Securing Files and Verifying File Integrity in Oracle® Solaris 11.2

Part No: E37122
July 2014

ORACLE

Contents

Using This Documentation

- **Overview** – Describes how to protect legitimate files, view hidden file permissions, and locate and prevent the execution of rogue files. Also describes how to verify the integrity of files over time on Oracle Solaris systems.
- **Audience** – System administrators.
- **Required knowledge** – Site security requirements.

Product Documentation Library

Late-breaking information and known issues for this product are included in the documentation library at http://www.oracle.com/pls/topic/lookup?ctx=E36784.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

♦ ♦ ♦ CHAPTER 1

Controlling Access to Files

This chapter describes how to protect files in Oracle Solaris. The chapter also describes how to protect the system from files whose permissions could compromise the system.

Note - ZFS is the default OS file system. To protect ZFS files with access control lists (ACLs), see Chapter 7, "Using ACLs and Attributes to Protect Oracle Solaris ZFS Files," in "Managing ZFS File Systems in Oracle Solaris 11.2 ".

This chapter covers the following topics:

Using UNIX Permissions to Protect Files

You can secure files through UNIX file permissions and through ACLs. Files with sticky bits, and files that are executable, require special security measures.

Commands for Viewing and Securing Files

This table describes the commands for monitoring and securing files and directories.

TABLE 1-1 Commands for Securing Files and Directories

Command	Description	Man Page
`ls`	Lists the files in a directory and information about the files.	`ls`(1)
`chown`	Changes the ownership of a file.	`chown`(1)

Command	Description	Man Page
`chgrp`	Changes the group ownership of a file.	`chgrp`(1)
`chmod`	Changes permissions on a file. You can use either symbolic mode, which uses letters and symbols, or absolute mode, which uses octal numbers, to change permissions on a file.	`chmod`(1)

File and Directory Ownership

Traditional UNIX file permissions can assign ownership to three classes of users:

- **user –** The file or directory owner, which is usually the user who created the file. The owner of a file can decide who has the right to read the file, to write to the file (make changes to it), or, if the file is a command, to execute the file.
- **group –** Members of a group of users.
- **others –** All other users who are not the file owner and are not members of the group.

The owner of the file can usually assign or modify file permissions. Additionally, the `root` account can change a file's ownership. To override system policy, see Example 1-2.

A file can be one of seven types. Each type is displayed by a symbol:

- (Minus symbol)	Text or program
b	Block special file
c	Character special file
d	Directory
l	Symbolic link
s	Socket
D	Door
P	Named pipe (FIFO)

UNIX File Permissions

The following table lists and describes the permissions that you can give to each class of user for a file or directory.

TABLE 1-2 File and Directory Permissions

Symbol	Permission	Object	Description
`r`	Read	File	Designated users can open and read the contents of a file.
		Directory	Designated users can list files in the directory.
`w`	Write	File	Designated users can modify the contents of the file or delete the file.
		Directory	Designated users can add files or add links in the directory. They can also remove files or remove links in the directory.
`x`	Execute	File	Designated users can execute the file, if it is a program or shell script. They also can run the program with one of the `exec(2)` system calls.
		Directory	Designated users can open files or execute files in the directory. They also can make the directory and the directories beneath it current.
`-`	Denied	File and Directory	Designated users cannot read, write, or execute the file.

These file permissions apply to regular files, and to special files such as devices, sockets, and named pipes (FIFOs).

For a symbolic link, the permissions that apply are the permissions of the file that the link points to.

You can protect the files in a directory and its subdirectories by setting restrictive file permissions on that directory. Note, however, that the `root` role has access to all files and directories on the system.

Special File Permissions Using `setuid`, `setgid` and Sticky Bit

Three special types of permissions are available for executable files and public directories: `setuid`, `setgid`, and sticky bit. When these permissions are set, any user who runs that executable file assumes the ID of the owner (or group) of the executable file.

You must be extremely careful when you set special permissions, because special permissions constitute a security risk. For example, a user can gain `root` capabilities by executing a program that sets the user ID (UID) to `0`, which is the UID of `root`. Also, all users can set special permissions for files that they own, which constitutes another security concern.

You should monitor your system for any unauthorized use of the `setuid` permission and the `setgid` permission to gain `root` capabilities. A suspicious permission grants ownership of an administrative program to a user rather than to `root` or `bin`. To search for

and list all files that use this special permission, see "How to Find Files With Special File Permissions" on page 22.

`setuid` Permission

When `setuid` permission is set on an executable file, a process that runs this file is granted access on the basis of the owner of the file. The access is *not* based on the user who is running the executable file. This special permission allows a user to access files and directories that are normally available only to the owner.

For example, the `setuid` permission on the `passwd` command makes it possible for users to change passwords. A `passwd` command with `setuid` permission would resemble the following:

```
-r-sr-sr-x   1 root     sys       56808 Jun 17 12:02 /usr/bin/passwd
```

This special permission presents a security risk. Some determined users can find a way to maintain the permissions that are granted to them by the `setuid` process even after the process has finished executing.

Note - The use of `setuid` permissions with the reserved UIDs (0-100) from a program might not set the effective UID correctly. Use a shell script, or avoid using the reserved UIDs with `setuid` permissions.

`setgid` Permission

The `setgid` permission is similar to the `setuid` permission. The process's effective group ID (GID) is changed to the group that owns the file, and a user is granted access based on the permissions that are granted to that group. The `/usr/bin/mail` command has `setgid` permissions:

```
-r-x--s--x   1 root   mail     71212 Jun 17 12:01 /usr/bin/mail
```

When the `setgid` permission is applied to a directory, files that are created in this directory belong to the group that owns the directory. The files do not belong to the group to which the creating process belongs. Any user who has write and execute permissions in the directory can create a file there. However, the file belongs to the group that owns the directory, not to the group that the user belongs to.

You should monitor your system for any unauthorized use of the `setgid` permission to gain `root` capabilities. A suspicious permission grants group access to such a program to an unusual

group rather than to `root` or `bin`. To search for and list all files that use this permission, see "How to Find Files With Special File Permissions" on page 22.

Sticky Bit

The *sticky bit* is a permission bit that protects the files within a directory. If the directory has the sticky bit set, a file can be deleted only by the file owner, the directory owner, or by a privileged user. The `root` user is an example of a privileged user. The sticky bit prevents a user from deleting other users' files from public directories such as `/tmp`:

```
drwxrwxrwt 7  root  sys   400 Sep  3 13:37 tmp
```

Be sure to set the sticky bit manually when you set up a public directory on a TMPFS file system. For instructions, see Example 1-5.

Default `umask` Value

When you create a file or directory, you create it with a default set of permissions. The system defaults are open. A text file has `666` permissions, which grants read and write permission to everyone. A directory and an executable file have `777` permissions, which grants read, write, and execute permission to everyone. Typically, users override the system defaults in their shell initialization files, such as `.bashrc` and `.kshrc.user`. An administrator can also set defaults in the `/etc/profile` file.

The value that the `umask` command assigns is subtracted from the default. This process has the effect of denying permissions in the same way that the `chmod` command grants them. For example, the `chmod 022` command grants write permission to group and others. The `umask 022` command denies write permission to group and others.

The following table shows some typical `umask` values and their effect on an executable file.

TABLE 1-3 `umask` Settings for Different Security Levels

Level of Security	`umask` Setting	Permissions Disallowed
Permissive (`744`)	`022`	`w` for group and others
Moderate (`751`)	`026`	`w` for group, `rw` for others
Strict (`740`)	`027`	`w` for group, `rwx` for others
Severe (`700`)	`077`	`rwx` for group and others

For more information about setting the `umask` value, see the `umask`(1) man page.

File Permission Modes

The `chmod` command enables you to change the permissions on a file. You must be `root` or the owner of a file or directory to change its permissions.

You can use the `chmod` command to set permissions in either of two modes:

- **Absolute Mode –** Use numbers to represent file permissions. When you change permissions by using the absolute mode, you represent permissions for each triplet by an octal mode number. Absolute mode is the method most commonly used to set permissions.
- **Symbolic Mode –** Use combinations of letters and symbols to add permissions or remove permissions.

The following table lists the octal values for setting file permissions in absolute mode. You use these numbers in sets of three to set permissions for owner, group, and other, in that order. For example, the value `644` sets read and write permissions for owner, and read-only permissions for group and other.

TABLE 1-4 Setting File Permissions in Absolute Mode

Octal Value	File Permissions Set	Permissions Description
`0`	`---`	No permissions
`1`	`--x`	Execute permission only
`2`	`-w-`	Write permission only
`3`	`-wx`	Write and execute permissions
`4`	`r--`	Read permission only
`5`	`r-x`	Read and execute permissions
`6`	`rw-`	Read and write permissions
`7`	`rwx`	Read, write, and execute permissions

The following table lists the symbols for setting file permissions in symbolic mode. Symbols can specify whose permissions are to be set or changed, the operation to be performed, and the permissions that are being assigned or changed.

TABLE 1-5 Setting File Permissions in Symbolic Mode

Symbol	Function	Description
`u`	*who*	User (owner)

Symbol	Function	Description
`g`	*who*	Group
`o`	*who*	Others
`a`	*who*	All
`=`	*operator*	Assign
`+`	*operator*	Add
`-`	*operator*	Remove
`r`	*permissions*	Read
`w`	*permissions*	Write
`x`	*permissions*	Execute
`l`	*permissions*	Mandatory locking, `setgid` bit is on, group execution bit is off
`s`	*permissions*	`setuid` or `setgid` bit is on
`t`	*permissions*	Sticky bit is on, execution bit for others is on

The *who operator permissions* designations in the function column specify the symbols that change the permissions on the file or directory.

who Specifies whose permissions are to be changed.

operator Specifies the operation to be performed.

permissions Specifies what permissions are to be changed.

