *not* require modification.

The convention used by the Solaris Security Toolkit software is to store all of the audit scripts in the `Audit` directory. However, for flexibility, the `JASS_AUDIT_DIR` environment variable is available for administrators who need to store audit scripts in different locations. By default, this variable is set to `JASS_HOME_DIR/Audit`.

JASS_CHECK_MINIMIZED

This variable is used in audit runs only. The value of this variable determines how the Solaris Security Toolkit software performs the `check_minimized` function that is included in many of the audit scripts. If this variable is set to `0`, which is the default value, then the `check_minimized` function responds immediately without performing any of its checks. If this variable has no value, the `check_minimized` function operates as normal. This variable is included to permit the exclusion of these checks from a software run when a system has not been minimized. Otherwise, the `check_minimized` functions would result in failure messages on non-minimized systems, thereby precluding an audit run passing successfully.

JASS_CONFIG_DIR

Starting with version 0.3 of the Solaris Security Toolkit software, the variable `JASS_CONFIG_DIR` was renamed to `JASS_HOME_DIR` to provide a clearer meaning as to its use. The `JASS_CONFIG_DIR` variable is deprecated and should no longer be used. See "`JASS_HOME_DIR`" on page 171.

JASS_DISABLE_MODE

This variable defines the approach used by the Solaris Security Toolkit software to disable services that are started from `run-control` scripts. For Solaris OS version 9, this variable is assigned the default value of `conf`, whereas all earlier releases default to the value of `script`.

Note – If a particular service does not use a configuration file, or it does not check for its existence prior to starting, then the software uses the `script` method when disabling the service.

When the JASS_DISABLE_MODE variable is set to conf, the software disables a service by moving aside its configuration file. This approach is effective on services that first check for the existence of a configuration file prior to starting. This approach leads to a more supportable and sustainable configuration because Solaris OS patches rarely replace these disabled configuration files.

When this variable is set to script, the software disables services by moving aside their respective run-control scripts. This approach is also effective because a service is not able to run, if it is never permitted to start. This configuration is less supportable, however, because Solaris OS patches install run-control scripts, re-enabling services that were disabled.

You should not change the default settings. Note that if security scanners are used, they should be adequately tested using this configuration. Setting this variable to conf could result in false positives, because most scanners typically (and erroneously) check only for the existence of run-control scripts. Note that the audit function does not have this limitation.

JASS_DISPLAY_HOSTNAME

Note – The JASS_DISPLAY_HOSTNAME variable is used only when JASS_VERBOSITY is less than or equal to 2.

This variable controls the display of host name information during audit runs. You can select the level of verbosity to be used by the Solaris Security Toolkit software. In single-line output modes (see "JASS_VERBOSITY" on page 181), you have the option of tagging each line with the host name of the system on which the software is being run. This value is the same as JASS_HOSTNAME. Including this information can be useful when processing runs from multiple systems. If this variable is set to 1, then the software prepends the host name of the target system to each line of output. Otherwise, the software does not include this information. By default, the software does not display this information.

JASS_DISPLAY_SCRIPTNAME

Note – The JASS_DISPLAY_SCRIPTNAME variable is used only when JASS_VERBOSITY is less than or equal to 2.

This variable controls the display of the current script name during audit runs. You can select the level of verbosity to be used by the Solaris Security Toolkit software. In single-line output modes (see "JASS_VERBOSITY" on page 181), you have the option of tagging each line with the name of the current audit script being run.

Including this information can be useful when attempting to determine the source of failure messages. If this variable is set to 1, then the software prepends the current audit script name to each line of output. Otherwise, the software does not include this information. By default, the software includes this information.

JASS_DISPLAY_TIMESTAMP

Note – The JASS_DISPLAY_TIMESTAMP variable is used only when JASS_VERBOSITY is less than or equal to 2.

This variable controls the display of timestamp information during audit runs. You can select the level of verbosity to be used by the Solaris Security Toolkit software. In single-line output modes (see "JASS_VERBOSITY" on page 181), you have the option of tagging each line with the timestamp associated with the software run. This value is the same as JASS_TIMESTAMP. Including this information can be useful when processing multiple runs from a single system or set of systems. If this variable is set to 1, then the software prepends the timestamp of the run to each line of output. Otherwise, the software does not include this information. By default, the software does not display this information.

JASS_FILES

This variable specifies a list of file system objects that are copied to the target system. Specify each of the objects listed in this variable by using its absolute path name. Each object is stored in a file system hierarchy under the root directory of JASS_HOME_DIR/Files.

Specifying Files With the JASS_FILES *Variable*

Note – Note that this functionality is basically equivalent to the JASS_FILES "+" function.

File lists are added to the contents of the general file list only when the Solaris Security Toolkit software is run on a defined version of the Solaris OS. A version-specific list is created by appending the major and minor operating system version to the end of the JASS_FILES variable, separated by underscores. The Solaris Security Toolkit software currently supports the options listed in TABLE 6-1.

TABLE 6-1 Supporting OS Versions in the JASS_FILES Variable

Variable	OS Version
JASS_FILES	Applies to any version of Solaris OS
JASS_FILES_5_5_1	Applies only to Solaris OS version 2.5.1
JASS_FILES_5_6	Applies only to Solaris OS version 2.6
JASS_FILES_5_7	Applies only to Solaris OS version 7
JASS_FILES_5_8	Applies only to Solaris OS version 8
JASS_FILES_5_9	Applies only to Solaris OS version 9

For example, the /etc/logadm.conf file is only applicable to the Solaris 9 OS. To install the Files/etc/logadm.conf file only on the Solaris 9 OS, use the following syntax.

```
JASS_FILES_5_9="
               /etc/logadm.conf
"
```

You can use the JASS_FILES variable to specify files in the following ways:

- Specify the file that is copied from the Solaris Security Toolkit software to the client.

  The following example is from the hardening.driver:

```
JASS_FILES="
               /etc/motd
"
```

  By defining the JASS_FILES environment variable to include this file, the /etc/motd file on the client is replaced by the JASS_HOME_DIR/Files/etc/motd file from the Solaris Security Toolkit software distribution. You can copy any file, directory, or symbolic link this way by simply including it in the Files directory and adding it to the JASS_FILES definition in the appropriate driver.

- Specify host-specific files.

Host-specific files are those that are only copied if the host name of the target system matches the host name assigned to the object in the `Files` directory. To use this capability, simply create files in the `Files` directory of the following form:

```
/etc/syslog.conf.$HOSTNAME
```

In this scenario, the `JASS_HOME_DIR/Files/etc/syslog.conf.`*HOSTNAME* file is copied to `JASS_ROOT_DIR/etc/syslog.conf` on the target system only if its host name matches the value defined by *HOSTNAME*. When there is both a `syslog.conf` and `syslog.conf.`*HOSTNAME*, the host-specific file takes precedence.

- Specify OS release-specific files.

