Securing Files and Verifying File Integrity in Oracle Solaris 11.4

E61022-02
August 2023

ORACLE

Securing Files and Verifying File Integrity in Oracle Solaris 11.4,

E61022-02

Primary Author: Cathleen Reiher, Sharon Veach

Contents

2 Using ACLs and Attributes to Protect Oracle Solaris ZFS Files

3 Labeling Files for Data Loss Protection

4 Verifying File Integrity by Using BART

Using This Documentation

Product Documentation Library

Documentation and resources for this product and related products are available at http://www.oracle.com/pls/topic/lookup?ctx=E37838-01.

Feedback

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

1

Controlling Access to Files

What's New in Files and File Systems in Oracle Solaris 11.4

This section highlights information for existing customers about important new features in files and file systems.

- Oracle Solaris labels data and user processes for privacy. This feature provides data loss protection for directories and information that site security requires to have special protections. While labeling is always on, it does not change the behavior of the system until the administrator configures a labeling hierarchy, applies labels to particular files and directories, and enables trusted users to run labeled processes.

  For more information, see Labeling Files for Data Loss Protection and Chapter 6, Labeling Processes for Data Loss Protection in *Securing Users and Processes in Oracle Solaris 11.4*.

- The `-P` and `-H` options to the recursive `chmod -R` command limit file permission changes across symbolic links. See How to Change File Permissions Across Symbolic Links and the `chmod`(1) man page.

- If your site uses the `account-policy` stencil, files in the `/etc` directory that contain security attributes, such as `/etc/default/login`, might not reflect the security policy of the system. Rather, the values of properties in the `account-policy:default` service indicate the security policy of the system. When the `account-policy` service is enabled, changes in the files in the `/etc` directory likely has no effect on security policy. For more information, see Modifying Rights System-Wide As SMF Properties in *Securing Users and Processes in Oracle Solaris 11.4* and the `account-policy`(8S) man page.

- Includes storage for per-user content in private file-system directories in the `/var/share/user` and `/tmp/volatile-user` directories. For more information, see the `filesystem`(7) man page.

Using UNIX Permissions to Protect Files

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

Commands for Viewing and Securing Files

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

Table 1-1 Commands for Securing Files and Directories

Command	Description	Man Page
`ls`	Lists the files in a directory and information about the files.	`ls`(1)

Table 1-1 (Cont.) Commands for Securing Files and Directories

Command	Description	Man Page
`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, who 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 Enabling Users to Change the Ownership of Their Own Files.

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

- (Minus symbol)
Text or program

b
Block special file

c
Character special file

d
Directory

l
Symbolic link

s
Socket

D
Door

P
Named pipe (FIFO)

UNIX File Permissions

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

Table 1-2 File and Directory Permissions

Symbol	Permission	Object	Description
`r`	Read	File	Designated users can open and read the contents of a file.
`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 execute the program with one of the `exec(2)` system calls.
`x`	Execute	Directory	Designated users can open files or run 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.

`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        62K Jun 14 14:14 /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      149K Jun 14 14:04 /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.

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 74  root  sys   18K Sep  7 17:07 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 Setting Special File Permissions in Absolute Mode.

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.

> **Note:**
>
> If you are using the `account-policy` service, you must modify the `login/environment/umask` SMF property. For more information and the procedure, see New Feature – Enabling the account-policy Service in *Securing Users and Processes in Oracle Solaris 11.4* and Modifying Login Environment Variables in *Securing Users and Processes in Oracle Solaris 11.4*. See also the `account-policy`(8S) man page.

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

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

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

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

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

File Permission Modes

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

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

- **Absolute Mode –** Use numbers to represent file permissions. When you change permissions by using the absolute mode, you represent permissions for each triplet by an

octal mode number. Absolute mode is the method most commonly used to set permissions.

- **Symbolic Mode –** Use combinations of letters and symbols to add permissions or remove permissions.

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

Table 1-4 Setting File Permissions in Absolute Mode

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

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

Table 1-5 Setting File Permissions in Symbolic Mode

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

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

who
Specifies whose permissions are to be changed.

operator
Specifies the operation to be performed.

permissions
Specifies what permissions are to be changed.

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

Table 1-6 Setting Special File Permissions in Absolute Mode

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

Using 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 Applying Immutability to a ZFS File.
- `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 Applying Special Attributes to ZFS Files.

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 Setting ACLs on 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 11.4*.

To prevent system compromise by executables in a mounted file system, you can use the `nosetuid` and `noexec` arguments to the `mount` command. For more information, see the `mount`(8) 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 procedures show how to display and change file permissions.