You can set special permissions on a file in absolute mode or symbolic mode. However, you must use symbolic mode to set or remove `setuid` permissions on a directory. In absolute mode, you set special permissions by adding a new octal value to the left of the permission triplet. See Example 1-5. The following table lists the octal values for setting special permissions on a file.

TABLE 1-6 Setting Special File Permissions in Absolute Mode

Octal Value	Special File Permissions
`1`	Sticky bit
`2`	`setgid`
`4`	`setuid`

Using Access Control Lists to Protect UFS Files

Traditional UNIX file protection provides read, write, and execute permissions for the three user classes: file owner, file group, and other. In a UFS file system, an access control list (ACL) provides better file security by enabling you to do the following:

- Define file permissions for the file owner, the group, other, specific users and groups
- Define default permissions for each of the preceding categories

---

Note - For ACLs in the ZFS file system and ACLs on NFSv4 files, see Chapter 7, "Using ACLs and Attributes to Protect Oracle Solaris ZFS Files," in "Managing ZFS File Systems in Oracle Solaris 11.2 ".

For example, if you want everyone in a group to be able to read a file, you can simply grant group read permissions on that file. However, if you want only one person in the group to be able to write to that file, you can use an ACL.

For more information about ACLs on UFS file systems, see *System Administration Guide: Security Services* for the Oracle Solaris 10 release.

Protecting Executable Files From Compromising Security

Programs read and write data on the stack. Typically, they execute from read-only portions of memory that are specifically designated for code. Some attacks that cause buffers on the stack to overflow try to insert new code on the stack and cause the program to execute it. Removing execute permission from the stack memory prevents these attacks from succeeding. That is, most programs can function correctly without using executable stacks.

64-bit processes always have non-executable stacks. By default, 32-bit SPARC processes have executable stacks. The `noexec_user_stack` variable enables you to specify whether the stacks of 32-bit processes are executable.

Once this variable is set, programs that attempt to execute code on their stack are sent a `SIGSEGV` signal. This signal usually results in the program terminating with a core dump. Such programs also generate a warning message that includes the name of the offending program, the process ID, and the real UID of the user who ran the program. For example:

```
a.out[347] attempt to execute code on stack by uid 555
```

The message is logged by the `syslog` daemon when the `syslog kern` facility is set to `notice` level. This logging is set by default in the `syslog.conf` file, which means that the message is sent to both the console and the `/var/adm/messages` file. For more information, see the `syslogd`(1M) and `syslog.conf`(4) man pages.

The `syslog` message is useful for observing potential security problems. The message also identifies valid programs that depend upon executable stacks that have been prevented from correct operation by setting the `noexec_user_stack` variable. If you do not want any messages logged, then set the log variable, `noexec_user_stack_log`, to zero in the `/etc/system` file. Even though messages are not being logged, the `SIGSEGV` signal can continue to cause the executing program to terminate with a core dump.

Programs can explicitly mark or prevent stack execution. The `mprotect` function in programs explicitly marks the stack as executable. For more information, see the `mprotect`(2) man page. A program compiled with `-M /usr/lib/ld/map.noexstk` makes the stack non-executable regardless of the system-wide setting.

Protecting Files

The following procedures protect files with UNIX permissions, locate files with security risks, and protect the system from compromise by these files.

Protecting Files With UNIX Permissions

The following task map points to procedures that list file permissions, change file permissions, and protect files with special file permissions.

Task	For Instructions
Display file information.	“How to Display File Information” on page 15
Change local file ownership.	“How to Change the Owner of a File” on page 17 “How to Change Group Ownership of a File” on page 18
Change local file permissions.	“How to Change File Permissions in Symbolic Mode” on page 18 “How to Change File Permissions in Absolute Mode” on page 19 “How to Change Special File Permissions in Absolute Mode” on page 21

▼ How to Display File Information

Display information about all the files in a directory by using the `ls` command.

- **Type the following command to display a long listing of all files in the current directory.**

```
% ls -la
```

`-l` Displays the long format that includes user ownership, group ownership, and file permissions.

`-a` Displays all files, including hidden files that begin with a dot (`.`).

For all options to the `ls` command, see the `ls`(1) man page.

Example 1-1 Displaying File Information

In the following example, a partial list of the files in the `/sbin` directory is displayed.

```
% cd /sbin
% ls -l
total 4960
-r-xr-xr-x   1 root   bin      12756 Dec 19  2013 6to4relay
lrwxrwxrwx   1 root   root        10 Dec 19  2013 accept -> cupsaccept
-r-xr-xr-x   1 root   bin      38420 Dec 19  2013 acctadm
-r-xr-xr-x   2 root   sys      70512 Dec 19  2013 add_drv
-r-xr-xr-x   1 root   bin       3126 Dec 19  2013 addgnupghome
drwxr-xr-x   2 root   bin         37 Dec 19  2013 amd64
-r-xr-xr-x   1 root   bin       2264 Dec 19  2013 applygnupgdefaults
-r-xr-xr-x   1 root   bin        153 Dec 19  2013 archiveadm
-r-xr-xr-x   1 root   bin      12644 Dec 19  2013 arp
.
.
.
```

Each line displays information about a file in the following order:

- Type of file – For example, `d`. For list of file types, see "File and Directory Ownership" on page 8.
- Permissions – For example, `r-xr-xr-x`. For description, see "File and Directory Ownership" on page 8.
- Number of hard links – For example, `2`.
- Owner of the file – For example, `root`.
- Group of the file – For example, `bin`.
- Size of the file, in bytes – For example, `12644`.
- Date the file was created or the last date that the file was changed – For example, `Dec 19 2013`.
- Name of the file – For example, `arp`.

How to Change the Owner of a File

Before You Begin If you are not the owner of the file or directory, you must be assigned the Object Access Management rights profile. To change a file that is a public object, you must assume the `root` role.

For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

1. **Display the permissions on a local file.**

```
% ls -l example-file
-rw-r--r--   1 janedoe   staff   112640 May 24 10:49 example-file
```

2. **Change the owner of the file.**

```
# chown stacey example-file
```

3. **Verify that the owner of the file has changed.**

```
# ls -l example-file
-rw-r--r--   1 stacey   staff   112640 May 26 08:50 example-file
```

To change permissions on NFS-mounted files, see Chapter 5, "Commands for Managing Network File Systems," in "Managing Network File Systems in Oracle Solaris 11.2 ".

Example 1-2 Enabling Users to Change the Ownership of Their Own Files

Security Consideration – You need a good reason to change the setting of the `rstchown` variable to zero. The default setting prevents users from listing their files as belonging to others so as to bypass space quotas.

In this example, the value of the `rstchown` variable is set to zero in the `/etc/system` file. This setting enables the owner of a file to use the `chown` command to change the file's ownership to another user. This setting also enables the owner to use the `chgrp` command to set the group ownership of a file to a group that the owner does not belong to. The change goes into effect when the system is rebooted.

```
set rstchown = 0
```

For more information, see the `chown`(1) and `chgrp`(1) man pages.

How to Change Group Ownership of a File

Before You Begin If you are not the owner of the file or directory, you must be assigned the Object Access Management rights profile. To change a file that is a public object, you must assume the `root` role.

For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

1. **Change the group ownership of a file.**

```
% chgrp scifi example-file
```

For information about setting up groups, see Chapter 1, "About User Accounts and User Environments," in "Managing User Accounts and User Environments in Oracle Solaris 11.2 ".

2. **Verify that the group ownership of the file has changed.**

```
% ls -l example-file
-rw-r--r--   1 stacey   scifi   112640 June 20 08:55  example-file
```

Also see Example 1-2.

How to Change File Permissions in Symbolic Mode

In the following procedure, a user changes permissions on a file that the user owns.

1. **Change permissions in symbolic mode.**

```
% chmod who operator permissions filename
```

who	Specifies whose permissions are to be changed.
operator	Specifies the operation to be performed.
permissions	Specifies what permissions are to be changed. For the list of valid symbols, see Table 1-5.
filename	Specifies the file or directory.

2. **Verify that the permissions of the file have changed.**

```
% ls -l filename
```

Note - If you are not the owner of the file or directory, you must be assigned the Object Access Management rights profile. To change a file that is a public object, you must assume the `root` role.

Example 1-3 Changing Permissions in Symbolic Mode

In the following example, the owner removes read permission others.

```
% chmod o-r example-file1
```

the following example, the owner adds read and execute permissions for user, group, and others.

```
% chmod a+rx example-file2
```

In the following example, the owner adds read, write, and execute permissions for group members.

```
% chmod g=rwx example-file3
```

▼ How to Change File Permissions in Absolute Mode

In the following procedure, a user changes permissions on a file that the user owns.

1. **Change permissions in absolute mode.**

```
% chmod nnn filename
```

nnn — Specifies the octal values that represent the permissions for the file owner, file group, and others, in that order. For the list of valid octal values, see Table 1-4.

filename — Specifies the file or directory.

Note - If you use the `chmod` command to change file or directory permissions on objects that have existing ACL entries, the ACL entries might change as well. The exact changes are dependent upon the `chmod` permission operation changes and the file system's `aclmode` and `aclinherit` property values.

For more information, see Chapter 7, "Using ACLs and Attributes to Protect Oracle Solaris ZFS Files," in "Managing ZFS File Systems in Oracle Solaris 11.2 ".

2. **Verify that the permissions of the file have changed.**

```
% ls -l filename
```

Note - If you are not the owner of the file or directory, you must be assigned the Object Access Management rights profile. To change a file that is a public object, you must assume the `root` role.

