Securing Files and Verifying File Integrity in Oracle® Solaris 11.3

Part No: E54827
October 2017

Securing Files and Verifying File Integrity in Oracle Solaris 11.3

Part No: E54827

Access to Oracle Support

Oracle customers that have purchased support 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.

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` si vous êtes malentendant.

Contents

Using This Documentation

- **Overview** – Describes how to protect legitimate files, view hidden file permissions, and locate suspicious 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

Documentation and resources for this product and related products are available at `http://www.oracle.com/pls/topic/lookup?ctx=E53394-01`.

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.

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 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)`
`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 2, "Enabling Users to Change the Ownership of Their Own Files," on page 20.

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 2 File and Directory Permissions

Symbol	Permission	Object	Description
`r`	Read	File	Designated users can open and read the contents of a file.

Symbol	Permission	Object	Description
r	Read	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.
w	Write	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.
x	Execute	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 24.

`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 24.

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 create a swap file or set up a public directory on a TMPFS file system. For instructions, see Example 5, “Setting Special File Permissions in Absolute Mode,” on page 24.

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 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 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 5 Setting File Permissions in Symbolic Mode

Symbol	Function	Description
`u`	*who*	User (owner)
`g`	*who*	Group
`o`	*who*	Others
`a`	*who*	All

Symbol	Function	Description
=	*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 5, "Setting Special File Permissions in Absolute Mode," on page 24. The following table lists the octal values for setting special permissions on a file.

TABLE 6 Setting Special File Permissions in Absolute Mode

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

Using File Attributes to Add Security to ZFS Files

In a ZFS file system, you can mark security-relevant files for special treatment. The file attributes can affect local files, NFS-mounted files, or CIFS-mounted files. The `chmod(1)` and `ls(1)` man pages describe how to set and list file attributes.

File attributes that have security implications include the following:

- `appendonly` attribute – Permits adding to the end of a file but prevents modifying existing contents. This attribute on a log file can prevent changes to log file entries. Requires the `PRIV_FILE_FLAG_SET` privilege on the process to set the attribute and all privileges to remove it.
- `immutable` attribute – Prevents modifying or deleting the contents of a file. Also prevents changing file metadata except for access time updates. On a directory, this attribute prevents the deletion of the directory and its files. Requires the `PRIV_FILE_FLAG_SET` privilege on the process to set the attribute and all privileges to remove it.

  For an example, see "Making a ZFS File Immutable" in *Securing Files and Verifying File Integrity in Oracle Solaris 11.3*.
- `nounlink` attribute – Prevents deletion of critical files or directories. On a directory, this attribute prevents the deletion or renaming of files. This attribute can prevent the accidental deletion of files that are critical for an application. Requires the `PRIV_FILE_FLAG_SET` privilege on the process to set the attribute and all privileges to remove it.
- `sensitive` attribute – Indicates that the file contains keying information, such as PINs or passwords. Sensitive files are not written to the audit record.
- `readonly` attribute – Permits no content change to a CIFS-mounted file. The owner of the file can set or clear this attribute, or a user or group with the `write_attributes` permission can set or clear this attribute.

For more information, see "Examples of Setting Security-Relevant Attributes on ZFS Files" on page 46.

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 2, "Using ACLs and Attributes to Protect Oracle Solaris ZFS Files".

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. Most programs can function correctly without using executable stacks.

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.

For how to prevent stacks from being used by malicious programs, see "Protecting the Process Heap and Executable Stacks From Compromise" in *Securing Systems and Attached Devices in Oracle Solaris 11.3*.

To prevent system compromise by executables in a mounted filesystem, you can use the `nosetuid` and `noexec` arguments to the `mount` command. For more information, see the `mount(1M)` man page.

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 18
Change local file ownership.	"How to Change the Owner of a File" on page 19

Task	For Instructions
	"How to Change Group Ownership of a File" on page 20
Change local file permissions.	"How to Change File Permissions in Symbolic Mode" on page 21 "How to Change File Permissions in Absolute Mode" on page 22 "How to Change Special File Permissions in Absolute Mode" on page 23

▼ 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 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 10.
- Permissions – For example, `r-xr-xr-x`. For description, see "File and Directory Ownership" on page 10.
- 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 11.3*.

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 11.3*.

Example 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. 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 11.3*.

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 11.3*.

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 2, "Enabling Users to Change the Ownership of Their Own Files," on page 20.

▼ 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 5, "Setting File Permissions in Symbolic Mode," on page 14.
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 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 4, "Setting File Permissions in Absolute Mode," on page 14.

   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 2, "Using ACLs and Attributes to Protect Oracle Solaris ZFS Files" in *Securing Files and Verifying File Integrity in Oracle Solaris 11.3*.

   ---

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 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 11.3*.

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 6, "Setting Special File Permissions in Absolute Mode," on page 15.

   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 `ls -v` command to make sure that the appropriate permissions are set for all ACL entries. For more information, see the `ls(1)` man page.

   ---

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

   `% ls -l` *filename*

Example 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 procedures find risky executables on the system and prevent programs from exploiting process heaps and executable stacks.

- "How to Find Files With Special File Permissions" on page 24 locates files with the `setuid` bit set, but that are not owned by the `root` user.
- "Protecting the Process Heap and Executable Stacks From Compromise" in *Securing Systems and Attached Devices in Oracle Solaris 11.3* prevents programs from malicious software attacks.

▼ 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 11.3*.

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) man page.

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

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

For background information, see "`setuid` Permission" on page 12.