  OS release-specific files are similar in concept to host-specific files, but are copied to the target system only if the target's version of the Solaris OS matches that assigned to the object in the Files directory. To use this functionality, create files in the Files directory with the following form:

```
/etc/syslog.conf+$OS
```

  In this example, the `JASS_HOME_DIR/Files/etc/syslog.conf+OS` file is copied to the target as `JASS_ROOT_DIR/etc/syslog.conf` only if the version of the Solaris OS on the target system matches the value defined by `OS`.

  The `OS` variable should mirror the output produced by the `uname -r` command. For example, if Solaris OS version 8 were being secured, then a file with the name of `JASS_HOME_DIR/Files/etc/syslog.conf+5.8` would be copied. This file would not be copied to any other OS release. The OS-specific files take precedence over generic files, but host-specific files take precedence over OS-specific files.

Also, the `JASS_FILES` variable supports OS-specific extensions. Use these extensions to specify a list of file system objects that should be copied only for certain versions of the Solaris OS. The OS-specific `JASS_FILES` extensions are supported for Solaris OS versions 5.5.1, 5.6, 7, 8, and 9. For example, to copy a list of files only for Solaris 8 OS, define the `JASS_FILES_5_8` variable and assign to it the list of files to be copied.

Customizing the JASS_FILES *Variable*

This section describes and illustrates how to customize the JASS_FILES environment variable. The following code examples are taken from the Drivers/config.driver file. This profile file performs basic configuration tasks on a platform. This example profile provides clear examples of how file templates, drivers, and finish scripts are used.

In the following example, this driver is configured to copy the /.cshrc and /.profile files from the JASS_HOME_DIR/Files directory onto the target platform when the driver.run function is called.

```
JASS_FILES="
/.cshrc
/.profile
"
```

To change the contents of either of these files, modify the copies of the files located in the JASS_HOME_DIR/Files directory. If you only need to add or remove file templates, simply adjust the JASS_FILES variable accordingly. Track changes to the Solaris Security Toolkit configuration using a change-control mechanism. For more information, refer to "Maintaining Version Control", Chapter 1, *Solaris Security Toolkit 4.1 Administration Guide*.

The software supports OS version-specific file lists. For detailed information, see the previous section "Specifying Files With the JASS_FILES Variable" on page 167.

JASS_FILES_DIR

This variable points to the location of the Files directory under JASS_HOME_DIR. This directory contains all of the file system objects that can be copied to the client.

To copy objects to a system, you must list a file in a JASS_FILES variable or one of its OS-specific extensions. These objects are copied to the client during hardening runs by the install-templates.fin script. Set the JASS_FILES variable within an individual driver. This variable is not defined by any other configuration file. For other methods of copying files using this variable, see "JASS_FILES" on page 167. By default, this variable is set to JASS_HOME_DIR/Files.

This variable does not normally require modification.

JASS_FINISH_DIR

Note – Normally, this variable should *not* require modification.

The convention used by the Solaris Security Toolkit software is to store all finish scripts in the `Finish` directory. However, for flexibility, the `JASS_FINISH_DIR` environment variable is for storing finish scripts in different locations. By default, this variable is set to `JASS_HOME_DIR/Finish`.

JASS_HOME_DIR

This variable defines the location of the Solaris Security Toolkit source tree. In JumpStart mode, the JumpStart variable `SI_CONFIG_DIR` sets the `JASS_HOME_DIR` variable. In standalone mode, it is set by the `jass-execute` script, which is included in the base directory.

This variable should not normally require modification, except when the Solaris Security Toolkit software is installed into a subdirectory of a pre-existing JumpStart installation. For these cases, append the path of the Solaris Security Toolkit source to `SI_CONFIG_DIR`, as in `SI_CONFIG_DIR/jass-`*n.n*, where *n.n* is the current version number of the software.

JASS_HOSTNAME

Caution – Do *not* change this variable, because several components of the framework rely on this variable being set properly.

This variable contains the host name of the system on which the Solaris Security Toolkit software is being run. This variable is set during software runs through the Solaris OS `uname -n` command within the `driver.init` script.

JASS_ISA_CAPABILITY

Note – Normally, this variable should *not* require modification.

This variable defines the Solaris OS instruction set potential of the target system. Use this variable to determine if the system has the potential of operating in 32- or 64-bit mode. This task is done to provide instruction set architecture (ISA) information for use by finish scripts. The value of this variable is defined based on a check for the

existence of the Solaris OS package, SUNWkvmx. If this package is installed, then the system is assumed to be 64-bit capable, and this variable is set to "64". Otherwise, the system is assumed to be only 32-bit capable, and this variable is set to "32".

JASS_LOG_BANNER

Note – The logBanner function only displays output when JASS_VERBOSITY variable is 3 or higher and the JASS_LOG_BANNER variable is *not* 0.

This variable controls the behavior of the logBanner function. The logBanner function generates all of the banner messages used by the Solaris Security Toolkit software. If this variable is set to 0, then the logBanner function responds immediately without displaying any information. Otherwise, the logBanner function displays the information passed to it as an argument. Use this variable to adjust the output of the software to better suit your needs. By default, this variable has no value and, therefore, the logBanner function operates normally.

JASS_LOG_ERROR

This variable controls the behavior of the logError function. The logError function generates messages with the prefix [ERR]. If this variable is set to 0, then the logError function responds immediately without displaying any information. Otherwise, the logError function displays the information passed to it as an argument. Use this variable to adjust the output of the software to better suit your needs. By default, this variable has no value and, therefore, the logError function operates normally.

JASS_LOG_FAILURE

This variable controls the behavior of the logFailure function. The logFailure function generates messages with the prefix [FAIL]. If this variable is set to 0, then the logFailure function responds immediately without displaying any information. Otherwise, the logFailure function displays the information passed to it as an argument. Use this variable to adjust the output of the software to better your needs. By default, this variable has no value and, therefore, the logFailure function operates normally.

JASS_LOG_NOTICE

This variable controls the behavior of the `logNotice` function. The `logNotice` function generates messages with the prefix `[NOTE]`. If this variable is set to `0`, then the `logNotice` function responds immediately without displaying any information. Otherwise, the `logNotice` function displays the information passed to it as an argument. Use this variable to adjust the output of the software to better suit your needs. By default, this variable has no value and, therefore, the `logNotice` function operates normally.

JASS_LOG_SUCCESS

This variable controls the behavior of the `logSuccess` function. The `logSuccess` function generates messages with the prefix `[PASS]`. If this variable is set to `0`, then the `logSuccess` function responds immediately without displaying any information. Otherwise, the `logSuccess` function displays the information passed to it as an argument. Use this variable to adjust the output to suit your needs. By default, this variable has no value and, therefore, the `logSuccess` function operates normally.

JASS_LOG_WARNING

This variable controls the behavior of the `logWarning` function. The `logWarning` function generates messages with the prefix `[WARN]`. If this variable is set to `0`, then the `logWarning` function responds immediately without displaying any information. Otherwise, the `logWarning` function displays the information passed to it as an argument. Use this variable to adjust the output to suit your needs. By default, this variable has no value and, therefore, the `logWarning` function operates normally.

JASS_MODE

Caution – Do *not* change this variable.