Example 1-4 Changing Permissions in Absolute Mode

In the following example, the administrator changes the permissions of a directory that is open to the public from `744` (read, write, execute; read-only; and read-only) to `755` (read, write, execute; read and execute; and read and execute).

```
# ls -ld public_dir
drwxr--r--  1 jdoe   staff    6023 Aug  5 12:06 public_dir
# chmod 755 public_dir
# ls -ld public_dir
drwxr-xr-x  1 jdoe   staff    6023 Aug  5 12:06 public_dir
```

In the following example, the file owner changes the permissions of an executable shell script from read and write to read, write, and execute.

```
% ls -l my_script
-rw------- 1 jdoe   staff    6023 Aug  5 12:06 my_script
% chmod 700 my_script
% ls -l my_script
-rwx------ 1 jdoe   staff    6023 Aug  5 12:06 my_script
```

▼ How to Change Special File Permissions in Absolute Mode

Before You Begin If you are not the owner of the file or directory, you must be assigned the Object Access Management rights profile. To change a file that is a public object, you must assume the `root` role.

For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

1. Change special permissions in absolute mode.

```
% chmod nnnn filename
```

nnnn	Specifies the octal values that change the permissions on the file or directory. The leftmost octal value sets the special permissions on the file. For the list of valid octal values for special permissions, see Table 1-6.
filename	Specifies the file or directory.

Note - When you use the `chmod` command to change the file group permissions on a file with ACL entries, both the file group permissions and the ACL mask are changed to the new permissions. Be aware that the new ACL mask permissions can change the permissions for additional users and groups who have ACL entries on the file. Use the `getfacl` command to make sure that the appropriate permissions are set for all ACL entries. For more information, see the `getfacl`(1) man page.

2. Verify that the permissions of the file have changed.

```
% ls -l filename
```

Example 1-5 Setting Special File Permissions in Absolute Mode

In the following example, the administrator sets the `setuid` permission on the `dbprog` file.

```
# chmod 4555 dbprog
# ls -l dbprog
-r-sr-xr-x   1 db     staff        12095 May  6 09:29 dbprog
```

In the following example, the administrator sets the `setgid` permission on the `dbprog2` file.

```
# chmod 2551 dbprog2
# ls -l dbprog2
-r-xr-s--x   1 db     staff       24576 May  6 09:30 dbprog2
```

In the following example, the administrator sets the sticky bit on the `public_dir` directory.

```
# chmod 1777 public_dir
# ls -ld public_dir
drwxrwxrwt   2 jdoe   staff          512 May 15 15:27 public_dir
```

Protecting Against Programs With Security Risk

The following task map points to procedures that find risky executables on the system, and that prevent programs from exploiting an executable stack.

Task	Description	For Instructions
Find files with special permissions.	Locates files with the `setuid` bit set, but that are not owned by the `root` user.	"How to Find Files With Special File Permissions" on page 22
Prevent executable stack from overflowing.	Prevents programs from exploiting an executable stack.	"How to Disable Programs From Using Executable Stacks" on page 23
Prevent logging of executable stack messages.	Turns off logging of executable stack messages.	Example 1-7

▼ How to Find Files With Special File Permissions

This procedure locates potentially unauthorized use of the `setuid` and `setgid` permissions on programs. A suspicious executable file grants ownership to a user rather than to `root` or `bin`.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

1. Find files with `setuid` permissions by using the `find` command.

```
# find directory -user root -perm -4000 -exec ls -ldb {} \; >/tmp/filename
```

`find` *directory* — Checks all mounted paths starting at the specified *directory*, which can be root (`/`), `/usr`, `/opt`, and so on.

`-user root` — Displays files owned only by `root`.

`-perm -4000` — Displays files only with permissions set to `4000`.

`-exec ls -ldb` — Displays the output of the `find` command in `ls -ldb` format. See the `ls`(1) man page.

`/tmp/`*filename* Is the file that contains the results of the `find` command.

For more information, see the `find`(1).

2. **Display the results in `/tmp/`*filename*.**

```
# more /tmp/filename
```

For background information, see "`setuid` Permission" on page 10.

Example 1-6 Finding Files With `setuid` Permissions

The output from the following example shows that a user in a group called `rar` has made a personal copy of `/usr/bin/rlogin`, and has set the permissions as `setuid` to `root`. As a result, the `/usr/rar/bin/rlogin` program runs with `root` permissions.

After investigating the `/usr/rar` directory and removing the `/usr/rar/bin/rlogin` command, the administrator archives the output from the `find` command.

```
# find /usr -user root -perm -4000 -exec ls -ldb {} \; > /var/tmp/ckprm
# cat /var/tmp/ckprm
-rwsr-xr-x  1 root   sys   28000  Jul 14 14:14  /usr/bin/atq
-rwsr-xr-x  1 root   sys   32364  Jul 14 14:14  /usr/bin/atrm
-r-sr-xr-x  1 root   sys   41432  Jul 14 14:14  /usr/bin/chkey
-rwsr-xr-x  1 root   bin   82804  Jul 14 14:14  /usr/bin/cdrw
-r-sr-xr-x  1 root   bin    8008  Jul 14 14:14  /usr/bin/mailq
-r-sr-sr-x  1 root   sys   45348  Jul 14 14:14  /usr/bin/passwd
-rwsr-xr-x  1 root   bin   37724  Jul 14 14:14  /usr/bin/pfedit
-r-sr-xr-x  1 root   bin   51440  Jul 14 14:14  /usr/bin/rcp
---s--x---  1 root   rar   41592  Jul 24 16:14  /usr/rar/bin/rlogin
-r-s--x--x  1 root   bin  166908  Jul 14 14:14  /usr/bin/sudo
-r-sr-xr-x  4 root   bin   24024  Jul 14 14:14  /usr/bin/uptime
-r-sr-xr-x  1 root   bin   79488  Jul 14 14:14  /usr/bin/xlock
# mv /var/tmp/ckprm /var/share/sysreports/ckprm
```

How to Disable Programs From Using Executable Stacks

For a description of the security risks of 32-bit executable stacks, see "Protecting Executable Files From Compromising Security" on page 14.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

1. **Edit the `/etc/system` file, and add the following lines:**

```
# pfedit /etc/system
...
set noexec_user_stack=1
set noexec_user_stack_log=1
```

2. **Reboot the system.**

```
# reboot
```

Example 1-7 Disabling the Logging of Executable Stack Messages

In this example, the administrator disables logging of executable stack messages, then reboots the system.

```
# cat /etc/system
set noexec_user_stack=1
set noexec_user_stack_log=0
# reboot
```

See Also For more information, read the following:

- https://blogs.oracle.com/gbrunett/entry/solaris_non_executable_stack_overview
- https://blogs.oracle.com/gbrunett/entry/solaris_non_executable_stack_continued
- https://blogs.oracle.com/gbrunett/entry/solaris_non_executable_stack_concluded

♦ ♦ ♦ CHAPTER 2

Verifying File Integrity by Using BART

This chapter describes the file integrity tool, BART. BART is a command-line tool that enables you to verify the integrity of files on a system over time. This chapter covers the following topics:

About BART

BART is a file integrity scanning and reporting tool that uses cryptographic-strength checksums and file system metadata to determine changes. BART can help you detect security breaches or troubleshoot performance issues on a system by identifying corrupted or unusual files. Using BART can reduce the costs of administering a network of systems by easily and reliably reporting discrepancies in the files that are installed on deployed systems.

BART enables you to determine what file-level changes have occurred on a system, relative to a known baseline. You use BART to create a baseline or *control manifest* from a fully installed and configured system. You can then compare this baseline with a snapshot of the system at a later time, generating a report that lists file-level changes that have occurred on the system after it was installed.

BART Features

BART uses simple syntax that is both powerful and flexible. The tool enables you to track file changes on a given system over time. You can also track file differences between similar systems. Such comparisons can help you locate corrupted or unusual files, or systems whose software is out of date.

Additional benefits and uses of BART include the following:

- You can specify which files to monitor. For example, you can monitor local customizations, which can assist you in reconfiguring software easily and efficiently.
- You can troubleshoot system performance issues.

BART Components

BART creates two main files, a *manifest* and a comparison file, or *report*. An optional *rules file* enables you to customize the manifest and report.

BART Manifest

A *manifest* is a file-level snapshot of a system at a particular time. The manifest contains information about attributes of files, which can include some uniquely identifying information, such as a checksum. Options to the `bart create` command can target specific files and directories. A rules file can provide more fine-grained filtering, as described in "BART Rules File" on page 27.

Note - By default, BART catalogs all ZFS file systems under the root (`/`) directory. Other file system types, such as NFS or TMPFS file systems, and mounted CD-ROMs are cataloged.

You can create a manifest of a system immediately after an initial Oracle Solaris installation. You can also create a manifest after configuring a system to meet your site's security policy. This type of control manifest provides you with a baseline for later comparisons.

A baseline manifest can be used to track file integrity on the same system over time. It can also be used as a basis for comparison with other systems. For example, you could take a snapshot of other systems on your network and then compare those manifests with the baseline manifest. Reported file discrepancies indicate what you need to do to synchronize the other systems with the baseline system.

For the format of a manifest, see "BART Manifest File Format" on page 38. To create a manifest, use the `bart create` command, as described in "How to Create a Control Manifest" on page 28.

BART Report

A BART report lists per-file discrepancies between two manifests. A *discrepancy* is a change to any attribute for a given file that is cataloged for both manifests. Additions or deletions of file entries are also considered discrepancies.