- How to Display File Information
- How to Change the Owner of a File
- How to Change Group Ownership of a File
- How to Change File Permissions in Symbolic Mode
- How to Change File Permissions in Absolute Mode
- How to Change Special File Permissions in Absolute Mode
- How to Change File Permissions Across Symbolic Links

How to Display File Information

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

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

  ```
  % ls -la
  ```

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

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

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

Example 1-1 Displaying File Information

In this 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      21K May 31  2016 6to4relay*
lrwxrwxrwx   1 root   root      10 May 31  2016 accept -> cupsaccept*
-r-xr-xr-x   1 root   bin      57K May 31  2016 acctadm*
-r-xr-xr-x   2 root   sys      94K May 31  2016 add_drv*
-r-xr-xr-x   1 root   bin      26K May 31  2016 admhist*
drwxr-xr-x   2 root   bin        9 May 31  2016 amd64
-r-xr-xr-x   1 root   bin      156 May 31  2016 archiveadm*
-r-xr-xr-x   1 root   bin      21K May 31  2016 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.
- Permissions – For example, `r-xr-xr-x`. For description, see File and Directory Ownership.
- 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 or kilobytes – For example, `156` and `21K`.
- Date the file was created or the last date that the file was changed – For example, `May 31 2016`.
- Name of the file – For example, `arp`.

How to Change the Owner of a File

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 11.4*.

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

```
% ls -l example-file
-rw-r--r--   1 janedoe   staff   12K 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   12K May 31 08:58 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 11.4*.

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

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

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

```
set rstchown = 0
```

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

How to Change Group Ownership of a File

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 11.4*.

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 11.4*.

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 Enabling Users to Change the Ownership of Their Own Files.

How to Change File Permissions in Symbolic Mode

In this 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 Setting File Permissions in Symbolic Mode.

   filename
   Specifies the file or directory.

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

   ```
   % ls -l filename
   ```

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

Example 1-3 Changing Permissions in Symbolic Mode

In this 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 this 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 this 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 Setting File Permissions in Absolute Mode.

   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 the "ACL Operation" section of the `chmod`(1) man page and Setting ACLs on ZFS Files.

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

   ```
   % ls -l filename
   ```

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

Example 1-4 Changing Permissions in Absolute Mode

In this 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    6K Aug  7 12:06 public_dir
# chmod 755 public_dir
# ls -ld public_dir
drwxr-xr-x  1 jdoe   staff    6K Aug  7 12:06 public_dir
```

In this 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    6K Aug  7 12:06 my_script
% chmod 700 my_script
% ls -l my_script
-rwx------ 1 jdoe   staff    6K Aug  7 12:06 my_script
```

How to Change Special File Permissions in Absolute Mode

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 11.4*.

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 Setting Special File Permissions in Absolute Mode.

   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. Review the "ACL Operation" section of the `chmod`(1) man page. 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 1-5 Setting Special File Permissions in Absolute Mode

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

```
# chmod 4555 dbprog
# ls -l dbprog
-r-sr-xr-x   1 db     staff        12K May  6 09:29 dbprog
```

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

```
# chmod 2551 dbprog2
# ls -l dbprog2
-r-xr-s--x   1 db     staff       24K May  6 09:30 dbprog2
```

In this 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
```

How to Change File Permissions Across Symbolic Links

If you are not the owner of the directory, you must be assigned the Object Access Management rights profile. To change a directory 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 11.4*.

1. **Change directory permissions in directories and files that are objects of symbolic links.**

   Choose one of the following options used with the recursive `-R` option of the `chmod` command.

   - `-P` – Changes the mode of a directory or file unless it is the object of a symbolic link, which prevents the traversal of any symbolic links.

     ```
     $ chmod -R -P mode
     directory | file
     ```

     This option is the most secure setting. To see what has changed or not changed and why, add the verbose `-v` option.

   - `-H` – Changes the mode of a directory or file that is the object of a symbolic link and the files in the file hierarchy below the directory.

     ```
     $ chmod -R -H mode
     directory | file
     ```

     If a symbolic link is found when traversing a file hierarchy, the mode of the target directory is changed but no recursion takes place. To see what has changed or not changed and why, add the verbose `-v` option.

   - `-L` – Changes the mode of objects of symbolic links and files by traversing all symbolic links recursively. This option is the default option and is the most permissive.

```
$ chmod -R -L mode
directory | file
```

For more information, see the chmod(1) man page.