This variable defines the way that the Solaris Security Toolkit software operates. This variable accepts one of six values: `APPLY`, `UNDO`, `AUDIT`, `HISTORY_LAST`, and `HISTORY_FULL`. Normally, this variable is set in standalone mode by the `jass-execute` command. In JumpStart mode, it defaults to `APPLY`. For the purpose of this variable, `APPLY` refers to hardening runs.

JASS_OS_REVISION

Caution – Do *not* change this variable, because it is set automatically.

This variable is a global variable specifying the OS version of the client on which the Solaris Security Toolkit software is being used. This variable is set automatically in the `driver.init` script through the `uname -r` command and exported so that all other scripts can access it.

JASS_OS_TYPE

Note – Only Trusted Solaris 8 OS is supported by the Solaris Security Toolkit.

This variable determines if the system being hardened or audited is a Solaris OS system or a Trusted Solaris™ OS system. If the system is running a generic version of Solaris OS, it is set to "`Generic`," otherwise it is set to "`TS8`". This variable is in the `driver.init` file.

JASS_PACKAGE_DIR

Note – Normally, this variable should *not* require modification.

The convention used by the Solaris Security Toolkit software is to store all software packages to be installed in the `Packages` directory. However, for flexibility, the `JASS_PACKAGE_DIR` variable is available to store packages in a different location. By default, in standalone mode, this variable is set to `JASS_HOME_DIR/Packages`.

In JumpStart mode, however, this variable is defined as a transient mount-point, `JASS_ROOT_DIR/tmp/jass-packages`. The package directory, stored on the JumpStart server, is mounted as this directory on this client during a JumpStart installation.

JASS_PATCH_DIR

Note – Normally, this variable should *not* require modification.

The convention used by the Solaris Security Toolkit software is to store all of the software patches to be installed in the `Patches` directory. However, for flexibility, the `JASS_PATCH_DIR` variable is available to store patches in a different location. By default, in standalone mode, this variable is set to `JASS_HOME_DIR/Patches`.

In JumpStart mode, however, this variable is defined as a transient mount-point, `JASS_ROOT_DIR/tmp/jass-patches`. The actual package directory, stored on the JumpStart server, is mounted as this directory on this client during a JumpStart installation.

JASS_PKG

Caution – Do *not* change this variable.

This variable defines the Solaris OS package name of the Solaris Security Toolkit software. This variable has a value of `SUNWjass`.

JASS_REPOSITORY

Caution – Do *not* change this variable.

This variable is part of the execution log and undo functions. The path specified by `JASS_REPOSITORY` defines the directory where the required run information is stored. This functionality facilitates the capture of information related to each script that is run, the resulting output of each, and the listing of files that were installed, modified, or removed during a run.

This variable is dynamically altered during the execution of the software. Any values assigned to this variable in any of the `init` files are overwritten. By default, this variable is assigned the value of `JASS_ROOT_DIR/var/opt/JASS_PKG/run/JASS_TIMESTAMP`.

JASS_ROOT_DIR

Caution – Do *not* change this variable.

This variable defines the root directory of the target's file system. For JumpStart mode, this directory is always `/a`. For standalone mode, this variable should be set to `/` or the root directory of the system.

Starting with Solaris Security Toolkit software version 0.2, the software automatically sets this variable's value in the `jass-execute` script, so manual modification is no longer required.

JASS_RUN_AUDIT_LOG

Caution – Do *not* change this variable.

This variable is part of the execution log. This variable defines the name and absolute path to the file that stores the output generated during an audit run. This information is collected to document which scripts were executed, in addition to the output of each audit check tested during the course of the run.

Any errors or warnings generated are stored in this file. The information stored in this file is equivalent to the output displayed on the screen during an audit run. By default, this variable is set to `JASS_REPOSITORY/jass-audit-log.txt`.

JASS_RUN_CHECKSUM

Caution – Do *not* change this variable.

This variable is part of the execution log and undo functionality. This variable is also used by the `jass-check-sum` program included in `JASS_HOME_DIR`. This variable defines the name and absolute path to the file that stores all of the checksum information used by the software. This information records the state of files both before and after modification. This information is used to determine if files changed since they were last modified by the software. This information is stored within the `JASS_REPOSITORY` directory structure and has a default value of `JASS_REPOSITORY/jass-checksums.txt`.

JASS_RUN_FINISH_LIST

This variable's name was changed before the Solaris Security Toolkit 4.0 software release. See "JASS_RUN_SCRIPT_LIST" on page 177.

JASS_RUN_INSTALL_LOG

Caution – Do *not* change this variable.

This variable is part of the execution log. This variable defines the name and absolute path to the file that stores the output generated during hardening runs. This information is collected to document which scripts are executed, in addition to listing any files that were installed, removed, or modified during a run.

Any errors or warnings generated are stored in this file. The information stored in this file is equivalent to the output displayed on the screen during a hardening run. By default, this variable is set to `JASS_REPOSITORY/jass-install-log.txt`.

JASS_RUN_MANIFEST

Caution – Do *not* change this variable.

This variable is part of the execution log and undo functionality. This variable defines the name and absolute path to the file that stores the manifest information associated with a run. The manifest file records the operations conducted as part of a hardening run. This file is also used in undo runs to determine which files must be moved, and in what order, to restore a system to a previous configuration. By default, this variable is set to `JASS_REPOSITORY/jass-manifest.txt`.

JASS_RUN_SCRIPT_LIST

Caution – Do *not* change this variable.

This variable is part of the execution log. This variable defines the name and absolute path to the file that stores a listing of all finish or audit scripts executed during a run. This information is collected for informational and debugging purposes and is stored within the `JASS_REPOSITORY` directory structure. By default, this variable is set to `JASS_REPOSITORY/jass-script-list.txt`.

JASS_RUN_UNDO_LOG

Caution – Do *not* change this variable.

This variable is part of the execution log. This variable defines the name and absolute path to the file that stores the output generated during an undo run. This information is collected to document which scripts were executed, in addition to listing any files that were installed, removed, or modified during a run.

Any errors or warnings generated are stored in this file. The information stored in this file is equivalent to the output displayed on the screen during an undo run. By default, this variable is set to `JASS_REPOSITORY/jass-undo-log.txt`.

JASS_RUN_VERSION

Caution – Do *not* change this variable.

This variable is part of the execution log. This variable defines the name and absolute path to the file containing the version and associated information for a run. This file typically includes information about the version, mode, and security profile used by the Solaris Security Toolkit software during its run. This information is collected to document the manner in which the software was used on a system. By default, this variable is set to `JASS_REPOSITORY/jass-version.txt`.

JASS_SAVE_BACKUP

Caution – The Solaris Security Toolkit undo feature is not available if you define the value of `JASS_SAVE_BACKUP` as `0`.

This variable controls the creation of backup files during hardening runs. The default value is 1, which causes the software to create a backup copy of any file modified on the client. If the value is changed to `0`, then all backup copies created during a run are removed at its completion.

Modify the `user.init` script if you want to prevent the creation of backup copies of files. The value in the `user.init` script overrides any value set in the variable.