Example 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    32432  Jul 14 14:14  /usr/bin/atq
-rwsr-xr-x  1 root   sys    32664  Jul 14 14:14  /usr/bin/atrm
-rwsr-xr-x  1 root   bin    82836  Jul 14 14:14  /usr/bin/cdrw
-r-sr-xr-x  1 root   sys    41448  Jul 14 14:14  /usr/bin/chkey
-r-sr-xr-x  1 root   bin     7968  Jul 14 14:14  /usr/bin/mailq
-r-sr-sr-x  1 root   sys    45364  Jul 14 14:14  /usr/bin/passwd
-rwsr-xr-x  1 root   bin    37740  Jul 14 14:14  /usr/bin/pfedit
-r-sr-xr-x  1 root   bin    51472  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   213092  Jul 14 14:14  /usr/bin/sudo
-r-sr-xr-x  4 root   bin    24056  Jul 14 14:14  /usr/bin/uptime
-r-sr-xr-x  1 root   bin    79540  Jul 14 14:14  /usr/bin/xlock
# mv /var/tmp/ckprm /var/share/sysreports/ckprm
```

♦ ♦ ♦ CHAPTER 2

Using ACLs and Attributes to Protect Oracle Solaris ZFS Files

This chapter provides information about using access control lists (ACLs) to protect your ZFS files by providing more granular permissions than the standard UNIX permissions.

This chapter covers the following topics:

Oracle Solaris ACL Model

The Oracle Solaris ACL model fully supports the interoperability that NFSv4 offers between UNIX and non-UNIX clients. ZFS ACLs are similar to Windows NT-style ACLs, and provide more fine-grained access control than standard file permissions provide. ACLs are set and displayed with the `chmod` and `ls` commands.

The ACL model has two types of Access Control Entries (ACEs) that affect access checking: `ALLOW` and `DENY`. Therefore, you cannot infer from any single ACE that defines a set of permissions whether the permissions that are not defined in that ACE are allowed or denied.

For a description of the model, see the *NFSv4 ACLs* section of the `acl(5)` man page. For information about backup products, see "Saving ZFS Data With Other Backup Products" in *Managing ZFS File Systems in Oracle Solaris 11.3*.

Rights to Modify ZFS ACLs

You can assign and modify ACLs of the files and directories that you own. For files that others own, you must get permission in one of the following ways to assign and modify those ACLs:

- The owner of the file or directory gave you the `write_acl` permission:

  ```
  $ chmod A+user:jdoe:write_acl:f:allow file.1
  ```
- You are assigned the Object Access Management rights profile. If you are not assigned a profile shell as your default shell, run the `pfbash` or `pfexec` command before running the command that changes the ACL.
- You are assigned the `root` role.

ACL Formats

ACLs have two basic formats:

- **Trivial ACL** – Contains only entries for traditional UNIX user categories that are represented as `owner@`, `group@`, and `everyone@`.

  For a newly created file, the default ACL grants `owner@` all permissions, and prevents `group@` and `everyone@` from modifying the file and its attributes:

  ```
  0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
  /read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
  1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
  2:everyone@:/** same as group@ **/
  ```

  For a newly created directory, the default ACL grants `owner@` all permissions, and prevents `group@` and `everyone@` from modifying the directory and its attributes:

  ```
  0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
  /append_data/read_xattr/write_xattr/execute/delete_child
  /read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
  1:group@:list_directory/read_data/read_xattr/execute/read_attributes/read_acl
  /synchronize:allow
  2:everyone@:/** same as group@ **/
  ```
- **Non-Trivial ACL** – Contains entries for added user categories. The entries might also include inheritance flags, or are ordered in a non-traditional way.

  A non-trivial entry is similar to the following example, which grants user `jan` specific permissions.

```
0:user:jan:read_data/write_data:file_inherit:allow
```

For a description of ACL access privileges, see the *NFSv4 ACLs* section of the `acl(5)` man page.

ACL Entry Descriptions

The following entry illustrates the elements that comprise an ACL entry. These elements apply to both trivial and non-trivial ACLs.

```
0:user:jan:read_data/write_data:file_inherit:allow
```

Index
: A number at the beginning of the entry, such as the number zero (`0`) in the example. The index identifies a specific entry and distinguishes the entry from others in the ACL.

ACL entry type
: The user category. In trivial ACLs, only entries for `owner@`, `group@`, and `everyone@` are set. Non-trivial ACL entry types are `user:`*username* and `group:`*groupname*. In the example, the entry type is `user:jan`.

Access privileges
: Permissions that are granted or denied to the entry type. In the example, the user is granted `read_data` and `write_data`.

Inheritance flags
: An optional list of ACL flags that control how permissions are propagated in a directory structure. In the sample entry, `file_inherit` is also granted to the user.

Permission control
: Determines whether the permissions in an entry are allowed or denied. In the example, the permissions are allowed.

The following table describes each ACL entry type.

TABLE 7 ACL Entry Types

ACL Entry Type	Format	Description
`owner@`	Trivial	Specifies the access granted to the owner of the object.
`group@`	Trivial	Specifies the access granted to the owning group of the object.
`everyone@`	Trivial	Specifies the access granted to any user or group that does not match any other ACL entry.
`user`	Non-trivial	With a user name, specifies the access granted to an additional user of the object. Must include the ACL entry ID, which can be a user name or a user ID.

ACL Entry Type	Format	Description
group	Non-trivial	With a group name, specifies the access granted to an additional group of the object. Must include the ACL entry ID, which can be a group name or a group ID.

ZFS ACL Sets

ZFS ACL sets are predefined combinations of ACL permissions. You cannot extend the sets.

- `full_set` – All permissions
- `modify_set` – All permissions except `write_acl` and `write_owner`
- `read_set` – `read_data`, `read_attributes`, `read_xattr`, and `read_acl`
- `write_set` – `write_data`, `append_data`, `write_attributes`, and `write_xattr`

You can apply an ACL set rather than separately setting individual permissions.

EXAMPLE 7 Applying an ACL Set to a File

With the addition of the `read_set` ACL set, the user `jan` can read file contents and the file's basic and extended attributes, and retain the default permissions.