2. **Verify that the permissions of all files and subdirectories are correct.**

```
% ls -lR
directory
```

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 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 11.4* prevents programs from malicious software attacks.

How to Find Files With Special File Permissions

You must assume the `root` role. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

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`.

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.

Example 1-6 Finding Files With `setuid` Permissions

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

After investigating the `/usr/rar` directory and removing the `/usr/rar/bin/pfedit` 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    47K  Jul 14 14:14  /usr/bin/atq
-rwsr-xr-x  1 root   sys    54K  Jul 14 14:14  /usr/bin/atrm
-rwsr-xr-x  1 root   bin   145K  Jul 14 14:14  /usr/bin/cdrw
-r-x--s--x  1 root   bin   149K  Jul 14 14:14  /usr/bin/mail
-r-sr-sr-x  1 root   sys    62K  Jul 14 14:14  /usr/bin/passwd
-rwsr-xr-x  1 root   rar    58K  Jul 24 14:14  /usr/rar/pfedit
-r-s--x--x  1 root   bin   208K  Jul 14 14:14  /usr/bin/sudo
-r-sr-xr-x  2 root   bin    26K  Jul 14 14:14  /usr/bin/uptime
# mv /var/tmp/ckprm /var/share/sysreports/ckprm
```

Preventing `tmpfs` File Systems From Filling Up the System

The size of the `tmpfs` file system is not limited by default. Therefore, `tmpfs` can grow to fill the available system memory and swap. Because the `/tmp` directory is used by all applications and users, an application can fill all available system memory. Similarly, an unprivileged user with malicious intent could cause a system slowdown by creating large files in the `/tmp` directory. To avoid a performance impact, you should limit the size of each `tmpfs` mount.

How to Limit the Size of the `tmpfs` File System

To edit the `vfstab` file, you must become an administrator who is assigned the `solaris.admin.edit/etc/vfstab` authorization. To read the changes into the OS, you must be assigned the Service Configuration rights profile. The `root` role has all of these rights. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

In this procedure, you base the size of the `tmpfs` file system on a percentage of system memory. You might try several values to achieve best system performance.

1. **Determine the amount of memory on your system.hardwareSPARC T series serversSPARC T series serversTMPFS configuration example**

> **Note:**
>
> The SPARC T7 series system that is used for the following example has 32 disks. The system has around 500 GB of memory.

```
% prtconf | head
System Configuration:  Oracle Corporation  sun4v
Memory size: 523776 Megabytes
```

```
System Peripherals (Software Nodes):

ORCL,SPARC-T7-1
    scsi_vhci, instance #0
        disk, instance #28
        disk, instance #29
        disk, instance #6
        disk, instance #5
        disk, instance #32
        ...
```

2. **Compute a memory limit for tmpfs.**

   Depending on the size of the system memory, you might want to compute a memory limit of around 20 percent for large systems and around 30 percent for smaller systems.

   - For a larger system, use `.20` as the multiplier.

     523776M x .20 ≈ 104755M

   - For a smaller system, use `.30` as the multiplier.

     260352M x .30 ≈ 78105M

3. **Modify the swap entry in the `/etc/vfstab` file with the size limit.**

```
# pfedit /etc/vfstab
#device     device       mount       FS      fsck    mount mount
#to mount   to fsck      point       type    pass    at boot options
#
...
#swap       -            /tmp        tmpfs   -       yes     -
swap        -            /tmp        tmpfs   -       yes     size=104700m
/dev/zvol/dsk/rpool/swap   -      -  swap    -       no      -
```

4. **Restart the `svc:/system/filesystem/local:default` service.**

```
# svcadm restart filesystem/local
```

5. **Verify that the size limit is in effect.**

```
% mount -v
swap on /system/volatile type tmpfs
read/write/setuid/devices/rstchown/xattr/dev=89c0006 on Thurs Feb 4 14:07:27 2016
swap on /tmp type tmpfs
read/write/setuid/devices/rstchown/xattr/size=104700m/dev=89c0006 on Thurs ...
```

6. **Monitor the memory usage and adjust it to the requirements of your site.**

   The `df` command is somewhat useful. The `swap` command provides the most useful statistics.