JASS_SCRIPTS

This variable specifies a list of finish scripts to execute on a target system when you want to use a specific driver. For each entry, make sure you provide a corresponding finish script with the same name located in the `JASS_FINISH_DIR` directory.

Also, store an audit script in `JASS_AUDIT_DIR`, corresponding to each finish script that is stored in `JASS_FINISH_DIR`.

Specifying Files With the JASS_SCRIPTS *Variable*

The JASS_SCRIPTS variable supports OS-specific extensions. Use these extensions to specify a list of finish scripts to execute only when the target system is running certain versions of the Solaris OS. Create a version-specific list by appending the major and minor operating system versions to the end of the JASS_SCRIPTS variable, separated by underscores. The Solaris Security Toolkit software supports the options listed in TABLE 6-2.

TABLE 6-2 Supporting OS Versions in the JASS_SCRIPT Variable

Variable	OS Version
JASS_SCRIPTS	Applies to any version of the Solaris OS
JASS_SCRIPTS_5_5_1	Applies only to the Solaris OS version 2.5.1
JASS_SCRIPTS_5_6	Applies only to the Solaris OS version 2.6
JASS_SCRIPTS_5_7	Applies only to the Solaris OS version 7
JASS_SCRIPTS_5_8	Applies only to the Solaris OS version 8
JASS_SCRIPTS_5_9	Applies only to the Solaris OS version 9

For example, to use the disable-something.fin script only on the Solaris 9 OS, you would add the following to the driver.

```
JASS_SCRIPTS_5_9="
disable-something.fin
"
```

In this example, assuming that the operating system is the Solaris 9 OS, the disable-something.fin script is added to the end of JASS_SCRIPTS.

Note – The OS-specific file and script lists are always appended to the generic list of files and scripts. As a result, they are always executed after their more general counterparts. For example, if JASS_SCRIPTS is "a b" and JASS_SCRIPTS_5_9 is "c d", after the append operation, JASS_SCRIPT is "a b c d" and JASS_SCRIPTS_5_9 is automatically discarded.

Customizing the JASS_SCRIPTS *Variable*

To add or remove finish scripts from a driver, modify the JASS_SCRIPTS variable as needed. Drivers provide a mechanism for grouping file templates and scripts into a single security profile. These profiles allow you to logically group customization. For

example, a single profile could be used to define a baseline that is applied to all of the systems within an organization. Alternatively, a profile could define the modifications that are done to secure systems operating as database servers. These profiles can be used individually or combined into more complex profiles.

```
JASS_SCRIPTS="
print-jass-environment.fin
install-recommended-patches.fin
install-jass.fin
set-root-password.fin
set-term-type.fin
"
```

In this example, five different scripts are configured to run when the `driver.run` function is executed. (See "Understanding Driver Functions and Processes" on page 67 for more information about `driver.run`.) These five scripts are grouped together because they represent system configuration changes that are not directly related to hardening, which is why they are grouped into the `config.driver`.

JASS_STANDALONE

Note – Normally, this variable should *not* require modification.

This variable controls whether the Solaris Security Toolkit software runs in standalone or JumpStart mode. This variable defaults to `0` for JumpStart installations and 1 when the `jass-execute` command is used to initiate a run.

JASS_SUFFIX

Caution – Do *not* change this variable.

This variable determines which suffixes must be appended onto backup copies of files. By default, this is set to `JASS.JASS_TIMESTAMP`. During a run, the value of the timestamp field changes to reflect the time a file is created. This action guarantees that all backup file names are unique.

This variable is dynamically altered during runs. Any value assigned to this variable in the `init` files is overwritten.

JASS_TIMESTAMP

Note – Normally, this variable should *not* require modification.

This variable creates the `JASS_REPOSITORY` directory, `/var/opt/SUNWjass/run/JASS_TIMESTAMP`. As noted previously, this directory contains the logs and manifest information for each run of the Solaris Security Toolkit software. This variable contains the timestamp associated with the start of a run, and its value is maintained for the entire run. As a result, its value is unique for each run. This unique value allows information for each run to be clearly separated from all others, based on the time that the run was started. By default, this variable is set to `date '+%EY%m%d%OH%OM%S'`. This command creates a timestamp of the form YYYYMMDDHHMMSS. For example, a run started at 1:30 p.m. on April 1, 2003 would be represented by the value `20030401013000`.

JASS_UNAME

This variable was renamed to `JASS_OS_REVISION` before the Solaris Security Toolkit 4.0 release. See "JASS_OS_REVISION" on page 174.

JASS_USER_DIR

This variable specifies the location of the configuration files `user.init` and `user.run`. By default, these files are stored in the `JASS_HOME_DIR/Drivers` directory. Use these files to customize the Solaris Security Toolkit software to meet the needs of your organization.

If you need to customize the Solaris Security Toolkit software, do so in these files to minimize the impact of Solaris Security Toolkit software upgrades in the future.

Global variables should be created and assigned either in the `user.init` file or within a driver. New functions or overrides of existing functions should be implemented in the `user.run` file. All variable or function overrides take precedence over their counterparts defined in the Solaris Security Toolkit software.

JASS_VERBOSITY

Caution – Do *not* modify this variable directly. Instead, use the `jass-execute` command with the `-V` option

This variable controls how the Solaris Security Toolkit software displays its results when running during audit runs. The software currently supports five different verbosity levels: 0 through 4. Set this variable to any of these values using the `-V` option with the `jass-execute` command.

Note – In hardening runs and other operations, this variable is set to `3` and normally should *not* be changed.

The verbosity levels used during audit runs are as listed in TABLE 6-3.

TABLE 6-3 Verbosity Levels for Audit Runs

Level	Description
`0`	Final. This mode results in only one line of output that indicates the combined result of the entire verification run. This mode is useful if a single `PASS` or `FAIL` is needed.
`1`	Consolidated. In this mode, one line of output per audit script is generated indicating the result of each audit script. In addition, subtotals are generated at the end of each script, as well as a grand total at the end of the run.
`2`	Brief. This mode combines the attributes of the Consolidated verbosity level and includes the results of the individual checks within each audit script. This mode is useful for quickly determining those items that passed and failed within a single audit script. The format of this mode still represents one result per line.
`3`	Full. This is the first of the multiline verbosity modes. In this mode, banners and headers are printed to illustrate more clearly the checks that are being run, their intended purpose, and how their results are determined. This is the default verbosity level and more suitable for those new to the Solaris Security Toolkit verification capability.
`4`	Debug. This mode extends upon the Full verbosity mode by including all entries that are generated by the logDebug logging function. Currently, this is not used by any of the Solaris Security Toolkit audit scripts, but it is included for completeness and to allow administrators to embed debugging statements within their code.

In the least verbose mode, level `0`, only a single line is displayed representing the overall result for a run. The output at this level would look like:

```
# ./jass-execute -a starfire_ssp-secure.driver -V 0
starfire_ssp-secure.driver [PASS] Grand Total : 0 Error(s)
```

JASS_VERSION

Caution – Do *not* change this variable.