```
$ pfexec chmod A+user:jan:read_set:allow file.1
$ ls -v file.1
-r--r--r--+  1 root     root      206695 Jul 20 13:43 file.1
0:user:jan:read_data/read_xattr/read_attributes/read_acl:allow
...
```

ACL Properties

The ZFS file system includes two properties that affect ACLs:

- `aclmode` – Modifies ACL behavior when a file is initially created or controls how an ACL is modified during a `chmod` operation.

  By default, ACL entries are discarded. Other possible modes are a `mask` that reduces user or group permissions, and a `passthrough` that leaves the ACLs in effect.

  For more information about `aclmode` values, see the `aclmode` entry in the `zfs`(1M) man page and Example 14, "Showing the Effects of the `aclmode` and `aclinherit` Properties on ACL Permissions," on page 38.

- `aclinherit` – Determines the behavior of ACL inheritance and ACL interaction with `chmod` operations.

  By default, `write_owner` and `write_acl` permissions are removed when an ACL entry is inherited. Other possible behaviors are to `discard` all ACL entries, only inherit `deny` entries, and leave the ACLs in effect with `passthrough`.

  For more information about `aclinherit` values, see "Effect of ACL Inherit Mode on ACL Inheritance" on page 42 and the `aclinherit` entry in the `zfs`(1M) man page.

ACL Inheritance Flags

ACL inheritance enables a newly created file or directory to inherit the ACLs that it should inherit without disregarding the existing permission bits on the parent directory. By default, ACLs are not inherited. A non-trivial ACL on a directory is not inherited to any subsequent directory. You must specify the inheritance of an ACL on a file or directory.

ZFS and NFSv4 provide the following inheritance flags. The letters in parentheses are the compact format of the flag:

- `file_inherit (f)` – Inherit from parent directory.
- `dir_inherit (d)` – Inherit from parent directory.
- `inherit_only (i)` – Newly created files or subdirectories inherit from the parent directory.
- `no_propagate (n)` – Inherit only to the first level directory.
- `failed_access (F)` – Alarm or audit record created at failed access.
- `successful_access (S)` – Alarm or audit record created at successful access.
- `-` – No permissions.
- `inherited (I)` – Indicator of inheritance.

For a full description of the optional inheritance flags, see the *NFSv4 ACLs* section of the `acl`(5) man page.

In addition, you can change the default ACL inheritance policy on a file system by using the `aclinherit` file system property. For more information about this property, see "ACL Properties" on page 30 and "Setting ACL Inheritance on ZFS Files" on page 39.

Note - Currently, the `successful_access`, `failed_access`, and `inherited` flags apply only to the SMB server.

Setting ACLs on ZFS Files

The primary rules of ACL access on a ZFS file are as follows:

- ZFS processes ACL entries in the order they are listed in the ACL, from the top down.
- Only ACL entries where the specified user matches the requester of the access are processed.
- Once an allow permission has been granted, it cannot be denied by a subsequent ACL deny entry in the same ACL permission set.
- The owner of the file is granted the `write_acl` permission unconditionally even if the permission is explicitly denied. Otherwise, any permission left unspecified is denied.

  In the cases of deny permissions or when an access permission is missing, the `PRIV_FILE*` privileges determine access. The privileges mechanism prevents file owners from getting locked out of their files and enables superuser to modify files for recovery purposes. For more information, see the `privileges(5)` man page.

Command Syntax for Setting and Modifying ACLs

To set or modify ACLs, use the `chmod` command. The command syntax resembles the syntax for setting permission bits on files, except that you type **A** before typing the operator (`+`, `=`, or `-`).

- `chmod` command syntax for trivial ACLs

  ```
  chmod [options] A[index]{+|=}owner@ |group@ |everyone@: \
    access-permissions/...[:inheritance-flags]:deny | allow file
  ```

  ```
  chmod [options] A-owner@, group@, everyone@: \
    access-permissions/...[:inheritance-flags]:deny | allow file ...
  ```

  ```
  chmod [options] A[index]- file
  ```

- `chmod` command syntax for non-trivial ACLs

  ```
  chmod [options] A[index]{+|=}user|group:name: \
  access-permissions/...[:inheritance-flags]:deny | allow file
  ```

  ```
  chmod [options] A-user|group:name: \
  access-permissions/...[:inheritance-flags]:deny | allow file ...
  ```

  ```
  chmod [options] A[index]- file
  ```

The `chmod` command uses the following operators for ACLs:

- `A+` adds an ACL entry. `An+` adds the ACL for the specified index number.

For example, `chmod A+` adds an ACL entry, while `chmod A3+` adds an ACL entry to index number 3.

- `A=` replaces the ACL. `A`*n*`=` replaces the ACL of the specified index number.

  For example, `chmod A=` replaces an entire ACL, while `chmod A3=` replaces only the existing ACL entry of index number 3.
- `A-` removes an ACL entry. Use this command syntax to restore a trivial ACL to the file. After you issue the command, only the entries for `owner@`, `group@`, and `everyone@` that comprise a trivial ACL remain.

  `A`*n*`-` removes the ACL from the specified index number. For example, `chmod A3-` removes the existing ACL entry from index number 3.

Permissions and inheritance flags are represented by unique letters listed in the *NFSv4 ACLs* section of the `acl(5)`. When you set ZFS ACLs, you can either use the letters that correspond to those permissions (compact mode) or type the permissions in full (verbose mode).

EXAMPLE 8 Setting ACLs on Files and Directories

The following examples illustrate the use of the `chmod` command to set ACLs on a file.

The following two commands are equivalent. The first command uses the compact mode of the permission. Each command grants read and execute permissions to user Tamiko on `file.1`.