```
% df -h /tmp
Filesystem Size Used Available Capacity Mounted on
swap          7.  4G     44M    7.4G 1%       /tmp

% swap -s
total: 190248k bytes allocated + 30348k reserved = 220596k used,
7743780k available
```

   For more information, see the `tmpfs`(4FS), `mount_tmpfs`(8), `df`(8), and `swap`(8) man pages.

2

Using ACLs and Attributes to Protect Oracle Solaris ZFS Files

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 information about ACLs and backup products, see Saving ZFS Data With Other Backup Products in *Managing ZFS File Systems in Oracle Solaris 11.4*.

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 has the following entries:

  ```
  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
  ```

  For a newly created directory, the default ACL has the following entries:

  ```
  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
  ```

- **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 might look like the following example, where permissions are specifically granted to user Jan.

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

ACL Entry Descriptions

Use the following sample entry as a reference to understand 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. In non-trivial ACLs, `user`:*username* and `group`:*groupname* are added. In the example, the entry type is `user:jan`.

Access privileges
Permissions that are granted or denied to the entry type. In the example, user Jan's permissions are `read_data` and `write_data`.

Inheritance flags
An optional list of ACL flags that control how permissions are propagated in a directory structure, including flags that audit access to files and directories. In the sample entry, `file_inherit` is also granted to user Jan.

Audit flag
An optional flag that enables you to audit access and changes that are being made to a file.

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

The following table describes each ACL entry type.

Table 2-1 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 contains a user name or user ID. If the value is not a valid numeric UID or user name, the ACL entry type is invalid.
`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 contains a group name or group ID. If the value is not a valid numeric GID or group name, the ACL entry type is invalid.

The following table describes ACL access privileges.

Table 2-2 ACL Access Privileges

Access Privilege	Compact Access Privilege	Description
add_file	w	Permission to add a new file to a directory.
add_subdirectory	p	On a directory, permission to create a subdirectory.
append_data	p	On a file, permission to modify from the end of the file (EOF).
delete	d	Permission to delete a file. For more information about specific delete permission behavior, see ACL delete and delete_child Permission Behavior.
delete_child	D	Permission to delete a file or directory within a directory. For more information about specific delete_child permission behavior, see ACL delete and delete_child Permission Behavior.
execute	x	Permission to execute a file or search the contents of a directory.
list_directory	r	Permission to list the contents of a directory.
read_acl	c	Permission to read the ACL (ls).
read_attributes	a	Permission to read basic attributes (non-ACLs) of a file, which are equivalent to stat level attributes. Allowing this access mask bit means the entity can execute ls(1) and stat(2).
read_data	r	Permission to read the contents of the file.
read_xattr	R	Permission to read the extended attributes of a file or perform a lookup in the file's extended attributes directory.
synchronize	s	Permission to access a file locally at the ZFS server with synchronized read and write operations.
write_xattr	W	Permission to create extended attributes or write to the extended attributes directory. Granting this permission to a user means that the user can create an extended attribute directory for a file. The attribute file's permissions control the user's access to the attribute.
write_data	w	Permission to modify or replace the contents of a file.
write_attributes	A	Permission to change the times associated with a file or directory to an arbitrary value.
write_acl	C	Permission to write the ACL or the ability to modify the ACL by using the chmod command.
write_owner	o	Permission to change the file's owner or group. Or, the ability to execute the chown or chgrp commands on the file. Permission to take ownership of a file or permission to change the group ownership of the file to a group of which the user is a member. If you want to change the file or group ownership to an arbitrary user or group, then the PRIV_FILE_CHOWN privilege is required.

The following table provides additional details about ACL delete and delete_child behavior.

Table 2-3 ACL `delete` and `delete_child` Permission Behavior

Parent Directory Permissions	Target Object Permissions		
" " (empty)	ACL allows delete	ACL denies delete	Delete permission unspecified
ACL allows `delete_child`	Permit	Permit	Permit
ACL denies `delete_child`	Permit	Deny	Deny
ACL allows only `write` and `execute`	Permit	Permit	Permit
ACL denies `write` and `execute`	Permit	Deny	Deny

ZFS ACL Sets

An ACL set consists of a combination of ACL permissions. These ACL sets of permissions are predefined and cannot be modified.

- `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 having to set individual permissions separately.

Example 2-1 Using an ACL Set to Assign a Combination of ACL Permissions

With the `read_set` ACL set, the user `jan` can read ACLs as well as file contents and their basic and extended attributes.

```
$ 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 Inheritance

ACL inheritance means that a newly created file or directory can inherit the ACLs that they are intended to inherit without disregarding the existing permission bits on the parent directory.

By default, ACLs are not propagated. If you set a non-trivial ACL on a directory, it is not inherited to any subsequent directory. You must specify the inheritance of an ACL on a file or directory.

The following table describes the optional inheritance flags.