This variable defines the version of the Solaris Security Toolkit software associated with the software distribution being used. This variable documents the version of the software and permits its use with logging and other functions.

Define Script Behavior Variables

Script behavior variables are those that are defined and used by the Solaris Security Toolkit software to affect the behavior of finish and audit scripts. The Solaris Security Toolkit software provides a robust and flexible framework for customizing its functionality to suit individual site requirements. One of our design goals was to limit the amount of source code that had to be modified for users to implement site-specific customization. The script variables provide an easy to use method for altering the behavior of a script without modifying the script's source code.

These variables are defined in the `JASS_HOME_DIR/Drivers/finish.init` file. Although they are global, their use is typically limited to a small set of finish and audit scripts. As described earlier in this chapter, you can customize these variables using techniques such as static, dynamic, and complex assignment in either the `user.init` file or within an individual driver.

Tune these variables where necessary to meet organizational or site security policy and requirements. Used in this manner, the software provides the greatest value in helping you improve and sustain the security posture of your environment.

This section describes the following script behavior variables:

JASS_ACCT_DISABLE

This variable contains a list of user accounts that should be disabled on a system. During hardening runs, these accounts are disabled by the `disable-system-accounts.fin` script. During audit runs, the `disable-system-accounts.aud` script inspects the accounts defined by this variable, to ensure that they are disabled.

By default, the following accounts are assigned to the `JASS_ACCT_DISABLE` variable:

- `daemon`
- `bin`
- `adm`
- `lp`

- uucp
- nuucp
- nobody
- smtp
- listen
- noaccess
- nobody4
- smmsp

JASS_ACCT_REMOVE

This variable contains a list of user accounts that should be removed from a system. During hardening runs, these accounts are removed by the `remove-unneeded-accounts.fin` script. During audit runs, the `remove-unneeded-accounts.aud` script inspects the system to ensure that the accounts do not exist.

By default, the following accounts are assigned to the `JASS_ACCT_REMOVE` variable:

- smtp
- listen
- nobody4

JASS_AGING_MAXWEEKS

This variable contains a numeric value specifying the maximum number of weeks passwords remain valid before they must be changed by users. The default value for this variable is `8` (weeks). This variable is used by the `set-user-password-reqs.fin` script and the `set-user-password-reqs.aud` script.

JASS_AGING_MINWEEKS

This variable contains a numeric value specifying the minimum number of weeks that must pass before users can change their passwords. This variable is used by the `set-user-password-reqs.fin` script and the `set-user-password-reqs.aud` script. This variable has a default value of `1` (week).

JASS_AGING_WARNWEEKS

This variable contains a numeric value specifying the number of weeks before passwords expire and users are warned. This warning is displayed to users upon login during the warning period. This variable is used by the `set-user-password-reqs.fin` script and the `set-user-password-reqs.aud` script. The default value of this variable is `1` (week).

JASS_AT_ALLOW

This variable contains a list of user accounts that should be permitted to use the `at` and `batch` facilities. During hardening runs, the `install-at-allow.fin` script adds each user account defined by this variable to the `JASS_ROOT_DIR/etc/cron.d/at.allow` file, if not already present. Similarly, during audit runs, the `install-at-allow.aud` script determines if each user account defined by this variable is listed in the `at.allow` file. Note that for a user account to be added or checked, it must also exist in `JASS_PASSWD`. By default, this variable contains no user accounts.

JASS_AT_DENY

This variable contains a list of user accounts that should be prevented from using the `at` and `batch` facilities. During hardening runs, the `update-at-deny.fin` script adds each user account defined by this variable to the `JASS_ROOT_DIR/etc/cron.d/at.deny` file, if not already present. Similarly, during audit runs, the `update-at-deny.aud` script determines if each user account defined by this variable is listed in the `at.deny` file. Note that for a user account to be added or checked, it must also exist in `JASS_PASSWD`. By default, this variable contains all of the user accounts defined on the system in the `JASS_PASSWD` file.

JASS_BANNER_DTLOGIN

This variable contains a string value that represents a file name containing a banner message to be displayed to users after logging into CDE. During hardening runs, this banner is installed by the `set-banner-dtlogin.fin` script. During audit runs, the existence of this banner is checked by the `set-banner-dtlogin.aud` script. The default value of this variable is `/etc/motd`.

JASS_BANNER_FTPD

This variable contains a string value that is used as a banner displayed to users prior to authenticating for FTP service. During hardening runs, this banner is installed by the `set-banner-ftpd.fin` script. During audit runs, the existence of this banner is checked by the `set-banner-ftpd.aud` script. The default value of this variable is `\"Authorized Use Only\"`.

Note – The back slash characters are required in this string to prevent the quote characters from being interpreted by the command shell. When installed in the relevant FTP configuration file, the string displays as "`Authorized Use Only`."

JASS_BANNER_SENDMAIL

This variable contains a string value that is used as a banner displayed to clients immediately after connecting to the sendmail service. During hardening runs, this banner is installed by the `set-banner-sendmail.fin` script. During audit runs, the existence of this banner is checked by the `set-banner-sendmail.aud` script. The default value of this variable is `Mail Server Ready`.

JASS_BANNER_SSHD

This variable contains a string value that represents a file name containing a banner message to be displayed to users prior to authenticating the Secure Shell service. During hardening runs, this banner is installed by the `set-banner-sshd.fin` script. During audit runs, the existence of this banner is checked by the `set-banner-sshd.aud script`. The default value of this variable is `/etc/issue`.

JASS_BANNER_TELNETD

Note – The back slash characters are required in this string to prevent the quote characters from being interpreted by the command shell. When installed in the relevant Telnet configuration file, the string displays as "`Authorized Use Only`."

This variable contains a string value that is used as a banner displayed to users prior to authenticating for Telnet service. During hardening runs, this banner is installed by the `set-banner-telnetd.fin` script. During audit runs, the existence of this banner is checked by the `set-banner-telnetd.aud` script. The default value of this variable is `\"Authorized Use Only\"`.

JASS_CORE_PATTERN

This variable contains a string value that represents the path name and core file naming pattern used by the `coreadm` facility. This variable is used to configure `coreadm` to restrict core files generated on the system to the specified directory and name based on the file pattern defined by this variable. During hardening runs, `coreadm` is configured by the `enable-coreadm.fin script`. During audit runs, the `coreadm` configuration is checked by the `enable-coreadm.aud` script. The default value of this variable is `/var/core/core.%f.%p.%n.%u.%g.%t`. For more information on the file naming options, refer to the `coreadm`(1M) manual page.

JASS_CPR_MGT_USER

This variable contains a string value that defines which users are permitted to perform checkpoint and resume functions on a system. During hardening runs, this restriction is implemented by the `set-power-restrictions.fin` script. During audit runs, this restriction is checked by the `set-power-restrictions.aud` script. The default value of this variable is "`-`", indicating that only the `root` account is permitted to perform these management functions. For more information, see the `/etc/default/power` information in Chapter 2.