```
$ chmod A+user:tamiko:rx:allow file.1
$ chmod A+user:tamiko:read_data/execute:allow file.1
```

Similarly, the following command grants user Tamiko inheritable read, write, and execute permissions for the newly created `dir.2` and its files.

```
$ chmod A+user:tamiko:rwx:fd:allow dir.2
```

The verbose mode of the permission grants the same access.

```
$ chmod A+user:tamiko:list_directory/write_data/execute:file_inherit/dir_inherit:allow
 dir.2
```

The use of the `A+` operator enables `group@` to write data to `file.1` and does not affect existing ACL entries.

```
$ chmod A+group@:w:allow file.1
```

EXAMPLE 9 Replacing ACLs

The use of the `A=` operator removes existing ACL entries for `file.1` and replaces them with the single entry for `everyone@`. This entry removes the remaining default permissions (`read_xattr/read_attributes/read_acl/synchronize:allow`) for `everyone@`.

```
$ chmod A=everyone@:rx:allow file.1
```

EXAMPLE 10 Removing ACLs

The following examples illustrate the use of the `chmod` command to remove ACLs from a file.

This example removes all non-trivial ACL entries for a file without listing each entry to be removed.

```
$ chmod A- file.1
```

In the following example, the owner grants `read_data/write_data` permissions to `group@`. This command removes the other default permissions, `read_xattr/read_attributes/read_acl/synchronize:allow`.

```
$ chmod A1=group@:read_data/write_data:allow file.1
$ ls -v file.1
-rw-rw-r--   1 root     root      206695 Jul 20 13:43 file.1
0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:read_data/write_data:allow
2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow
```

EXAMPLE 11 Removing an Added ACL Entry by Index Number

In the following example, `read_data/execute` permissions are added for the user Alice on the `test.dir` directory. Alice's entry is index number 0.

```
$ chmod A0+user:alice:read_data/execute:allow test.dir
$ ls -dv test.dir
drwxr-xr-x+  2 root     root           2 Jul 20 14:23 test.dir
0:user:alice:list_directory/read_data/execute:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

```
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

In the following example, access permissions are removed for user Alice by using the index number of her ACL entry.

```
$ chmod A0- test.dir
$ ls -dv test.dir
drwxr-xr-x   2 root     root           2 Jul 20 14:23 test.dir
0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
2:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

```
$ chmod A3=everyone@:list_directory/read_data/read_xattr/execute/read_attributes \
/read_acl/synchronize:allow:failed_access:audit dir1

$ ls -v
total 1
drwxr-xr-x 2 foo staff 2 Feb 1 19:28 dir1
     0:everyone@:list_directory/read_data/read_attributes/read_acl:failed_access:audit
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
     3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

Displaying ACL Information

With the `ls` command, you can display ACL information in one of two formats. The `-v` option displays the permissions in full or verbose form. The `-V` option generates compact output by using letters that represent the permissions and flags.

EXAMPLE 12 Displaying ACLs in Compact and Verbose Format

The following example shows how the same ACL information is displayed in verbose format and compact format:

```
$ ls -v file.1
-rw-r--r--   1 root     root      206695 Jul 20 14:27 file.1
0:owner@:read_data/write_data/append_data/read_attributes
/write_xattr/read_xattr/write_attributes/read_acl/write_acl
/write_owner/synchronize:allow
1:group@:read_data/read_attributes/read_xattr/read_acl
/synchronize:allow
2:everyone@:read_data/append_data/read_xattr/read_acl
/synchronize:allow

$ ls -V file.1
-rw-r--r--   1 root     root      206695 Jul 20 14:27 file.1
owner@:rw-p--aARWcCos:-------:allow
group@:r-----a-R-c--s:-------:allow
everyone@:r-----a-R-c--s:-------:allow
```

For a description of ACL access privileges, see the *NFSv4 ACLs* section of the `acl`(5) man page.

ACL Interaction With Permission Bits

In ZFS files, the UNIX permission bits correspond to the ACL entries, but are cached. When you change a file's permission bits, the file's ACL is updated accordingly. Likewise, modifying a file's ACL causes changes in the permission bits.

For more information about permission bits, see `chmod`(1).

The following examples show the relationship between a file or directory's ACLs and the permission bits and how permission changes in one affect the other.

EXAMPLE 13 Showing How ACLs and Permission Bits Interact

The first example begins with the following ACL for `file.2`. The permission bits, 644, display as `-rw-r--r--`.

```
$ ls -v file.2
-rw-r--r--   1 root     root        2693 Jul 20 14:26 file.2
0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

The chmod command removes the ACL entry for everyone@. Accordingly, the read permission for everyone is also removed, so the permission bits change to 640, which display as -rw-r-----.

```
$ chmod A2- file.2
$ ls -v file.2
-rw-r-----   1 root     root        2693 Jul 20 14:26 file.2
0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

Next, the ACL is replaced with just read_data/write_data permissions for everyone@. Because no owner@ or group@ ACL entry exists to override the permissions for owner and group, the permission bits become 666, which display as -rw-rw-rw-.