For a useful comparison, the two manifests must target the same file systems. You must also create and compare the manifests with the same options and rules file.

For the format of a report, see “BART Reporting” on page 40. To create a report, use the `bart compare` command, as described in “How to Compare Manifests for the Same System Over Time” on page 31.

BART Rules File

A BART rules file is a file that you create to filter or target particular files and file attributes for inclusion or exclusion. You then use this file when creating BART manifests and reports. When you compare manifests, the rules file aids in flagging discrepancies between the manifests.

Note - When you create a manifest by using a rules file, you must use the same rules file to create the comparison manifest. You must also use the rules file when comparing the manifests. Otherwise, the report would list many invalid discrepancies.

Using a rules file to monitor specific files and file attributes on a system requires planning. Before you create a rules file, decide which files and file attributes to monitor on the system.

As a result of user error, a rules file can also contain syntax errors and other ambiguous information. If a rules file has errors, these errors are also reported.

For the format of a rules file, see “BART Rules File Format” on page 39 and the `bart_rules(4)` man page. To create a rules file, see “How to Customize a BART Report by Using a Rules File” on page 36.

About Using BART

The `bart` command is used to create and compare manifests. Any user can run this command. However, users can only catalog and monitor files that they have permission to access. So, users and most roles can usefully catalog the files in their home directory, but the `root` account can catalog all files, including system files.

BART Security Considerations

BART manifests and reports are readable by anyone. If BART output might contain sensitive information, take appropriate measures to protect the output. For example, use options that generate output files with restrictive permissions or place output files in a protected directory.

Using BART

Task	Description	For Instructions
Create a BART manifest.	Generates a list of information about every file that is installed on a system.	“How to Create a Control Manifest” on page 28
Create a custom BART manifest.	Generates a list of information about specific files that are installed on a system.	“How to Customize a Manifest” on page 30
Compare BART manifests.	Generates a report that compares changes to a system over time. Or, generates a report that compares one or several systems to a control system.	“How to Compare Manifests for the Same System Over Time” on page 31 “How to Compare Manifests From Different Systems” on page 33
(Optional) Customize a BART report.	Generates a custom BART report in one of the following ways: ■ By specifying attributes ■ By using a rules file	“How to Customize a BART Report by Specifying File Attributes” on page 36 “How to Customize a BART Report by Using a Rules File” on page 36

▼ How to Create a Control Manifest

This procedure explains how to create a baseline, or control, manifest for comparison. Use this type of manifest when you are installing many systems from a central image. Or, use this type of manifest to run comparisons when you want to verify that the installations are identical. For more information about control manifests, see “BART Manifest” on page 26. To understand the format conventions, see Example 2-1.

Note - Do not attempt to catalog networked file systems. Using BART to monitor networked file systems consumes large resources to generate manifests of little value.

Before You Begin You must assume the `root` role. For more information, see “Using Your Assigned Administrative Rights” in “Securing Users and Processes in Oracle Solaris 11.2 ”.

1. **After customizing your Oracle Solaris system to your site's security requirements, create a control manifest and redirect the output to a file.**

   ```
   # bart create options > control-manifest
   ```

`-R` Specifies the root directory for the manifest. All paths specified by the rules are interpreted relative to this directory. All paths reported in the manifest are relative to this directory.

`-I` Accepts a list of individual files to be cataloged, either on the command line or read from standard input.

`-r` Is the name of the rules file for this manifest. A `-` argument reads the rules file from standard input.

`-n` Turns off content signatures for all regular files in the file list. This option can be used to improve performance. Or, you can use this option if the contents of the file list are expected to change, as in the case of system log files.

2. Examine the contents of the manifest.

For an explanation of the format, see Example 2-1.

3. (Optional) Protect the manifest.

One way to protect system manifests is to place them in a directory that only the `root` account can access.

```
# mkdir /var/adm/log/bartlogs
# chmod 700 /var/adm/log/bartlogs
# mv control-manifest /var/adm/log/bartlogs
```

Choose a meaningful name for the manifest. For example, use the system name and date that the manifest was created, as in `mach1-120313`.

Example 2-1 Explanation of the BART Manifest Format

In this example, an explanation of the manifest format follows the sample output.

```
# bart create
! Version 1.1
! HASH SHA256
! Saturday, September 07, 2013 (22:22:27)
# Format:
#fname D size mode acl dirmtime uid gid
#fname P size mode acl mtime uid gid
#fname S size mode acl mtime uid gid
#fname F size mode acl mtime uid gid contents
#fname L size mode acl lnmtime uid gid dest
#fname B size mode acl mtime uid gid devnode
#fname C size mode acl mtime uid gid devnode
/ D 1024 40755 user::rwx,group::r-x,mask:r-x,other:r-x
3ebc418eb5be3729ffe7e54053be2d33ee884205502c81ae9689cd8cca5b0090 0 0
.
```

```
.
.
/zone D 512 40755 user::rwx group::r-x,mask:r-x,other:r-x 3f81e892
154de3e7bdfd0d57a074c9fae0896a9e2e04bebfe5e872d273b063319e57f334 0 0
.
.
.
```

Each manifest consists of a header and file entries. Each file entry is a single line, depending on the file type. For example, for each file entry in the preceding output, type `F` specifies a file and type `D` specifies a directory. Also listed is information about size, content, user ID, group ID, and permissions. File entries in the output are sorted by the encoded versions of the file names to correctly handle special characters. All entries are sorted in ascending order by file name. All nonstandard file names, such as those that contain embedded newline or tab characters, quote the nonstandard characters before sorting.

Lines that begin with `!` supply metadata about the manifest. The manifest version line indicates the manifest specification version. The hash line indicates the hash mechanism that was used. For more information about the `SHA256` hash that is used as a checksum, see the `sha2`(3EXT) man page.

The date line shows the date on which the manifest was created, in date form. See the `date`(1) man page. Some lines are ignored by the manifest comparison tool. Ignored lines include metadata, blank lines, lines that consist only of white space, and comments that begin with `#`.

▼ How to Customize a Manifest

You can customize a manifest in one of the following ways:

- By specifying a subtree

  Specifying an individual subtree is an efficient way to monitor changes to selected, important files, such as all files in the `/etc` directory.
- By specifying a file name

  Specifying a file name is an efficient way of monitoring particularly sensitive files, such as the files that configure and run a database application.
- By using a rules file

  By using a rules file to create and compare manifests gives you the flexibility to specify multiple attributes for more than one file or subtree. From the command line, you can specify a global attribute definition that applies to all files in a manifest or report. From a rules file, you can specify attributes that do not apply globally.

Before You Begin You must assume the `root` role. For more information, see “Using Your Assigned Administrative Rights” in “Securing Users and Processes in Oracle Solaris 11.2 ”.

1. **Determine which files to catalog and monitor.**
2. **Create a custom manifest by using one of the following options:**
   - By specifying a subtree:

     ```
     # bart create -R subtree
     ```
   - By specifying a file name or file names:

     ```
     # bart create -I filename...
     ```

     For example:

     ```
     # bart create -I /etc/system /etc/passwd /etc/shadow
     ```
   - By using a rules file:

     ```
     # bart create -r rules-file
     ```
3. **Examine the contents of the manifest.**
4. **(Optional) Save the manifest in a protected directory for future use.**

   For an example, see Step 3 in "How to Create a Control Manifest" on page 28.

Tip - If you used a rules file, save the rules file with the manifest. For a useful comparison, you must run the comparison with the rules file.

How to Compare Manifests for the Same System Over Time

By comparing manifests over time, you can locate corrupted or unusual files, detect security breaches, and troubleshoot performance issues on a system.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

1. **Create a control manifest of the files to monitor on the system.**

   ```
   # bart create -R /etc > control-manifest
   ```
2. **(Optional) Save the manifest in a protected directory for future use.**

   For an example, see Step 3 in "How to Create a Control Manifest" on page 28.
3. **At a later time, prepare an identical manifest to the control manifest.**

```
# bart create -R /etc > test-manifest
```

4. Protect the second manifest.

```
# mv test-manifest /var/adm/log/bartlogs
```

5. Compare the two manifests.

Use the same command-line options and rules file to compare the manifests that you used to create them.

```
# bart compare options control-manifest test-manifest > bart-report
```

6. Examine the BART report for oddities.

Example 2-2 Tracking File Changes for the Same System Over Time

This example shows how to track the changes in the `/etc` directory over time. This type of comparison enables you to locate important files on the system that have been compromised.

- Create a control manifest.

```
# cd /var/adm/logs/manifests
# bart create -R /etc > system1.control.090713
! Version 1.1
! HASH SHA256
! Saturday, September 07, 2013 (11:11:17)
# Format:
#fname D size mode acl dirmtime uid gid
#fname P size mode acl mtime uid gid
#fname S size mode acl mtime uid gid
#fname F size mode acl mtime uid gid contents
#fname L size mode acl lnmtime uid gid dest
#fname B size mode acl mtime uid gid devnode
#fname C size mode acl mtime uid gid devnode
/.cpr_config F 2236 100644 owner@:read_data/write_data/append_data/read_xattr/wr
ite_xattr/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchr
onize:allow,group@:read_data/read_xattr/read_attributes/read_acl/synchronize:all
ow,everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
4e271c59 0 0 3ebc418eb5be3729ffe7e54053be2d33ee884205502c81ae9689cd8cca5b0090
/.login F 1429 100644 owner@:read_data/write_data/append_data/read_xattr/write_x
attr/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize
:allow,group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow,ev
eryone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
4bf9d6d7 0 3 ff6251a473a53de68ce8b4036d0f569838cff107caf1dd9fd04701c48f09242e
.
.
```

```
.
```

- Later, create a test manifest by using the same command-line options.

```
# bart create -R /etc > system1.test.101013
Version 1.1
! HASH SHA256
! Monday, October 10, 2013 (10:10:17)
# Format:
#fname D size mode acl dirmtime uid gid
#fname P size mode acl mtime uid gid
#fname S size mode acl mtime uid gid
#fname F size mode acl mtime uid gid contents
#fname L size mode acl lnmtime uid gid dest
#fname B size mode acl mtime uid gid devnode
#fname C size mode acl mtime uid gid devnode
/.cpr_config F 2236 100644 owner@:read_data/write_data/append_data/read_xattr/wr
ite_xattr/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchr
onize:allow,group@:read_data/read_xattr/read_attributes/read_acl/synchronize:all
ow,everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
4e271c59 0 0 3ebc418eb5be3729ffe7e54053be2d33ee884205502c81ae9689cd8cca5b0090
.
.
.
```

- Compare the manifests.

```
# bart compare system1.control.090713 system1.test.101013
/security/audit_class
mtime  4f272f59
```

The output indicates that the modification time on the `audit_class` file has changed since the control manifest was created. If this change is unexpected, you can investigate further.