Table 2-4 ACL Inheritance Flags

Inheritance Flag	Compact Inheritance Flag	Description
file_inherit	f	Only inherit the ACL from the parent directory to the directory's files.
dir_inherit	d	Only inherit the ACL from the parent directory to the directory's subdirectories.
inherit_only	i	Inherit the ACL from the parent directory. Applies only to newly created files or subdirectories and not the directory itself. This flag requires the file_inherit flag, the dir_inherit flag, or both, to indicate what to inherit.
no_propagate	n	Only inherit the ACL from the parent directory to the first-level contents of the directory, not the second-level or subsequent contents. This flag requires the file_inherit flag, the dir_inherit flag, or both, to indicate what to inherit.
–	N/A	No permission granted.
successful_access	S	Indicates whether an alarm or audit record should be initiated upon a successful access. This flag is used with audit or alarm ACE types.
failed_access	F	Indicates whether an alarm or audit record should be initiated when an access fails. This flag is used with audit or alarm ACE types.
inherited	I	Indicates that an ACE was inherited.

In addition, you can set a default ACL inheritance policy on the file system that is more strict or less strict by using the aclinherit file system property. For more information about this property, see ACL Properties.

For more information about setting ACL inheritance on ZFS files, see Setting ACL Inheritance on ZFS Files.

ACL Properties

The ZFS file system includes the ACL properties to determine the specific behavior of ACL inheritance and ACL interaction with chmod operations. These properties are:

- aclinherit – Determine the behavior of ACL inheritance. Values include the following:
  - restricted – For new objects, the write_owner and write_acl permissions are removed when an ACL entry is inherited. This is the default mode.
  - discard – For new objects, no ACL entries are inherited when a file or directory is created. The ACL on the file or directory is equal to the permission mode of the file or directory.
  - noallow – For new objects, only inheritable ACL entries that have an access type of deny are inherited.
  - passthrough – When a property value is set to passthrough, files are created with a mode determined by the inheritable ACEs. If no inheritable ACEs exist that affect the

mode, then the mode is set in accordance to the requested mode from the application.

  - `passthrough-x` – Has the same semantics as `passthrough` except that files are created with the execute (`x`) permission only if the execute permission is set in file creation mode and in an inheritable ACE that affects the mode.
  - `passthrough-mode-preserve` – A file system has the same semantics as `passthrough` except that the `owner@`, `group@`, and `everyone@` ACEs are overridden by values from the mode that is requested by the application when creating files and directories.

  For more information about the `aclinherit` modes, see Modifying ACL Inheritance With the ACL Inherit Mode.

- `aclmode` – Modifies ACL behavior when a file is initially created or controls how an ACL is modified during a `chmod` operation. Values include the following:
  - `discard` – Deletes all ACL entries that do not represent the mode of the file. This is the default mode.
  - `mask` – Reduces user or group permissions. The permissions are reduced such that they are no greater than the group permission bits unless it is a user entry that has the same UID as the owner of the file or directory. In this case, the ACL permissions are reduced so that they are no greater than owner permission bits. The mask value also preserves the ACL across mode changes, provided that an explicit ACL set operation has not been performed.
  - `passthrough` – Indicates that no changes are made to the ACL other than generating the necessary ACL entries to represent the new mode of the file or directory.

  For more information about using the `aclmode` property, see ACL Properties and Modified ACL Permissions.

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 privilege subsystem determines the access request that is granted for the owner of the file or for superuser. This mechanism prevents owners of files from getting locked out of their files and enables superuser to modify files for recovery purposes.

Command Syntax for Setting 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 specify *A* before typing the operator (+, =, or -).

- Command syntax for trivial ACLs

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

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

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

- Command syntax for non-trivial ACLs

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

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

The `chmod` command uses the following operators:

- `A+` adds an ACL entry.
- `A=` replaces an ACL entry.

  To replace an entire ACL for a file, use this operator without specifying an index ID. In the following example, ACL entries for `file.1` are removed and replaced with the single entry for `everyone@`.

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

- `A-` removes an ACL entry.

  To universally remove all non-trivial ACL entries for a file, use this operator and specify the file name without listing each entry to be removed.

  ```
  $ chmod A- filename
  ```

  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.

> **Caution:**
>
> Be careful with modifying existing ACLs. Using the operators without an index has a different effect from using them with an index. For example, `chmod A=` replaces an entire ACL, while `chmod A3=` replaces only the existing entry that has index number 3.

Permissions and inheritance flags are represented by unique letters listed in ACL Access Privileges and ACL Inheritance Flags. 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).