```
$ chmod A=everyone@:rw:allow file.2
$ ls -v file.2
-rw-rw-rw-   1 root     root        2440 Jul 20 14:28 file.3     Permission bits become 666.
0:everyone@:read_data/write_data:allow
```

If you replace this ACL with read permissions just for user Alice, the file will become inaccessible because no trivial ACL entries exist. Consequently, the permission bits are set to 000, which denies Alice access to file.2, as well as denies access to everyone else.

```
$ chmod A=user:alice:r:allow file.2
$ ls -v file.2
----------+  1 root     root        2440 Jul 20 14:28 file.3
0:user:alice:read_data:allow
```

If you set the permission bits for an inaccessible file, the default trivial ACL permissions are reset. The following command sets the bits for file.2 to 655. Automatically, the default trivial ACL permissions are set. Because the permission bits are set to 655, the owner is denied execute access.

```
$ pfexec chmod 655 file.2
$ ls -v file.3
-rw-r-xr-x   1 root     root        2440 Jul 20 14:28 file.3
0:user:alice:read_data:allow
1:owner@:execute:deny
2:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
3:group@:read_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow
4:everyone@:read_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow
```

EXAMPLE 14 Showing the Effects of the `aclmode` and `aclinherit` Properties on ACL Permissions

The following examples illustrate how specific `aclmode` and `aclinherit` property values affect ACL behavior. If these properties are set, ACL permissions for a file or directory are either reduced or expanded to be consistent with the owning group.

In this example, the administrator who runs the `zfs set` commands must be assigned the ZFS File System Management rights profile. To run the `chown` command, the administrator is assigned the Object Access Management rights profile.

Suppose that the `aclmode` property is set to `mask` and the `aclinherit` property is set to `restricted` in the pool, and that the original file and group ownership and ACL permissions are as follows:

```
$ pfbash ;  zfs set aclmode=mask system1/data
$ zfs set aclinherit=restricted system1/data

$ ls -lV file.1
-rwxrwx---+  1 root     root      206695 Aug 30 16:03 file.1
user:amy:r-----a-R-c---:-------:allow
user:rory:r-----a-R-c---:-------:allow
group:sysadmin:rw-p--aARWc---:-------:allow
group:staff:rw-p--aARWc---:-------:allow
owner@:rwxp--aARWcCos:-------:allow
group@:rwxp--aARWc--s:-------:allow
everyone@:------a-R-c--s:-------:allow
```

To understand the meaning of the values set for the two properties, see "ACL Properties" on page 30.

A `chown` operation changes the ownership of `file.1` to Amy and the group Staff.

```
$ chown amy:staff file.1
```

Amy then changes the permission bits for `file.1` to 640. Because the ACL properties were previously set, the permissions for the groups in the ACL are reduced so that they do not exceed the permissions of the owning Staff.

```
$ su - amy
$ chmod 640 file.1
$ ls -lV file.1
-rw-r-----+  1 amy      staff     206695 Aug 30 16:03 file.1
user:amy:r-----a-R-c---:-------:allow
user:rory:r-----a-R-c---:-------:allow
group:sysadmin:r-----a-R-c---:-------:allow
group:staff:r-----a-R-c---:-------:allow
owner@:rw-p--aARWcCos:-------:allow
```

```
group@:r-----a-R-c--s:-------:allow
everyone@:------a-R-c--s:-------:allow
```

Amy then changes the permission bits to 770. Consequently, the permissions of the groups in the ACL are also changed to match the permission of the owning group Staff.

```
$ chmod 770 file.1
$ ls -lV file.1
-rwxrwx---+  1 amy      staff     206695 Aug 30 16:03 file.1
user:amy:r-----a-R-c---:-------:allow
user:rory:r-----a-R-c---:-------:allow
group:sysadmin:rw-p--aARWc---:-------:allow
group:staff:rw-p--aARWc---:-------:allow
owner@:rwxp--aARWcCos:-------:allow
group@:rwxp--aARWc--s:-------:allow
everyone@:------a-R-c--s:-------:allow
```

Setting ACL Inheritance on ZFS Files

You can determine how ACLs are inherited on files and directories.

The aclinherit property can be set globally on a file system. By default, aclinherit is set to restricted.

For more information, see "ACL Inheritance Flags" on page 31.

Enabling the ACL on a Directory to Be Inherited

This section identifies the file ACEs that are applied when the file_inherit flag is set.

In the following example, an administrator who is assigned the Object Access Management rights profile adds read_data/write_data permissions and enables them to be inherited for user alice in the test2.dir directory.

```
$ pfbash ; chmod A+user:alice:read_data/write_data:file_inherit:allow test2.dir
$ ls -dv test2.dir
drwxr-xr-x+  2 root     root           2 Jul 20 14:55 test2.dir
0:user:alice:read_data/write_data:file_inherit:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes/read_acl/
synchronize:allow
```

```
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes/read_acl/
synchronize:allow
```

In the following example, user `alice`'s permissions are applied on the newly created `test2.dir/file.2` file.