JASS_CRON_ALLOW

This variable contains a list of user accounts that should be permitted to use the `cron` facility. During hardening runs, the `update-cron-allow.fin` script adds each user defined by this variable to the `JASS_ROOT_DIR/etc/cron.d/cron.allow` file, if not already present. Similarly, during audit runs, the `update-cron-allow.aud` script determines if each user defined by this variable is listed in the `cron.allow` file. Note that for a user account to be added or checked, it must also exist in `JASS_PASSWD`. By default, this variable contains only the root account.

JASS_CRON_DENY

This variable contains a list of user accounts that should be prevented from using the `cron` facility. During hardening runs, the `update-cron-deny.fin` script adds each user defined by this variable to the `JASS_ROOT_DIR/etc/cron.d/cron.deny` file, if not already present. Similarly, during audit runs, the `update-cron-deny.aud` script determines if each user defined by this variable is listed in the `cron.deny` file. Note that for a user account to be added or checked, it must also exist in `JASS_PASSWD`. By default, this variable contains all of the user accounts defined in

the JASS_PASSWD file with user identifiers less than 100 and greater than 60000. Typically, these ranges are reserved for administrative access. Note that by default, the root account is explicitly excluded from this list.

JASS_CRON_LOG_SIZE

This variable contains a numeric value representing the maximum size, in blocks, that the cron facility log file can be before it is rotated. During hardening runs, this setting is installed by the update-cron-log-size.fin script. During audit runs, this setting is checked by the update-cron-log-size.aud script. The default value of this variable is 20480 (or 10 megabytes). This size is an increase over the default Solaris OS value of 1024 (or 0.5 megabytes).

JASS_FIXMODES_DIR

This variable contains a string value representing the absolute path to the FixModes software distribution, if present. If the FixModes software is installed from the tar distribution by the Solaris Security Toolkit, it is installed into the directory defined by this variable. During hardening runs, this variable is used by the install-fix-modes.fin script to install and run the FixModes software. During audit runs, the FixModes software is run by the install-fix-modes.aud script. The default value of this variable is /opt.

JASS_FIXMODES_OPTIONS

This variable contains a list of options that are passed to the FixModes software when it is run during hardening runs from the install-fix-modes.fin script. This variable is not used during audit runs. By default, no options are specified by this variable.

JASS_FTPD_UMASK

This variable contains a numeric (octal) value that represents the file creation mask (UMASK) to be used by the FTP service. During hardening runs, this setting is installed by the set-ftpd-umask.fin script. During audit runs, this setting is checked by the set-ftpd-umask.aud script. The default value of this variable is 022.

JASS_FTPUSERS

This variable contains a list of user accounts that should be prevented from using the FTP service. During hardening runs, the `install-ftpusers.fin` script adds each user defined by this variable to either the `JASS_ROOT_DIR/etc/ftpusers` file (Solaris 8 OS or earlier) or the `JASS_ROOT_DIR/etc/ftpd/ftpusers` file (Solaris 9 OS) if not already present.

Similarly, during audit runs, the `install-ftpusers.aud` script determines if each user account defined by this variable is listed in the `ftpusers` file. By default, this variable contains all of the user accounts defined in the `JASS_PASSWD` file with user identifiers less than `100` and greater than `60000`. Typically these ranges are reserved for administrative access.

JASS_KILL_SCRIPT_DISABLE

This variable contains a Boolean value that determines whether the kill run-control scripts should be disabled or simply left in place when a service is disabled. The start run-control scripts are always disabled. Some administrators prefer to have the kill scripts left in place so that any services that are started manually can be properly terminated during a system shutdown or reboot. By default, this variable is set to 1 indicating that the kill run-control scripts should be disabled. Setting this variable to 0 configures the software to ignore kill run-control scripts.

JASS_LOGIN_RETRIES

This variable contains a numeric value specifying the number of consecutive failed login attempts that can occur before the login process logs the failure and terminates the connection. During hardening runs, this setting is installed by the `set-login-retries.fin` script. During audit runs, the `set-login-retries.aud` script checks that this setting is installed. By default, this variable has a value of `3`.

JASS_MD5_DIR

This variable contains a string value representing the absolute path to the MD5 software distribution, if present. If the MD5 software is installed from the `tar` distribution by the Solaris Security Toolkit, it is installed into the directory defined by this variable. During hardening runs, this variable is used by the `install-md5.fin` script to install the MD5 software. During audit runs, `install-md5.aud` script checks for the existence of the MD5 software at the location defined by this variable. The default value of this variable is `/opt`.

JASS_NOVICE_USER

This variable controls the display of information for novice Solaris Security Toolkit users. This variable provides additional guidance for less-experienced administrators. You can disable this capability by setting the `JASS_NOVICE_USER` variable to `0` (zero) in the `JASS_HOME_DIR/Drivers/user.init` file.

JASS_PASS_LENGTH

This variable contains a numeric value specifying the minimum length of a user password. The default value for this variable is `8` (characters). This variable is used by the `set-user-password-reqs.fin` script and the `set-user-password-reqs.aud` script.

JASS_PASSWD

Note – This variable should *not* require modification.

This variable contains a string value that specifies the location of the password file on the target system. This variable is used in many of the scripts and for dynamic assignment of many variables. This variable has a default value of `JASS_ROOT_DIR/etc/passwd`.

JASS_POWER_MGT_USER

This variable contains a string value that defines which users are permitted to perform power management functions on a system. During hardening runs, this restriction is implemented by the `set-power-restrictions.fin` script. During audit runs, this restriction is checked by the `set-power-restrictions.aud` script. The default value of this variable is "`-`", indicating that only the `root` account is permitted to perform these management functions. For more information, see the `/etc/default/power` information in Chapter 2.

JASS_REC_PATCH_OPTIONS

This variable contains a string value that specifies options to be passed to the `patchadd` or `installpatch` commands when installing a Solaris Recommended and Security Patch Cluster on a system. For information on available options, refer to the `patchadd`(1M) manual page or the `installpatch` program code. During

hardening runs, this variable is used by the `install-recommended-patches.fin` script when installing the patch cluster on the system. This variable is not used during audit runs. By default, no options are assigned to this variable.

JASS_RHOSTS_FILE

This variable contains a string value that specifies the file where the list of `.rhosts` or `hosts.equiv` files found on the system are stored. This variable is used during hardening runs by the `print-rhosts.fin` script. This variable is not used during audit runs. By default, no file name is assigned to this variable. As a result, the output of the `print-rhosts.fin` script is displayed on the screen.

JASS_ROOT_GROUP

This variable contains a numeric value that is used as the `root` user's primary group identifier value. During hardening runs, this setting is installed by the `set-root-group.fin` script. During audit runs, this setting is checked by the `set-root-group.aud` script. By default, this value is set to `0` (or root). This value overrides the Solaris OS default value of `1` (or other).

JASS_ROOT_PASSWORD

Caution – Change the value of this string from the default value that ships with the Solaris Security Toolkit software. Failure to do so could leave systems vulnerable because the password is publicly known.

Note – This script operates only when the system is running from a miniroot during a JumpStart installation, to prevent the root password from being accidentally overwritten with a widely known value.