In this example, both commands grant read and execute permissions to user Alice on `file.1`:

- `chmod A+user:alice:rx:allow file.1`
- `chmod A+user:alice:read_data/execute:allow file.1`

Likewise, to grant user Tamiko inheritable read, write, and execute permissions for the newly created `dir.2` and its files, you can use either one of the following commands:

- `chmod A+user:tamiko:rwx:fd:allow dir.2`
- `chmod A+user:tamiko:read_data/write_data/execute:file_inherit/dir_inherit:allow dir.2`

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.

The following example shows how the same ACL information is displayed in both verbose 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 an explanation of the permissions for each user category, see ACL Access Privileges.

Modifying ACLs on ZFS Files

This section provides sample commands for setting and displaying ACLs.

In the following example, `write_data` permissions are granted for `group@`. The index of `group@` is `1`.

```
# 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
```

In the following example, `read_data/execute` permissions are added for the user Alice on the `test.dir` directory.

```
$ 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.

```
$ 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
```

In the following example, auditing is set for `everyone@` in `dir.1`. If any user attempts to access `dir.1` and fails, that access failure is recorded in the audit log.

```
$ 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
```

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 2-2 ACLs and Permission Bits

The first example begins with the following ACL for `file.2`, whose permission bits are set to 644.

```
$ ls -v file.2
-rw-r--r--   1 root     root        2693 Jul 20 14:26 file.2
Permission bits are 644.
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 bits for `everyone` are also removed and are changed to 640.

```
$ chmod A2- file.2
Access is removed for everyone@
$ ls -v file.2
-rw-r-----   1 root     root        2693 Jul 20 14:26 file.2Permission bits
become 640.
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@`. No `owner@` or `group@` ACL entry exists to override the permissions for `owner` and `group`. Consequently, the permission bits become 666.

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

Next, the ACL is replaced with read permissions just for user Alice. The command, however, leaves no trivial ACL entries. Consequently, the permission bits are set to 000, which denies Alice access to `file.2`. The file effectively becomes inaccessible.

```
$ chmod A=user:alice:r:allow file.2
$ ls -v file.2
----------+  1 root     root        2440 Jul 20 14:28 file.3Permission bits
become 000.
0:user:alice:read_data:allow
```

The example ends with showing how setting permission bits also update the ACL. The bits for `file.2` are set to 655. Automatically, default trivial ACL permissions are set.

```
$ chmod 655 file.2
$ ls -v file.3
-rw-r-xr-x   1 root     root        2440 Jul 20 14:28 file.3Permission bits set
to 655.
0:user:alice:read_data:allow
1:owner@:execute:denyBased on 655 bits, ACL execute permission is denied.
2:owner@:read_data/write_data/append_data/read_xattr/write_xattrDefault ACL
```

```
entries restored.
/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 2-3 ACL Properties and Modified 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.

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

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

Amy then changes `file.1`'s permission bits to 640. Because of the ACL properties that were previously set, the permissions for the groups in the ACL are reduced in order to 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.

Granting ACLs That Are Inherited by Files

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 Management rights profile adds `read_data/write_data` permissions for user `alice` so that she has read access on any newly created files 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. Because she is granted `read_data:file_inherit:allow`, she can read the contents of any newly created 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 the `aclinherit` property for this file system is set to the default mode, `restricted`, user `alice` does not have `write_data` permission on `file.2` because the group permission of the file does not allow it.

The `inherit_only` permission, which is applied when the `file_inherit` or `dir_inherit` flags are set, is used to propagate the ACL through the directory structure. As such, user `alice` is granted or denied permission from `everyone@` permissions only if he 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
```

Granting ACLs That Are Inherited by Both Files and Directories

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.

In the following 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 following output is an informational message that indicates that the ACE is inherited.

```
$ 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
```

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. The default `aclinherit` property is `restricted`, which means that `write_data` and `execute` permissions are not inherited.

In the following 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 the following example illustrates, `alice`'s `read_data/write_data/execute` permissions are reduced based on the owning group's permissions.

```
$ 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
```

Modifying ACL Inheritance With the ACL Inherit Mode

This section describes the `aclinherit` property values.

Example 2-4 ACL Inheritance With the ACL Inherit Mode Set to `discard`

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 2-5 ACL Inheritance With the ACL Inherit Mode Set to `noallow`

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/cindy
$ 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
```

As the following example shows, 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 2-6 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 2-7 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.

ACL Inherit `passthrough-mode-preserve` Mode

The following section includes examples of using the `aclinherit=passthrough-mode-preserve` property setting.