```
$ touch test2.dir/file.2
$ ls -v test2.dir/file.2
-rw-r--r--+  1 root     root           0 Jul 20 14:56 test2.dir/file.2
0:user:alice:read_data:inherited:allow
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

- Because she is granted `read_data:file_inherit:allow`, she can read the contents of any newly created file.
- Because the group permission on the `file.2` does not include `write_data` permission, `alice` also does not have this permission. The `aclinherit` property for this file system in default mode, `restricted`, which prevents the user in the ACL from having more permissions than the group permissions.

The `inherit_only` permission is applied when the `file_inherit` or `dir_inherit` flags are set. `inherit_only` propagates the ACL through the directory structure. As such, user `alice` is granted or denied permission from `everyone@` permissions only if she is the file owner or is a member of the file's group owner. For example:

```
$ mkdir test2.dir/subdir.2
$ ls -dv test2.dir/subdir.2
drwxr-xr-x+  2 root     root           2 Jul 20 14:57 test2.dir/subdir.2
0:user:alice:list_directory/read_data/add_file/write_data:file_inherit
/inherit_only/inherited:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes/read_acl/
synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

Effect of `file_inherit` and `dir_inherit` Flags

This section provides examples that identify the file and directory ACLs that are applied when both the `file_inherit` and `dir_inherit` flags are set. The examples also show the interaction between ACLs and permission bits when the default `aclinherit` property, `restricted`, is in effect.

EXAMPLE 15 Setting and Viewing Inheritable ACLs

In this example, user `alice` is granted read, write, and execute permissions that are inherited for newly created files and directories.

```
$ pfexec chmod A+user:alice:read_data/write_data/execute:file_inherit/dir_inherit:allow
test3.dir
$ ls -dv test3.dir
drwxr-xr-x+  2 root     root           2 Jul 20 15:00 test3.dir
0:user:alice:list_directory/read_data/add_file/write_data/execute
:file_inherit/dir_inherit:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

The `inherited` text in the output for index number 0 is informational.

```
$ touch test3.dir/file.3
$ ls -v test3.dir/file.3
-rw-r--r--+  1 root     root           0 Jul 20 15:01 test3.dir/file.3
0:user:alice:read_data:inherited:allow
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

EXAMPLE 16 Viewing Effect of `aclinherit` `restricted` on `file_inherit:allow` ACLs

In these examples, because the permission bits of the parent directory for `group@` and `everyone@` deny write and execute permissions, user `alice` is denied write and execute permissions despite the `chmod` command explicitly assigning her these permissions. The default `aclinherit` property is `restricted`, which prevents `write_data` and `execute` from being inherited.

In this example, user `alice` is granted read, write, and execute permissions that are inherited for newly created files, but are not propagated to subsequent contents of the directory.

```
$ pfexec chmod A+user:alice:read_data/write_data/execute:file_inherit/no_propagate:allow
test4.dir
```

```
$ ls -dv test4.dir
drwxr--r--+  2 root     root           2 Mar  1 12:11 test4.dir
0:user:alice:list_directory/read_data/add_file/write_data/execute
:file_inherit/no_propagate:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:list_directory/read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/read_attributes/read_acl/
synchronize:allow
```

As a result of the default ACL inheritance value, `restricted`, the `write_data` and `execute` permissions are removed for `alice` in `file.4` because her permissions cannot be greater than the group's permissions for files in that directory.

```
$ touch test4.dir/file.4
$ ls -v test4.dir/file.4
-rw-r--r--+  1 root     root           0 Jul 20 15:09 test4.dir/file.4
0:user:alice:read_data:inherited:allow
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

Effect of ACL Inherit Mode on ACL Inheritance

This section describes the `aclinherit` property values.

EXAMPLE 17 ACL Viewing the Effect of `discard` on ACL Inheritance

If the `aclinherit` property on a file system is set to `discard`, then ACLs can potentially be discarded when the permission bits on a directory change. For example:

```
$ pfbash ; zfs set aclinherit=discard system1/cindy
$ chmod A+user:alice:read_data/write_data/execute:dir_inherit:allow test5.dir
$ ls -dv test5.dir
drwxr-xr-x+  2 root     root           2 Jul 20 14:18 test5.dir
0:user:alice:list_directory/read_data/add_file/write_data/execute:dir_inherit:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes/read_acl/
synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes/read_acl/
synchronize:allow
```

If, at a later time, you decide to tighten the permission bits on a directory, the non-trivial ACL is discarded. For example:

```
$ pfexec chmod 744 test5.dir
$ ls -dv test5.dir
drwxr--r--   2 root     root           2 Jul 20 14:18 test5.dir
0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
1:group@:list_directory/read_data/read_xattr/read_attributes/read_acl/synchronize:allow
2:everyone@:list_directory/read_data/read_xattr/read_attributes/read_acl/
synchronize:allow
```

EXAMPLE 18 Viewing the Effect of `noallow` on ACL Inheritance

In the following example, two non-trivial ACLs with file inheritance are set. One ACL allows `read_data` permission, and one ACL denies `read_data` permission. This example also illustrates how you can specify two ACEs in the same `chmod` command.

```
$ pfbash ; zfs set aclinherit=noallow system1/jdoe
$ chmod A+user:alice:read_data:file_inherit:deny,user:lp:read_data:file_inherit:allow
test6.dir
$ ls -dv test6.dir
drwxr-xr-x+  2 root     root           2 Jul 20 14:22 test6.dir
0:user:alice:read_data:file_inherit:deny
1:user:lp:read_data:file_inherit:allow
2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
3:group@:list_directory/read_data/read_xattr/execute/read_attributes/read_acl/
synchronize:allow
4:everyone@:list_directory/read_data/read_xattr/execute/read_attributes/read_acl/
synchronize:allow
```