How to Compare Manifests From Different Systems

By comparing manifests from different systems, you can determine if the systems are installed identically or have been upgraded in synch. For example, if you customized your systems to a particular security target, this comparison finds any discrepancies between the manifest that represents your security target, and the manifests from the other systems.

Before You Begin You must assume the `root` role. For more information, see “Using Your Assigned Administrative Rights” in “Securing Users and Processes in Oracle Solaris 11.2 ”.

1. Create a control manifest.

```
# bart create options > control-manifest
```

For the options, see the bart(1M) man page.

2. **(Optional) Save the manifest in a protected directory for future use.**

   For an example, see Step 3 in "How to Create a Control Manifest" on page 28.

3. **On the test system, use the same `bart` options to create a manifest.**

   ```
   # bart create options > test1-manifest
   ```

4. **(Optional) Save the manifest in a protected directory for future use.**

5. **To perform the comparison, copy the manifests to a central location.**

   For example:

   ```
   # cp control-manifest /net/test-server/var/adm/logs/bartlogs
   ```

   If the test system is not an NFS-mounted system, use `sftp` or another reliable means to copy the manifests to a central location.

6. **Compare the manifests and redirect the output to a file.**

   ```
   # bart compare control-manifest test1-manifest > test1.report
   ```

7. **Examine the BART report for oddities.**

Example 2-3 Identifying a Suspect File in the `/usr/bin` Directory

This example compares the contents of the `/usr/bin` directory on two systems.

- Create a control manifest.

```
# bart create -R /usr/bin > control-manifest.090713
! Version 1.1
! HASH SHA256
! Saturday, September 07, 2013 (11:11:17)
# Format:
#fname D size mode acl dirmtime uid gid
#fname P size mode acl mtime uid gid
#fname S size mode acl mtime uid gid
#fname F size mode acl mtime uid gid contents
#fname L size mode acl lnmtime uid gid dest
#fname B size mode acl mtime uid gid devnode
#fname C size mode acl mtime uid gid devnode
/2to3 F 105 100555 owner@:read_data/read_xattr/write_xattr/execute/read_attribut
es/write_attributes/read_acl/write_acl/write_owner/synchronize:allow,group@:read
```

```
_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow,everyone@:re
ad_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow 4bf9d261 0
2 154de3e7bdfd0d57a074c9fae0896a9e2e04bebfe5e872d273b063319e57f334
/7z F 509220 100555 owner@:read_data/read_xattr/write_xattr/execute/read_attribu
tes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow,group@:rea
d_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow,everyone@:r
ead_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow 4dadc48a 0
2 3ecd418eb5be3729ffe7e54053be2d33ee884205502c81ae9689cd8cca5b0090
...
```

- Create an identical manifest for each system that you want to compare with the control system.

```
# bart create -R /usr/bin > system2-manifest.101013
! Version 1.1
! HASH SHA256
! Monday, October 10, 2013 (10:10:22)
# Format:
#fname D size mode acl dirmtime uid gid
#fname P size mode acl mtime uid gid
#fname S size mode acl mtime uid gid
#fname F size mode acl mtime uid gid contents
#fname L size mode acl lnmtime uid gid dest
#fname B size mode acl mtime uid gid devnode
#fname C size mode acl mtime uid gid devnode
/2to3 F 105 100555 owner@:read_data/read_xattr/write_xattr/execute/read_attribut
es/write_attributes/read_acl/write_acl/write_owner/synchronize:allow,group@:read
_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow,everyone@:re
ad_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow 4bf9d261 0
2 154de3e7bdfd0d57a074c9fae0896a9e2e04bebfe5e872d273b063319e57f334
...
```

- Copy the manifests to the same location.

```
# cp control-manifest.090713 /net/system2.central/bart/manifests
```

- Compare the manifests.

```
# bart compare control-manifest.090713 system2.test.101013 > system2.report
/su:
gid  control:3  test:1
/ypcat:
mtime  control:3fd72511  test:3fd9eb23
```

The output indicates that the group ID of the `su` file in the `/usr/bin` directory is not the same as that of the control system. This information might indicate that a different version of the software was installed on the test system. Because the GID is changed, the more likely reason is that someone has tampered with the file.

How to Customize a BART Report by Specifying File Attributes

This procedure is useful to filter the output from existing manifests for specific file attributes.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

1. **Determine which file attributes to check.**
2. **Compare two manifests that contain the file attributes to be checked.**

   For example:

   ```
   # bart compare -i lnmtime,mtime control-manifest.121513 \
     test-manifest.010514 > bart.report.010514
   ```

   Use a comma in the command-line syntax to separate each file attribute.
3. **Examine the BART report for oddities.**

How to Customize a BART Report by Using a Rules File

By using a rules file, you can customize a BART manifest for particular files and file attributes of interest. By using different rules files on default BART manifests, you can run different comparisons for the same manifests.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

1. **Determine which files and file attributes to monitor.**
2. **Create a rules file with the appropriate directives.**
3. **Create a control manifest with the rules file that you created.**

   ```
   # bart create -r myrules1-file > control-manifest
   ```
4. **(Optional) Save the manifest in a protected directory for future use.**

   For an example, see Step 3 in "How to Create a Control Manifest" on page 28.
5. **Create an identical manifest on a different system, at a later time, or both.**

```
# bart create -r myrules1-file > test-manifest
```

6. **Compare the manifests by using the same rules file.**

```
# bart compare -r myrules1-file control-manifest test-manifest > bart.report
```

7. **Examine the BART report for oddities.**

Example 2-4 Using a Rules File to Customize BART Manifests and the Comparison Report

The following rules file directs the `bart create` command to list all attributes of the files in the `/usr/bin` directory. In addition, the rules file directs the `bart compare` command to report only size and content changes in the same directory.

```
# Check size and content changes in the /usr/bin directory.
# This rules file only checks size and content changes.
# See rules file example.

IGNORE all
CHECK size contents
/usr/bin
```

- Create a control manifest with the rules file that you created.

```
# bart create -r usrbinrules.txt > usr_bin.control-manifest.121013
```

- Prepare an identical manifest whenever you want to monitor changes to the `/usr/bin` directory.

```
# bart create -r usrbinrules.txt > usr_bin.test-manifest.121113
```

- Compare the manifests by using the same rules file.

```
# bart compare -r usrbinrules.txt usr_bin.control-manifest.121013 \
usr_bin.test-manifest.121113
```

- Examine the output of the `bart compare` command.

```
 /usr/bin/gunzip:  add
/usr/bin/ypcat:
delete
```

The preceding output indicates that the `/usr/bin/ypcat` file was deleted, and the `/usr/bin/gunzip` file was added.

BART Manifests, Rules Files, and Reports

This section describes the format of files that BART uses and creates.

BART Manifest File Format

Each manifest file entry is a single line, depending on the file type. Each entry begins with *fname*, which is the name of the file. To prevent parsing problems from special characters embedded in file names, the file names are encoded. For more information, see "BART Rules File Format" on page 39.

Subsequent fields represent the following file attributes:

type
: Type of file with the following possible values:
- B for a block device node
- C for a character device node
- D for a directory
- F for a file
- L for a symbolic link
- P for a pipe
- S for a socket

size
: File size in bytes.

mode
: Octal number that represents the permissions of the file.

acl
: ACL attributes for the file. For a file with ACL attributes, this contains the output from `acltotext`.

uid
: Numerical user ID of the owner of this entry.

gid
: Numerical group ID of the owner of this entry.

dirmtime
: Last modification time, in seconds, since 00:00:00 UTC, January 1, 1970, for directories.

lnmtime
: Last modification time, in seconds, since 00:00:00 UTC, January 1, 1970, for links.

mtime
: Last modification time, in seconds, since 00:00:00 UTC January 1, 1970, for files.

contents
: Checksum value of the file. This attribute is only specified for regular files. If you turn off context checking, or if checksums cannot be computed, the value of this field is `-`.

dest
: Destination of a symbolic link.

devnode Value of the device node. This attribute is for character device files and block device files only.

For more information, see the `bart_manifest(4)` man page.

BART Rules File Format

Rules files are text files that consist of lines that specify which files are to be included in the manifest and which file attributes are to be included in the manifest or the report. Lines that begin with **#**, blank lines, and lines that contain white space are ignored by the tool.