The following parent directory has the following ACL and the `aclinherit=passthrough-mode-preserve` property setting. Note that this setting configures an inheritance that prevents an SMB server from creating a two-member ACL, which affects NFS clients negatively.

```
drwxrwxrwx+  4 nobody   other          4 Oct 15 13:49 .
user:marks:rwxp--aAR-----:fd-----:allow
owner@:rwxp-DaARWcCos:fd-----:allow
group@:rwxp-DaARWc--s:fd-----:allow
everyone@:rwxp-DaARWc--s:fd-----:allow
```

Directly creating an SMB directory results in the directory having the following ACL:

```
# ls -dV smb.dir
drwxrwxrwx+  2 marks    staff          2 Oct 15 14:03 smb.dir
user:marks:rwxp--aAR-----:fd----I:allow
owner@:rwxp-DaARWcCos:fd----I:allow
group@:rwxp-DaARWc--s:fd----I:allow
everyone@:rwxp-DaARWc--s:fd----I:allow
```

In the SMB case, the `passthrough-mode-preserve` property setting configures a pure inheritance of the ACEs and no longer creates the two-member ACL shown previously.

Using NFS to create the `dir2` directory results in the directory having the following ACL:

```
# umask
0022
# mkdir dir2
# ls -dV dir2
drwxr-xr-x+  2 root     root           2 Oct 15 13:49 dir2
user:marks:r-x---a-R-----:fd----I:allow
owner@:rwxp-DaARWcCos:fd----I:allow
group@:r-x---a-R-c--s:fd----I:allow
everyone@:r-x---a-R-c--s:fd----I:allow
```

Note that a umask of 0022 results in the `mkdir` command creating the directory with a mode of 0755.

The `owner@`, `group@`, and `everyone@` entry values are overridden by the mode values specified by the `mkdir` request.

If you set the `aclmode` property to `mask`, running the `chmod 700 dir2` command creates the following ACL:

```
# chmod 700 dir2
# ls -dV dir2
drwx------+  2 root     root           2 Dec  1 13:51 dir2
```

```
user:marks:------a-R-----:fd----I:allow
owner@:rwxp-DaARWcCos:fd----I:allow
group@:------a-R-c--s:fd----I:allow
everyone@:------a-R-c--s:fd----I:allow
```

In this case, the permissions for `owner@`, `group@`, and `everyone@` are replaced to adjust the ACL to mode 0700. The `marks` entry is updated in accordance with the existing `mask` semantics. All of the inheritance bits are preserved.

Setting the `aclmode` property to `discard` results in the following ACL:

```
# chmod 755 dir2
# ls -dV dir2
drwxr-xr-x   2 root     root           2 Dec  1 13:51 dir2
owner@:rwxp-DaARWcCos:fd-----:allow
group@:r-x---a-R-c--s:fd-----:allow
everyone@:r-x---a-R-c--s:fd-----:allow
```

This case preserves the inheritance bits again. This behavior occurs only when you set the `aclinherit` property to `passthrough-mode-preserve`.

The following examples set the `aclinherit` property value to `passthrough`, which results in different behavior from inheriting ACLs and results in a different effect from the `chmod` command.

```
# mkdir dir3
# ls -dV dir3
drwxrwxrwx+  2 root     root           2 Dec  1 15:46 dir3
user:marks:rwxp--aAR-----:fd----I:allow
owner@:rwxp-DaARWcCos:fd----I:allow
group@:rwxp-DaARWc--s:fd----I:allow
everyone@:rwxp-DaARWc--s:fd----I:allow
```

The previous `mkdir dir3` command inherits all of the ACEs directly from the parent directory and overrides the creation-mode passed to the `mkdir` command. Also, this command ignores the user's umask. Use this setting when you want to force the creation mode of every file and directory to be the same value. Note that the umask and creation mode are ignored only if one or more inheritable `owner@`, `group@`, or `everyone@` ACEs exist.

Now, using the `chmod 0700 dir3` command results in the following ACL when the `aclmode` property is set to `mask`:

```
drwxr-xr-x+  2 root     root           2 Dec  1 15:46 dir3
user:marks:r-x---a-R-----:fd----I:allow
owner@:rwxp-DaARWcCos:fdi---I:allow
group@:rwxp-DaARWc--s:fdi---I:allow
everyone@:rwxp-DaARWc--s:fdi---I:allow
owner@:rwxp-DaARWcCos:-------:allow
group@:r-x---a-R-c--s:-------:allow
everyone@:r-x---a-R-c--s:-------:allow
```

This `chmod` commands splits the `owner@`, `group@`, and `everyone@` ACEs into two sets of entries. Note that the `fdi`-marked entries apply to inheritance only and are not considered for access-control decisions. These entries exist for future propagation. The second set of `owner@`, `group@`, and `everyone@` ACEs reflects the mode that you requested with the `chmod` command.