When a new file is created, the ACL that allows `read_data` permission is discarded.

```
$ touch test6.dir/file.6
$ ls -v test6.dir/file.6
-rw-r--r--+  1 root     root           0 Jul 20 14:23 test6.dir/file.6
0:user:alice:read_data:inherited:deny
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

ACL passthrough Inherit Mode

A file system that has the aclinherit property set to passthrough inherits all inheritable ACL entries without any modifications made to the ACL entries when they are inherited. Files are created with a permission mode that is determined by the inheritable ACEs. If no inheritable ACEs exist that affect the permission mode, then the permission mode is set in accordance to the requested mode from the application.

EXAMPLE 19 ACL Inheritance With ACL Inherit Mode Set to passthrough in Verbose Mode

If the aclinherit property on the system1/cindy file system is set to passthrough, then user alice would inherit the ACL applied on test4.dir for the newly created file.5 as follows:

```
$ pfexec zfs set aclinherit=passthrough system1/cindy
$ touch test4.dir/file.5
$ ls -v test4.dir/file.5
-rw-r--r--+  1 root     root           0 Jul 20 14:16 test4.dir/file.5
0:user:alice:read_data/write_data/execute:inherited:allow
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

EXAMPLE 20 ACL Inheritance With ACL Inherit Mode Set to passthrough in Compact Mode

The following examples use compact ACL syntax to show how to inherit permission bits by setting aclinherit mode to passthrough.

In this example, an ACL is set on test1.dir to force inheritance. The syntax creates an owner@, group@, and everyone@ ACL entry for newly created files. Newly created directories inherit an @owner, group@, and everyone@ ACL entry.

```
$ pfbash ; zfs set aclinherit=passthrough system1/cindy
$ pwd
/system1/cindy
$ mkdir test1.dir

$ chmod A=owner@:rwxpcCosRrWaAdD:fd:allow,group@:rwxp:fd:allow,
everyone@::fd:allow test1.dir
$ ls -Vd test1.dir
drwxrwx---+  2 root     root           2 Jul 20 14:42 test1.dir
owner@:rwxpdDaARWcCos:fd-----:allow
group@:rwxp----------:fd-----:allow
everyone@:--------------:fd-----:allow
```

In this example, a newly created file inherits the ACL that was specified to be inherited to newly created files.

```
$ cd test1.dir
$ touch file.1
$ ls -V file.1
-rwxrwx---+  1 root     root           0 Jul 20 14:44 file.1
owner@:rwxpdDaARWcCos:------I:allow
group@:rwxp----------:------I:allow
everyone@:--------------:------I:allow
```

In this example, a newly created directory inherits both ACEs that control access to this directory as well as ACEs for future propagation to children of the newly created directory.

```
$ mkdir subdir.1
$ ls -dV subdir.1
drwxrwx---+  2 root     root           2 Jul 20 14:45 subdir.1
owner@:rwxpdDaARWcCos:fd----I:allow
group@:rwxp----------:fd----I:allow
everyone@:--------------:fd----I:allow
```

The `fd----I` entries are for propagating inheritance and are not considered during access control.

In the following example, a file is created with a trivial ACL in another directory where inherited ACEs are not present.

```
$ cd /system1/cindy
$ mkdir test2.dir
$ cd test2.dir
$ touch file.2
$ ls -V file.2
-rw-r--r--   1 root     root           0 Jul 20 14:48 file.2
owner@:rw-p--aARWcCos:-------:allow
group@:r-----a-R-c--s:-------:allow
everyone@:r-----a-R-c--s:-------:allow
```

ACL Inherit `passthrough-x` Mode

When `aclinherit=passthrough-x` is enabled, files are created with the execute (`x`) permission for `owner@`, `group@`, or `everyone@`, but only if execute permission is set in the file creation mode and in an inheritable ACE that affects the mode.

The following example shows how to inherit the execute permission by setting the `aclinherit` mode to `passthrough-x`.

```
$ pfexec zfs set aclinherit=passthrough-x system1/cindy
```

The following ACL is set on `/system1/cindy/test1.dir` to provide executable ACL inheritance for files for `owner@`.

```
$ pfexec chmod A=owner@:rwxpcCosRrWaAdD:fd:allow,group@:rwxp:fd:allow,
everyone@::fd:allow test1.dir
$ ls -Vd test1.dir
drwxrwx---+  2 root     root           2 Jul 20 14:50 test1.dir
owner@:rwxpdDaARWcCos:fd-----:allow
group@:rwxp----------:fd-----:allow
everyone@:--------------:fd-----:allow
```

A file (`file1`) is created with requested permissions `0666`. The resulting permissions are `0660`. The execution permission was not inherited because the creation mode did not request it.

```
$ touch test1.dir/file1
$ ls -V test1.dir/file1
-rw-rw----+  1 root     root           0 Jul 20 14:52 test1.dir/file1
owner@:rw-pdDaARWcCos:------I:allow
group@:rw-p----------:------I:allow
everyone@:--------------:------I:allow
```

Next, an executable called `t` is generated by using the `cc` compiler in the `testdir` directory.

```
$ cc -o t t.c
$ ls -V t
-rwxrwx---+  1 root     root        7396 Dec  3 15:19 t
owner@:rwxpdDaARWcCos:------I:allow
group@:rwxp----------:------I:allow
everyone@:--------------:------I:allow
```

The resulting permissions are `0770` because `cc` requested permissions `0777`, which caused the execute permission to be inherited from the `owner@`, `group@`, and `everyone@` entries.