The input files have three types of directives:

- Subtree directive, with optional pattern matching modifiers
- `CHECK` directive
- `IGNORE` directive

EXAMPLE 2-5 Rules File Format

```
<Global CHECK/IGNORE Directives>
<subtree1> [pattern1..]
<IGNORE/CHECK Directives for subtree1>

<subtree2> [pattern2..]
<subtree3> [pattern3..]
<subtree4> [pattern4..]
<IGNORE/CHECK Directives for subtree2, subtree3, subtree4>
```

Note - All directives are read in order. Later directives can override earlier directives.

A subtree directive *must* begin with an absolute pathname, followed by zero or more pattern matching statements.

BART Rules File Attributes

The `CHECK` and `IGNORE` statements define which file attributes to track or ignore. The metadata that begins each manifest lists the attribute *keywords* per file type. See Example 2-1.

The `all` keyword indicates all file attributes.

BART Quoting Syntax

The rules file specification language that BART uses is the standard UNIX quoting syntax for representing nonstandard file names. Embedded tab, space, newline, or special characters are encoded in their octal forms to enable the tool to read file names. This nonuniform quoting syntax prevents certain file names, such as those containing an embedded carriage return, from being processed correctly in a command pipeline. The rules specification language allows the expression of complex file name filtering criteria that would be difficult and inefficient to describe by using shell syntax alone.

For more information, see the `bart_rules`(4) man page.

BART Reporting

In default mode, a BART report checks all the files installed on the system, with the exception of modified directory timestamps (`dirmtime`):

```
CHECK all
IGNORE dirmtime
```

If you supply a rules file, then the global directives of `CHECK all` and `IGNORE dirmtime`, in that order, are automatically prepended to the rules file.

BART Output

The following exit values are returned:

0	Success
1	Nonfatal error when processing files, such as permission problems
>1	Fatal error, such as an invalid command-line option

The reporting mechanism provides two types of output: verbose and programmatic:

- Verbose output is the default output and is localized and presented on multiple lines. Verbose output is internationalized and is human-readable. When the `bart compare` command compares two system manifests, a list of file differences is generated.

  The structure of the output is as follows:

  filename attribute control:control-val test:test-val

filename	Name of the file that differs between the control manifest and the test manifest.
attribute	Name of the file attribute that differs between the manifests that are compared. The *control-val* precedes the *test-val*. When discrepancies for multiple attributes occur in the same file, each difference is noted on a separate line.

Following is an example of attribute differences for the `/etc/passwd` file. The output indicates that the `size`, `mtime`, and `contents` attributes have changed.

```
/etc/passwd:
size control:74 test:81
mtime control:3c165879 test:3c165979
contents control:daca28ae0de97afd7a6b91fde8d57afa
test:84b2b32c4165887355317207b48a6ec7
```

- Programmatic output is generated with the `-p` option to the `bart compare` command. This output is suitable for programmatic manipulation.

  The structure of the output is as follows:

  filename attribute control-val test-val [*attribute control-val test-val*]*

filename	Same as the *filename* attribute in the default format
attribute control-val test-val	A description of the file attributes that differ between the control and test manifests for each file

For a list of attributes that are supported by the `bart` command, see "BART Rules File Attributes" on page 39.

For more information, see the `bart`(1M) man page.

Security Glossary

Access Control List (ACL) An access control list (ACL) provides finer-grained file security than traditional UNIX file protection provides. For example, an ACL enables you to allow group read access to a file, while allowing only one member of that group to write to the file.

admin principal A user principal with a name of the form *username*`/admin` (as in `jdoe/admin`). An admin principal can have more privileges (for example, to change policies) than a regular user principal. See also principal name, user principal.

AES Advanced Encryption Standard. A symmetric 128-bit block data encryption technique. The U.S. government adopted the Rijndael variant of the algorithm as its encryption standard in October 2000. AES replaces user principal encryption as the government standard.

algorithm A cryptographic algorithm. This is an established, recursive computational procedure that encrypts or hashes input.

application server See network application server.

asynchronous audit event Asynchronous events are the minority of system events. These events are not associated with any process, so no process is available to be blocked and later woken up. Initial system boot and PROM enter and exit events are examples of asynchronous events.

audit files Binary audit logs. Audit files are stored separately in an audit file system.

audit policy The global and per-user settings that determine which audit events are recorded. The global settings that apply to the audit service typically affect which pieces of optional information are included in the audit trail. Two settings, `cnt` and `ahlt`, affect the operation of the system when the audit queue fills. For example, audit policy might require that a sequence number be part of every audit record.

audit trail The collection of all audit files from all hosts.

authenticated rights profile A rights profile that requires the assigned user or role to type a password before executing an operation from the profile. This behavior is similar to `sudo` behavior. The length of time that the password is valid is configurable.

authentication The process of verifying the claimed identity of a principal.

authenticator Authenticators are passed by clients when requesting tickets (from a KDC) and services (from a server). They contain information that is generated by using a session key known only by the client and server, that can be verified as of recent origin, thus indicating that the transaction is secure. When used with a ticket, an authenticator can be used to authenticate a user principal. An authenticator includes the principal name of the user, the IP address of the user's host, and a time stamp. Unlike a ticket, an authenticator can be used only once, usually when access to a service is requested. An authenticator is encrypted by using the session key for that client and that server.

authorization 1. In Kerberos, the process of determining if a principal can use a service, which objects the principal is allowed to access, and the type of access that is allowed for each object.

2. In user rights management, a right that can be assigned to a role or user (or embedded in a rights profile) for performing a class of operations that are otherwise prohibited by security policy. Authorizations are enforced at the user application level, not in the kernel.

basic set The set of privileges that are assigned to a user's process at login. On an unmodified system, each user's initial inheritable set equals the basic set at login.

Blowfish A symmetric block cipher algorithm that takes a variable-length key from 32 bits to 448 bits. Its author, Bruce Schneier, claims that Blowfish is optimized for applications where the key does not change often.

client Narrowly, a process that makes use of a network service on behalf of a user; for example, an application that uses `rlogin`. In some cases, a server can itself be a client of some other server or service.

More broadly, a host that a) receives a Kerberos credential, and b) makes use of a service that is provided by a server.

Informally, a principal that makes use of a service.

client principal (RPCSEC_GSS API) A client (a user or an application) that uses RPCSEC_GSS-secured network services. Client principal names are stored in the form of `rpc_gss_principal_t` structures.

clock skew The maximum amount of time that the internal system clocks on all hosts that are participating in the Kerberos authentication system can differ. If the clock skew is exceeded between any of the participating hosts, requests are rejected. Clock skew can be specified in the `krb5.conf` file.

confidentiality See privacy.

consumer In the Cryptographic Framework feature of Oracle Solaris, a consumer is a user of the cryptographic services that come from providers. Consumers can be applications, end users, or kernel operations. Kerberos, IKE, and IPsec are examples of consumers. For examples of providers, see provider.

credential An information package that includes a ticket and a matching session key. Used to authenticate the identity of a principal. See also ticket, session key.

credential cache A storage space (usually a file) that contains credentials that are received from the KDC.

cryptographic algorithm See algorithm.

DES Data Encryption Standard. A symmetric-key encryption method developed in 1975 and standardized by ANSI in 1981 as ANSI X.3.92. DES uses a 56-bit key.

device allocation Device protection at the user level. Device allocation enforces the exclusive use of a device by one user at a time. Device data is purged before device reuse. Authorizations can be used to limit who is permitted to allocate a device.

device policy Device protection at the kernel level. Device policy is implemented as two sets of privileges on a device. One set of privileges controls read access to the device. The second set of privileges controls write access to the device. See also policy.

Diffie-Hellman protocol Also known as public key cryptography. An asymmetric cryptographic key agreement protocol that was developed by Diffie and Hellman in 1976. The protocol enables two users to exchange a secret key over an insecure medium without any prior secrets. Diffie-Hellman is used by Kerberos.

digest See message digest.

DSA Digital Signature Algorithm. A public key algorithm with a variable key size from 512 to 4096 bits. The U.S. Government standard, DSS, goes up to 1024 bits. DSA relies on SHA1 for input.

ECDSA Elliptic Curve Digital Signature Algorithm. A public key algorithm that is based on elliptic curve mathematics. An ECDSA key size is significantly smaller than the size of a DSA public key needed to generate a signature of the same length.

effective set The set of privileges that are currently in effect on a process.

flavor Historically, *security flavor* and *authentication flavor* had the same meaning, as a flavor that indicated a type of authentication (AUTH_UNIX, AUTH_DES, AUTH_KERB). RPCSEC_GSS is also a security flavor, even though it provides integrity and privacy services in addition to authentication.

forwardable ticket A ticket that a client can use to request a ticket on a remote host without requiring the client to go through the full authentication process on that host. For example, if the user `david` obtains a forwardable ticket while on user `jennifer`'s machine, `david` can log in to his own machine without being required to get a new ticket (and thus authenticate himself again). See also proxiable ticket.

FQDN Fully qualified domain name. For example, `central.example.com` (as opposed to simply `denver`).

GSS-API The Generic Security Service Application Programming Interface. A network layer that provides support for various modular security services, including the Kerberos service.