This variable contains a string value that is used as the encrypted password for the root account. During hardening runs, this setting is installed by the `set-root-password.fin` script. During audit runs, this setting is checked by the `set-root-password.aud` script. By default, this variable is set to `JdqZ5HrSDYM.o`. This encrypted string equates to the clear-text string `t00lk1t`.

JASS_SADMIND_OPTIONS

This variable contains a string value specifying options to be used with the `sadmind` daemon that is executed from the `inetd` process. During hardening runs, this setting is installed by the `install-sadmind-options.fin` script. During audit runs, these settings are checked by the `install-sadmind-options.aud` script. By default, this variable has a value of `-S 2` to instruct the `sadmind` daemon to use strong authentication (`AUTH_DES`) when communicating with clients.

JASS_SENDMAIL_MODE

Note – Due to changes in `sendmail` versions and configurations, this variable is applicable only to Solaris 8 OS versions. Other mechanisms are used for newer and earlier Solaris OS versions to achieve the same goal.

Note – The back slash characters are required in this string to prevent the quotation marks from being interpreted by the command shell. When installed in the relevant `sendmail` configuration file, the string displays as "".

This variable contains a string value specifying options to be used by the `sendmail` daemon to determine its operation. During hardening runs, the `disable-sendmail.fin` script configures the daemon for the operation specified by this variable. During audit runs, the `disable-sendmail.aud` script checks to ensure that the `sendmail` daemon is configured for the correct operation. The default value of this variable is `\"\"`. This value indicates that the `sendmail` daemon should operate in queue-processing mode only. This value overrides the default value where the `sendmail` daemon is configured to operate as a daemon and receive incoming mail.

JASS_SGID_FILE

This variable contains a string value that specifies the file where the list of `set-group-id` files found on the system are stored. This variable is used during hardening runs by the `print-sgid-files.fin` script. This variable is not used during audit runs. By default, no file name is assigned to this variable. As a result, the output of the `print-sgid-files.fin` script is displayed on the screen.

JASS_SHELLS

This variable contains a list of shells to add to the JASS_ROOT_DIR/etc/shells file. During hardening runs, the install-shells.fin script adds each shell defined by this variable to the JASS_ROOT_DIR/etc/shells file, if not already present. Similarly, during audit runs, the install-shells.aud script determines if each shell defined by this variable is listed in the shells file.

The default value for this variable are as follows:

- /bin/csh
- /bin/jsh
- /bin/ksh
- /bin/sh
- /sbin/sh
- /sbin/jsh
- /usr/bin/csh
- /usr/bin/jsh
- /usr/bin/ksh
- /usr/bin/sh

For Solaris OS versions 8 and newer, the following shells are added to the default value:

- /bin/bash
- /bin/pfcsh
- /bin/pfksh
- /bin/pfsh
- /bin/tcsh
- /bin/zsh
- /usr/bin/bash
- /usr/bin/pfcsh
- /usr/bin/pfksh
- /usr/bin/pfsh
- /usr/bin/tcsh
- /usr/bin/zsh

JASS_SHELL_DISABLE

This variable contains a file name that specifies the location of the shell used when disabling user accounts. During hardening runs, this variable is used by the disable-system-accounts.fin script when installing the shell for the accounts defined by JASS_ACCT_DISABLE variable. During audit runs, this variable is used by the disable-system-accounts.aud script to check that the shell program exists on the system and that the accounts defined by JASS_ACCT_DISABLE are configured to use the shell.

JASS_SUID_FILE

This variable contains a string value that specifies the file where the list of `set-user-id` files found on the system are stored. This variable is used during hardening runs by the `print-suid-files.fin` script. This variable is not used during audit runs. By default, no file name is assigned to this variable. As a result, the output of the `print-suid-files.fin` script is displayed on the screen.

JASS_SUSPEND_PERMS

This variable contains a string value that defines which users are permitted to perform system suspend or resume functions. During hardening runs, this restriction is implemented by the `set-sys-suspend-restrictions.fin` script. During audit runs, this restriction is checked by the `set-sys-suspend-restrictions.aud` script. The default value of this variable is "`-`", indicating that only the `root` account is permitted to perform these management functions. For more information, refer to the `/etc/default/sys-suspend` file.

JASS_SVCS_DISABLE

Caution – When using the default list of services, be certain to have either console access to the system, Secure Shell access (for Solaris 9 OS), or a nondefault remote access capability because Telnet, RSH, and RLOGIN servers are all disabled by default.

This variable simplifies the removal of different services from the `JASS_ROOT_DIR/etc/inet/inetd.conf` file. During hardening runs, the `update-inetd-conf.fin` script disables each `inetd` service defined by this variable, unless it is also listed in the `JASS_SVCS_ENABLE` variable. Similarly, during audit runs, the `update-inetd-conf.aud` script determines that the appropriate `inetd` services are disabled on the system. By default, the list of services disabled by this variable includes all of the entries that are provided by default with the Solaris OS.

The `JASS_SVCS_DISABLE` and `JASS_SVCS_ENABLE` variables provide a straightforward and easy-to-use mechanism for modifying the default behavior of `update-inetd-conf.fin` without requiring any modifications to the script itself. The four configuration possibilities for modifying these variables are as follows:

Example 1:

`JASS_SVCS_DISABLE` (defined)

JASS_SVCS_ENABLE (not defined)

This example is the default case for backward compatibility with older versions of the Solaris Security Toolkit software. In this case, the services listed in JASS_SVCS_DISABLE are commented out of the /etc/inetd.conf file when the update-inetd-conf.fin script is run.

Example 2:

JASS_SVCS_DISABLE (not defined)

JASS_SVCS_ENABLE (defined)

Only services listed in JASS_SVCS_ENABLE are left enabled. All other services, including those that are not Sun specific, are disabled. This example permits the implementation of the principle "all that is not explicitly permitted is denied."

Example 3:

JASS_SVCS_DISABLE (defined)

JASS_SVCS_ENABLE (defined)

The services in JASS_SVCS_DISABLE are disabled and JASS_SVCS_ENABLE are left enabled. Services not covered in the list are unaffected. If a service is listed in both JASS_SVCS_ENABLE and JASS_SVCS_DISABLE, then it is enabled because JASS_SVCS_ENABLE takes precedence.

Example 4:

JASS_SVCS_DISABLE (undefined)

JASS_SVCS_ENABLE (undefined)

In this example, none of the services are affected because there is no explicit direction defined.

JASS_SVCS_ENABLE

This variable contains a list of inetd services that are expected to be enabled on a system. During hardening runs, the update-inetd-conf.fin finish script enables any service listed in this variable that is currently disabled. If the service is already enabled, no action is taken. During audit runs, the update-inetd-conf.aud script determines if the services defined by this variable are enabled on the system. By default, this variable contains no services. As a result, the behavior of the update-inetd-conf.fin script and update-inetd-conf.aud script is controlled solely by the contents of the JASS_SVCS_DISABLE variable.