Examples of Setting Security-Relevant Attributes on ZFS Files

This section shows how to add security-relevant attributes to ZFS files and how to display them.

For more information, review the following:

- "Using File Attributes to Add Security to ZFS Files" on page 15
- `ls(1)` and `chmod(1)` man pages

Note - If you are working in a non-global zone, you cannot set the `immutable`, `nounlink`, or `appendonly` attributes by default. You must add the privilege `file_flag_set` to the zone to enable setting these attributes. See "How to Modify Zone Privileges" in *Creating and Using Oracle Solaris Zones*.

EXAMPLE 21 Making a ZFS File Immutable

A user who is assigned the Object Access Management rights profile makes a file immutable by running the following command:

```
$ chmod S+ci file.1
$ echo this >>file.1
-bash: file.1: Not owner
$ rm file.1
rm: cannot remove `file.1': Not owner
```

To display the permissions, the user runs the `ls -l/c` command:

```
$ ls -l/c file.1
-rw-r--r--+  1 root     root      206695 Jul 20 14:27 file.1
{A-----im----}
```

To make the file accessible again, the user runs the following command:

```
$ chmod S-ci file.1
$ ls -l/c file.1
-rw-r--r--+  1 root     root      206695 Jul 20 14:27 file.1
{A------m----}
```

EXAMPLE 22 Making a ZFS File Read-Only

The following example shows how to apply read-only access to a ZFS file.

```
$ chmod S+cR file.2
$ echo this >>file.2
-bash: file.2: Not owner
```

Preventing Accidental Deletions With the `nounlink` Attribute

The `nounlink` attribute complements the immutability of files or directories in ZFS by securing them from being accidentally removed. However, unlike the `immutable` attribute, `nounlink` only prevents a file from being deleted or renamed. The file can still be changed by applications or by users.

This behavior is useful for a broad set of files, for example, log files and datafiles from your database. One obvious requirement is that your application must not delete the files as a regular pattern of operation. The `nounlink` attribute would prevent the deletion.

EXAMPLE 23 Protecting Files in a Directory From Deletion

In this example, an administrator who is assigned the Object Access Management rights profile and a `pfbash` login shell prevents the accidental deletion of important applications. With the `nounlink` attribute set on a directory, the file owners, the administrator, and even the `root` role must take extra steps to delete the files in that directory.

```
$ cd /apps/ADMIN
$ chmod S+vnounlink
$ chmod touch test2
$ chmod echo text >> test2
$ cat test2
    text
$ rm test2
    rm: test not removed: Not owner
$ chmod S-vnounlink test2
$ rm test2
$ ls test2
    test2: No such file or directory
```

The owner can still update the files in the directory, and can still remove the file contents by accident. But, even as `root`, the files are undeletable without removing the `nounlink` attribute.

`nounlink` can make a single file undeletable:

```
$ cd /apps/ADMIN
$ chmod S+vnounlink importantApp
```

Displaying and Changing ZFS File Attributes

You can display and set special attributes with the following syntax:

```
$ ls -l/v file.3
-r--r--r--   1 root     root      206695 Jul 20 14:59 file.3
{archive,nohidden,noreadonly,nosystem,noappendonly,nonodump,
noimmutable,av modified,noav_quarantined,nonounlink,nooffline,nosparse}
$ chmod S+cR file.3
```

```
$ ls -l/v file.3
-r--r--r--   1 root     root      206695 Jul 20 14:59 file.3
{archive,nohidden,readonly,nosystem,noappendonly,nonodump,noimmutable,
av_modified,noav_quarantined,nonounlink,nooffline,nosparse}
```

Some of these attributes apply only in an Oracle Solaris SMB environment.

You can clear all attributes on a file. For example:

```
$ chmod S-a file.3
$ ls -l/v file.3
-r--r--r--   1 root     root      206695 Jul 20 14:59 file.3
{noarchive,nohidden,noreadonly,nosystem,noappendonly,nonodump,
noimmutable,noav_modified,noav_quarantined,nonounlink,nooffline,nosparse}
```

♦ ♦ ♦ CHAPTER 3

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 53.

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 64. To create a manifest, use the `bart create` command, as described in "How to Create a Control Manifest" on page 54.

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 67. To create a report, use the `bart compare` command, as described in "How to Compare Manifests for the Same System Over Time" on page 58.

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 66 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 63.

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 54
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 56
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 58 "How to Compare Manifests From Different Systems" on page 60
(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 62 "How to Customize a BART Report by Using a Rules File" on page 63

▼ 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 52. To understand the format conventions, see Example 24, "Explanation of the BART Manifest Format," on page 55.

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 11.3*.

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 `-` (minus sign) 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 24, "Explanation of the BART Manifest Format," on page 55.

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 24 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 11.3*.

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 54.

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 11.3*.

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 54.

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 25 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 11.3*.

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 54.

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 26 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 11.3*.

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 11.3*.

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 54.

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 27 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 66.

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 28 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 24, "Explanation of the BART Manifest Format," on page 55.

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 66.

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

File Security Glossary

Access Control List (ACL)
A list associated with a file that contains information about which users or groups have permission to access or modify the file. 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.

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 home directories be encrypted.

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. 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. For a full description of privileges, see the `privileges(5)` man page.

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.

privileged user
A user whom you have decided can perform administrative tasks at some level of trust.

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

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 profile Also referred to as a *profile*. A collection of security overrides that enable regular users to perform privileged actions.

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

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 policy See policy.

Index

B

C

D

E

F

G

I

K

L

Q

R

S

T

U

V

W

Z