GSS-API provides for security authentication, integrity, and privacy services. See also authentication, integrity, privacy.

hardening The modification of the default configuration of the operating system to remove security vulnerabilities that are inherent in the host.

hardware provider In the Cryptographic Framework feature of Oracle Solaris, a device driver and its hardware accelerator. Hardware providers offload expensive cryptographic operations from the computer system, thus freeing CPU resources for other uses. See also provider.

host A system that is accessible over a network.

host principal A particular instance of a service principal in which the principal (signified by the primary name `host`) is set up to provide a range of network services, such as `ftp`, `rcp`, or `rlogin`. An example of a host principal is `host/central.example.com@EXAMPLE.COM`. See also server principal.

inheritable set The set of privileges that a process can inherit across a call to `exec`.

initial ticket A ticket that is issued directly (that is, not based on an existing ticket-granting ticket). Some services, such as applications that change passwords, might require tickets to be marked `initial` so as to assure themselves that the client can demonstrate a knowledge of its secret key. This assurance is important because an initial ticket indicates that the client has recently authenticated itself (instead of relying on a ticket-granting ticket, which might existed for a long time).

instance The second part of a principal name, an instance qualifies the principal's primary. In the case of a service principal, the instance is required. The instance is the host's fully qualified domain name, as in `host/central.example.com`. For user principals, an instance is optional. Note, however, that `jdoe` and `jdoe/admin` are unique principals. See also primary, principal name, service principal, user principal.

integrity A security service that, in addition to user authentication, provides for the validity of transmitted data through cryptographic checksumming. See also authentication, privacy.

invalid ticket A postdated ticket that has not yet become usable. An invalid ticket is rejected by an application server until it becomes validated. To be validated, an invalid ticket must be presented to the KDC by the client in a TGS request, with the `VALIDATE` flag set, after its start time has passed. See also postdated ticket.

KDC Key Distribution Center. A machine that has three Kerberos V5 components:

- Principal and key database
- Authentication service
- Ticket-granting service

Each realm has a master KDC and should have one or more slave KDCs.

Kerberos An authentication service, the protocol that is used by that service, or the code that is used to implement that service.

The Kerberos implementation in Oracle Solaris that is closely based on Kerberos V5 implementation.

While technically different, “Kerberos” and “Kerberos V5” are often used interchangeably in the Kerberos documentation.

Kerberos (also spelled Cerberus) was a fierce, three-headed mastiff who guarded the gates of Hades in Greek mythology.

Kerberos policy A set of rules that governs password usage in the Kerberos service. Policies can regulate principals' accesses, or ticket parameters, such as lifetime.

key 1. Generally, one of two main types of keys:

- A *symmetric key* – An encryption key that is identical to the decryption key. Symmetric keys are used to encrypt files.
- An *asymmetric key* or *public key* – A key that is used in public key algorithms, such as Diffie-Hellman or RSA. Public keys include a private key that is known only by one user, a public key that is used by the server or general resource, and a private-public key pair that combines the two. A private key is also called a *secret* key. The public key is also called a *shared* key or *common* key.

2. An entry (principal name) in a keytab file. See also keytab file.

3. In Kerberos, an encryption key, of which there are three types:

- A *private key* – An encryption key that is shared by a principal and the KDC, and distributed outside the bounds of the system. See also private key.
- A *service key* – This key serves the same purpose as the private key, but is used by servers and services. See also service key.
- A *session key* – A temporary encryption key that is used between two principals, with a lifetime limited to the duration of a single login session. See also session key.

keystore A keystore holds passwords, passphrases, certificates, and other authentication objects for retrieval by applications. A keystore can be specific to a technology, or a location that several applications use.

keytab file A key table file that contains one or more keys (principals). A host or service uses a keytab file in the much the same way that a user uses a password.

kvno Key version number. A sequence number that tracks a particular key in order of generation. The highest kvno is the latest and most current key.

least privilege A security model which gives a specified process only a subset of superuser powers. The least privilege model assigns enough privilege to regular users that they can perform personal administrative tasks, such as mount file systems and change the ownership of files. On the

other hand, processes run with just those privileges that they need to complete the task, rather than with the full power of superuser, that is, all privileges. Damage due to programming errors like buffer overflows can be contained to a non-root user, which has no access to critical abilities like reading or writing protected system files or halting the machine.

limit set The outside limit of what privileges are available to a process and its children.

MAC 1. See message authentication code (MAC).

2. Also called labeling. In government security terminology, MAC is Mandatory Access Control. Labels such as Top Secret and Confidential are examples of MAC. MAC contrasts with DAC, which is Discretionary Access Control. UNIX permissions are an example of DAC.

3. In hardware, the unique system address on a LAN. If the system is on an Ethernet, the MAC is the Ethernet address.

master KDC The main KDC in each realm, which includes a Kerberos administration server, `kadmind`, and an authentication and ticket-granting daemon, `krb5kdc`. Each realm must have at least one master KDC, and can have many duplicate, or slave, KDCs that provide authentication services to clients.

MD5 An iterative cryptographic hash function that is used for message authentication, including digital signatures. The function was developed in 1991 by Rivest. Its use is deprecated.

mechanism 1. A software package that specifies cryptographic techniques to achieve data authentication or confidentiality. Examples: Kerberos V5, Diffie-Hellman public key.

2. In the Cryptographic Framework feature of Oracle Solaris, an implementation of an algorithm for a particular purpose. For example, a DES mechanism that is applied to authentication, such as CKM_DES_MAC, is a separate mechanism from a DES mechanism that is applied to encryption, CKM_DES_CBC_PAD.

message authentication code (MAC) MAC provides assurance of data integrity and authenticates data origin. MAC does not protect against eavesdropping.

message digest A message digest is a hash value that is computed from a message. The hash value almost uniquely identifies the message. A digest is useful for verifying the integrity of a file.

minimization The installation of the minimal operating system that is necessary to run the server. Any software that does not directly relate to the operation of the server is either not installed, or deleted after the installation.

name service scope The scope in which a role is permitted to operate, that is, an individual host or all hosts that are served by a specified naming service such as NIS LDAP.

network application server A server that provides a network application, such as `ftp`. A realm can contain several network application servers.

network policies The settings that network utilities configure to protect network traffic. For information about network security, see "Securing the Network in Oracle Solaris 11.2 ".

nonattributable audit event An audit event whose initiator cannot be determined, such as the AUE_BOOT event.

NTP Network Time Protocol. Software from the University of Delaware that enables you to manage precise time or network clock synchronization, or both, in a network environment. You can use NTP to maintain clock skew in a Kerberos environment. See also clock skew.

PAM Pluggable Authentication Module. A framework that allows for multiple authentication mechanisms to be used without having to recompile the services that use them. PAM enables Kerberos session initialization at login.

passphrase A phrase that is used to verify that a private key was created by the passphrase user. A good passphrase is 10-30 characters long, mixes alphabetic and numeric characters, and avoids simple prose and simple names. You are prompted for the passphrase to authenticate use of the private key to encrypt and decrypt communications.

password policy The encryption algorithms that can be used to generate passwords. Can also refer to more general issues around passwords, such as how often the passwords must be changed, how many password attempts are permitted, and other security considerations. Security policy requires passwords. Password policy might require passwords to be encrypted with the AES algorithm, and might make further requirements related to password strength.

permitted set The set of privileges that are available for use by a process.

policy Generally, a plan or course of action that influences or determines decisions and actions. For computer systems, policy typically means security policy. Your site's security policy is the set of rules that define the sensitivity of the information that is being processed and the measures that are used to protect the information from unauthorized access. For example, security policy might require that systems be audited, that devices must be allocated for use, and that passwords be changed every six weeks.

For the implementation of policy in specific areas of the Oracle Solaris OS, see audit policy, policy in the Cryptographic Framework, device policy, Kerberos policy, password policy, and rights policy.

policy for public key technologies In the Key Management Framework (KMF), policy is the management of certificate usage. The KMF policy database can put constraints on the use of the keys and certificates that are managed by the KMF library.

policy in the Cryptographic Framework In the Cryptographic Framework feature of Oracle Solaris, policy is the disabling of existing cryptographic mechanisms. The mechanisms then cannot be used. Policy in the Cryptographic Framework might prevent the use of a particular mechanism, such as `CKM_DES_CBC`, from a provider, such as DES.

postdated ticket A postdated ticket does not become valid until some specified time after its creation. Such a ticket is useful, for example, for batch jobs that are intended to run late at night, since the

ticket, if stolen, cannot be used until the batch job is run. When a postdated ticket is issued, it is issued as `invalid` and remains that way until a) its start time has passed, and b) the client requests validation by the KDC. A postdated ticket is normally valid until the expiration time of the ticket-granting ticket. However, if the postdated ticket is marked `renewable`, its lifetime is normally set to be equal to the duration of the full life time of the ticket-granting ticket. See also invalid ticket, renewable ticket.

primary The first part of a principal name. See also instance, principal name, realm.

principal 1. A uniquely named client/user or server/service instance that participates in a network communication. Kerberos transactions involve interactions between principals (service principals and user principals) or between principals and KDCs. In other words, a principal is a unique entity to which Kerberos can assign tickets. See also principal name, service principal, user principal.

2. (RPCSEC_GSS API) See client principal, server principal.

principal name 1. The name of a principal, in the format *primary/instance@REALM*. See also instance, primary, realm.

2. (RPCSEC_GSS API) See client principal, server principal.

principle of least privilege See least privilege.

privacy A security service, in which transmitted data is encrypted before being sent. Privacy also includes data integrity and user authentication. See also authentication, integrity, service.

private key A key that is given to each user principal, and known only to the user of the principal and to the KDC. For user principals, the key is based on the user's password. See also key.

private-key encryption In private-key encryption, the sender and receiver use the same key for encryption. See also public-key encryption.

privilege 1. In general, a power or capability to perform an operation on a computer system that is beyond the powers of a regular user. Superuser privileges are all the rights that superuser is granted. A privileged user or privileged application is a user or application that has been granted additional rights.