JASS_TMPFS_SIZE

Note – Adjust this variable to ensure that it is large enough to meet the current and expected `/tmp` needs for system functions and applications running on the system.

This variable contains a string value representing the amount of space to allocate to the `/tmp` (`tmpfs`) file system. During hardening runs, this setting is installed by the `set-tmpfs-limit.fin` script. During audit runs, this setting is checked by the `set-tmpfs-limit.aud` script. This variable has a default value of `512` megabytes.

JASS_UMASK

This variable contains a numeric (octal) value that represents the file creation mask (`umask`). During hardening runs, this setting is used by the `set-system-umask.fin` and `set-user-umask.fin` scripts. During audit runs, this setting is checked by the `set-system-umask.aud` and `set-user-umask.aud` scripts. The default value of this variable is `022`.

JASS_UNOWNED_FILE

This variable contains a string value that specifies the file where the list of unowned files found on the system are stored. A file is considered unowned if its user or group assignment does not correspond to a valid user or group on the system. This variable is used during hardening runs by the `print-unowned-objects.fin` script. This variable is not used during audit runs. By default, no file name is assigned to this variable. As a result, the output of the `print-unowned-objects.fin` script is displayed on the screen.

JASS_WRITABLE_FILE

This variable contains a string value that specifies the file where the list of world-writable files found on the system are stored. This variable is used during hardening runs by the `print-world-writable-objects.fin` script. This variable is not used during audit runs. By default, no file name is assigned to this variable. As a result, the output of the `print-world-writable-objects.fin` script is displayed on the screen.

Define JumpStart Mode Variables

JumpStart mode variables are those that are defined and used by the Solaris Security Toolkit software solely when operating in JumpStart mode. These variables facilitate the use of the Solaris Security Toolkit software either as a JumpStart framework or integrated as part of a larger build environment. These variables are mentioned separately because they are only relevant during a JumpStart installation.

These variables are defined in the `JASS_HOME_DIR/Drivers/user.init` file. They are in the `user.init` file because they typically require modification in contrast to most of the other variables that can be used as-is with no modification.

Note – In some cases, such as with multihomed JumpStart servers, special customization might be required.

Tune these variables where necessary to best suit the environment in which the Solaris Security Toolkit software is used.

This section describes the following JumpStart mode variables:

JASS_PACKAGE_MOUNT

This variable defines the named resource or location where the Solaris Security Toolkit software expects to find the software packages that it might be required to install onto a client. This resource is defined as an NFS path of the form: *`host name`*`:`*`/path/to/software`*. This resource is mounted to `JASS_PACKAGE_DIR` by the `mount_filesystems` function during the execution of the `driver.run` script.

The location of this resource must be specified by host name or IP address, and the complete path must be listed to provide the NFS daemon enough information to mount the directory during a run. Because a host name or IP address can be specified in the value of the environment variable, it often requires modification.

The Solaris Security Toolkit software attempts to configure the correct host name and directory path automatically; however, this automatic configuration might not be applicable to your environment. By default, this variable is set to `HOSTNAME:/jumpstart/Packages`. The `HOSTNAME` variable is dynamically assigned to the address of the NFS server from which the client has mounted the `/cdrom` file system.

JASS_PATCH_MOUNT

This variable defines the named resource or location where the Solaris Security Toolkit software should expect to find the software patches that it may be required to install onto the client. This resource is defined as an NFS path of the form: *host name*:*/path/to/patches*. This resource is mounted to `JASS_PATCH_DIR` by the `mount_filesystems` function during the execution of the `driver.run` script.

The location of this resource must be specified by host name or IP address, and the complete path must be listed to provide the NFS daemon enough information to mount the directory during the Toolkit run. Because a host name or IP address can be specified in the value of the environment variable, it often requires modification.

The Solaris Security Toolkit software attempts to configure the correct host name and directory path automatically; however, this automatic configuration might not be applicable to your environment. By default, this variable is set to `HOSTNAME:/jumpstart/Patches`. The `HOSTNAME` variable is dynamically assigned to the address of the NFS server from which the client has mounted the `/cdrom` file system.

Glossary

This list defines abbreviations and acronyms in the Solaris Security Toolkit.

A

ab2 AnswerBook2

ABI Application Binary Interface

ARP Address Resolution Protocol

ASPPP Asynchronous Point-to-Point Protocol

B

BIND Berkeley Internet Name Domain

BSD Berkeley Software Distribution

BSM Basic Security Model *(Solaris)*

C

CD compact disc

CD-ROM compact disc–read-only memory

CDE Common Desktop Environment

`cp`**(1)** copy files

`cron`**(1M)** clock daemon

D

DHCP Dynamic Host Configuration Protocol

DMI Desktop Management Interface

DMTF Distributed Management Task Force

DNS Domain Name System

E

EEPROM electronically erasable programmable read-only memory

F

FTP File Transfer Protocol

G

GID group identifier

H

HTTP HyperText Transfer Protocol

I

ID identifier

IETF Internet Engineering Task Force

INETD Internet service daemon

IP Internet Protocol

ISA instruction set architecture

J

JASS JumpStart Architecture and Security Scripts, *now* Solaris Security Toolkit

K

KDC Kerberos Key Distribution

L

LDAP Lightweight Directory Access Protocol

`lp`**(1)** line printer *(submit print request)*

M

MAN management network *(Sun Fire High-End Systems internal I1 network)*

MD5 message-digest 5 algorithm

MIP Mobile Internet Protocol

MSP midframe service processor

`mv`**(1)** move files

N

NFS Network File System

NG Next Generation

NIS, NIS+ Network Information Services

NSCD name service cache daemon

O

OE operating environment, *formerly used for Solaris*

OEM Original Equipment Manufacturer

OS Operating System, *now used for Solaris*

P

PAM Pluggable Authentication Module

PDF Portable Document Format

PICL Platform Information and Control Library

PPP Point-to-Point Protocol

PROM programmable read-only memory

Q

QA quality assurance

R

RBAC role-based access control

rc run-control *(file or script)*

`rlogin`**(1)** remote login

RFC Remote Function Call

RPC Remote Procedure Call

`rsh`**(1)** remote shell

S

SA system administrator

SC system controller *(Sun Fire High-End and Midrange Systems)*

`scp`**(1)** secure copy (remote file copy program)

SCCS Source Code Control System

SLP Service Location Protocol

SMA System Management Agent

SMC Solaris Management Console

SNMP Simple Network Management Protocol

SP service provider

SPARC Scalable Processor Architecture

SPC SunSoft Print Client

SSH Secure Shell *(Solaris)*

SSP system service processor *(Sun Enterprise 1000 Servers)*

`stdio` standard input/output

SunONE Sun Open Network Environment, *currently* Sun Java System, *formerly* iPlanet

T

TCP Transmission Control Protocol

`tftp`**(1)** trivial file transfer program

`ttl` time-to-live

U

U.S. United States

UDP User Diagram Protocol

UID user identifier

UUCP UNIX-to-UNIX Copy

V

VOLD Volume Management daemon

W

WBEM Web-based Enterprise Management

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X