2. A discrete right on a process in an Oracle Solaris system. Privileges offer a finer-grained control of processes than does `root`. Privileges are defined and enforced in the kernel. Privileges are also called *process privileges* or *kernel privileges*. For a full description of privileges, see the `privileges`(5) man page.

privilege escalation Gaining access to resources that are outside the range of resources that your assigned rights, including rights that override the defaults, permit. The result is that a process can perform unauthorized operations.

privilege model A stricter model of security on a computer system than the superuser model. In the privilege model, processes require privilege to run. Administration of the system can be divided into discrete parts that are based on the privileges that administrators have in their processes. Privileges can be assigned to an administrator's login process. Or, privileges can be assigned to be in effect for certain commands only.

privilege set A collection of privileges. Every process has four sets of privileges that determine whether a process can use a particular privilege. See limit set, effective set set, permitted set set, and inheritable set set.

Also, the basic set set of privileges is the collection of privileges that are assigned to a user's process at login.

privilege-aware Programs, scripts, and commands that turn on and off the use of privilege in their code. In a production environment, the privileges that are turned on must be supplied to the process, for example, by requiring users of the program to use a rights profile that adds the privileges to the program. For a full description of privileges, see the `privileges`(5) man page.

privileged application An application that can override system controls. The application checks for security attributes, such as specific UIDs, GIDs, authorizations, or privileges.

privileged user A user who is assigned rights beyond the rights of regular user on a computer system. See also trusted users.

profile shell In rights management, a shell that enables a role (or user) to run from the command line any privileged applications that are assigned to the role's rights profiles. The profile shell versions correspond to the available shells on the system, such as the `pfbash` version of `bash`.

provider In the Cryptographic Framework feature of Oracle Solaris, a cryptographic service that is provided to consumers. PKCS #11 libraries, kernel cryptographic modules, and hardware accelerators are examples of providers. Providers plug in to the Cryptographic Framework, so are also called *plugins*. For examples of consumers, see consumer.

proxiable ticket A ticket that can be used by a service on behalf of a client to perform an operation for the client. Thus, the service is said to act as the client's proxy. With the ticket, the service can take on the identity of the client. The service can use a proxiable ticket to obtain a service ticket to another service, but it cannot obtain a ticket-granting ticket. The difference between a proxiable ticket and a forwardable ticket is that a proxiable ticket is only valid for a single operation. See also forwardable ticket.

public object A file that is owned by the `root` user and readable by the world, such as any file in the `/etc` directory.

public-key encryption An encryption scheme in which each user has two keys, one public key and one private key. In public-key encryption, the sender uses the receiver's public key to encrypt the message, and the receiver uses a private key to decrypt it. The Kerberos service is a private-key system. See also private-key encryption.

QOP Quality of Protection. A parameter that is used to select the cryptographic algorithms that are used in conjunction with the integrity service or privacy service.

RBAC Role-based access control, the user rights management feature of Oracle Solaris. See rights.

RBAC policy See rights policy.

realm 1. The logical network that is served by a single Kerberos database and a set of Key Distribution Centers (KDCs).

2. The third part of a principal name. For the principal name `jdoe/admin@CORP.EXAMPLE.COM`, the realm is `CORP.EXAMPLE.COM`. See also principal name.

reauthentication The requirement to provide a password to perform a computer operation. Typically, `sudo` operations require reauthentication. Authenticated rights profiles can contain commands that require reauthentication. See authenticated rights profile.

relation A configuration variable or relationship that is defined in the `kdc.conf` or `krb5.conf` files.

renewable ticket Because having tickets with very long lives is a security risk, tickets can be designated as `renewable`. A renewable ticket has two expiration times: a) the time at which the current instance of the ticket expires, and b) maximum lifetime for any ticket. If a client wants to continue to use a ticket, the client renews the ticket before the first expiration occurs. For example, a ticket can be valid for one hour, with all tickets having a maximum lifetime of ten hours. If the client that holds the ticket wants to keep it for more than an hour, the client must renew the ticket. When a ticket reaches the maximum ticket lifetime, it automatically expires and cannot be renewed.

rights An alternative to the all-or-nothing superuser model. User rights management and process rights management enable an organization to divide up superuser's privileges and assign them to users or roles. Rights in Oracle Solaris are implemented as kernel privileges, authorizations, and the ability to run a process as a specific UID or GID. Rights can be collected in a rights profile and a role.

rights policy The security policy that is associated with a command. Currently, `solaris` is the valid policy for Oracle Solaris. The `solaris` policy recognizes privileges and extended privilege policy, authorizations, and `setuid` security attributes.

rights profile Also referred to as a profile. A collection of security overrides that can be assigned to a role or user. A rights profile can include authorizations, privileges, commands with security attributes, and other rights profiles that are called supplementary profiles.

role A special identity for running privileged applications that only assigned users can assume.

RSA A method for obtaining digital signatures and public key cryptosystems. The method was first described in 1978 by its developers, Rivest, Shamir, and Adleman.

scan engine A third-party application, residing on an external host, that examines a file for known viruses.

SEAM The product name for the initial version of Kerberos on Solaris systems. This product is based on the Kerberos V5 technology that was developed at the Massachusetts Institute of Technology. SEAM is now called the Kerberos service. It continues to differ slightly from the MIT version.

secret key See private key.

Secure Shell A special protocol for secure remote login and other secure network services over an insecure network.

security attributes Overrides to security policy that enable an administrative command to succeed when the command is run by a user other than superuser. In the superuser model, the `setuid root` and `setgid` programs are security attributes. When these attributes are applied to a command, the command succeeds no matter who runs the command. In the privilege model, kernel privileges and other rights replace `setuid root` programs as security attributes. The privilege model is compatible with the superuser model, in that the privilege model also recognizes the `setuid` and `setgid` programs as security attributes.

security flavor See flavor.

security mechanism See mechanism.

security policy See policy.

security service See service.

seed A numeric starter for generating random numbers. When the starter originates from a random source, the seed is called a *random seed*.

separation of duty Part of the notion of least privilege. Separation of duty prevents one user from performing or approving all operations that complete a transaction. For example, in RBAC, you can separate the creation of a login user from the assignment of security overrides. One role creates the user. A separate role can assign security attributes, such as rights profiles, roles, and privileges to existing users.

server A principal that provides a resource to network clients. For example, if you `ssh` to the system `central.example.com`, then that system is the server that provides the `ssh` service. See also service principal.

server principal (RPCSEC_GSS API) A principal that provides a service. The server principal is stored as an ASCII string in the form *service*@*host*. See also client principal.

service 1. A resource that is provided to network clients, often by more than one server. For example, if you `rlogin` to the machine `central.example.com`, then that machine is the server that provides the `rlogin` service.

	2. A security service (either integrity or privacy) that provides a level of protection beyond authentication. See also integrity and privacy.
service key	An encryption key that is shared by a service principal and the KDC, and is distributed outside the bounds of the system. See also key.
service principal	A principal that provides Kerberos authentication for a service or services. For service principals, the primary name is a name of a service, such as `ftp`, and its instance is the fully qualified host name of the system that provides the service. See also host principal, user principal.
session key	A key that is generated by the authentication service or the ticket-granting service. A session key is generated to provide secure transactions between a client and a service. The lifetime of a session key is limited to a single login session. See also key.
SHA1	Secure Hashing Algorithm. The algorithm operates on any input length less than 2^{64} to produce a message digest. The SHA1 algorithm is input to DSA.
single-system image	A single-system image is used in Oracle Solaris auditing to describe a group of audited systems that use the same naming service. These systems send their audit records to a central audit server, where the records can be compared as if the records came from one system.
slave KDC	A copy of a master KDC, which is capable of performing most functions of the master. Each realm usually has several slave KDCs (and only one master KDC). See also KDC, master KDC.
software provider	In the Cryptographic Framework feature of Oracle Solaris, a kernel software module or a PKCS #11 library that provides cryptographic services. See also provider.
stash file	A stash file contains an encrypted copy of the master key for the KDC. This master key is used when a server is rebooted to automatically authenticate the KDC before it starts the `kadmind` and `krb5kdc` processes. Because the stash file includes the master key, the stash file and any backups of it should be kept secure. If the encryption is compromised, then the key could be used to access or modify the KDC database.
superuser model	The typical UNIX model of security on a computer system. In the superuser model, an administrator has all-or-nothing control of the system. Typically, to administer the machine, a user becomes superuser (`root`) and can do all administrative activities.
synchronous audit event	The majority of audit events. These events are associated with a process in the system. A non-attributable event that is associated with a process is a synchronous event, such as a failed login.
TGS	Ticket-Granting Service. That portion of the KDC that is responsible for issuing tickets.
TGT	Ticket-Granting Ticket. A ticket that is issued by the KDC that enables a client to request tickets for other services.

ticket An information packet that is used to securely pass the identity of a user to a server or service. A ticket is valid for only a single client and a particular service on a specific server. A ticket contains the principal name of the service, the principal name of the user, the IP address of the user's host, a time stamp, and a value that defines the lifetime of the ticket. A ticket is created with a random session key to be used by the client and the service. Once a ticket has been created, it can be reused until the ticket expires. A ticket only serves to authenticate a client when it is presented along with a fresh authenticator. See also authenticator, credential, service, session key.

ticket file See credential cache.

trusted users Users whom you have decided can perform administrative tasks at some level of trust. Typically, administrators create logins for trusted users first and assign administrative rights that match the users' level of trust and ability. These users then help configure and maintain the system. Also called *privileged users*.

user principal A principal that is attributed to a particular user. A user principal's primary name is a user name, and its optional instance is a name that is used to described the intended use of the corresponding credentials (for example, `jdoe` or `jdoe/admin`). Also known as a user instance. See also service principal.

virtual private network (VPN) A network that provides secure communication by using encryption and tunneling to connect users over a public network.

Index

G

I

K

L

M

N

O

P

Q

R

S

T

U

V

W