Setting the `aclmode` property value to `discard` results in the following ACL:

```
# ls -dV dir3
drwxr-xr-x   2 root     root           2 Dec  1 15:46 dir3
owner@:rwxp-DaARWcCos:-------:allow
group@:r-x---a-R-c--s:-------:allow
everyone@:r-x---a-R-c--s:-------:allow
```

This setting replaces the original ACL with a new one that corresponds to the new file mode.

Applying Special Attributes to ZFS Files

This section shows how to apply special attributes to ZFS files and how to display them. For more information about displaying and applying special attributes, see the `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.

Applying Immutability to a ZFS File

Use the following syntax to make a file immutable:

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

You can display special attributes on ZFS files by using the following syntax:

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

Use the following syntax to remove file immutability:

```
$ 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----}
$ rm file.1
```

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.

For some examples, see the following blog entry.

Applying Read-Only Access to a ZFS File

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: Insufficient privileges
```

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}
```

3

Labeling Files for Data Loss Protection

About Labeling in Oracle Solaris

Oracle Solaris enables you to configure systems that enforce company security policy in software by using labels. You can use provided labels or customize the labels to display your organization's security phrases, such as Confidential - Internal Only. An Oracle Solaris label policy enables you to assign these labels to existing or new file systems that contain sensitive data, and assign a set of trusted users the ability to access the files based on the users' clearances. This selective access is useful for file systems that contain data such as credit card numbers, financial records, and marketing plans. Regular users will work within your default label policy, such as Confidential - Internal Only. They cannot access files at a higher security level, such as Confidential - Restricted.

You can use labeling with other features of Oracle Solaris, including package update, immutable zones, automated install, and SMF to provide robust and easily maintained systems that protect data at every stage of the lifecycle, from file creation to file archival and retrieval. This section provides an overview of labeling terminology and use in Oracle Solaris.

Label Policy

Labels in Oracle Solaris implement a set of access rules for sensitive data that is in addition to discretionary access control (DAC). You configure labels to reflect your site's security policy around sensitive data.

All Oracle Solaris systems have a label policy. By default, the policy is unrestricted so that only DAC controls access to files. A default encodings file enforces this unrestricted label policy. The labeling service that runs on all Oracle Solaris 11.4 systems is `labeld:clearance`.

Label policy is configured in an **encodings** file. Oracle Solaris provides two sample encodings files: the default file and a compliance encodings file. To view these files, see Viewing and Testing Sample Label Encodings Files.

Every system that contains sensitive data must contain a copy of your customized encodings file. One strategy is to put sensitive data in zones on designated systems, label the ZFS datasets in those zones, and restrict access to the data by labeled user processes and SMF service processes.

Most of your file systems will not be labeled. Therefore, their files inherit the system's lowest label, `ADMIN_LOW`. All clearances that you assign to users and processes dominate this label, so all files on unlabeled systems are available to all logins.

To administer labels you must be in the `root` role or be assigned the Object Label Management rights profile.

Labels and Clearances

Oracle Solaris labels files and processes. Labels are assigned to files to indicate the sensitivity of the information. When assigned to processes, labels are called clearances.

Processes such as user processes can access files equal to or lower than the process label. Typical labels are Public and Confidential - Restricted.

Oracle Solaris provides the highest and lowest labels, `ADMIN_HIGH` and `ADMIN_LOW`. These labels cannot be changed or internationalized. The `ADMIN_HIGH` label is number 255 and dominates all classifications and includes all compartments. The `ADMIN_LOW` label, number 0, is the lowest classification and contains no compartments. All labels dominate `ADMIN_LOW`. On an unlabeled system, the `ADMIN_LOW` label cannot be changed. Processes with a clearance can access files that the clearance dominates. For example, a process that runs at Confidential - Restricted can access files at that label and at the Public label.

Label Components

Labels and clearances consist of a single classification and zero or more compartments. The classification portion of a label indicates a relative level of trust. **Classifications** are hierarchical – a higher classification number indicates a higher level of trust. When a label is assigned to a file, the label's classification is one indication of the sensitivity of the information that the file contains.

Compartments provide a more fine-grained mechanism for specifying the user's level of trust. Compartments are typically used to indicate the scope of the trust. For example, a Human Resources compartment would indicate that the level of trust applies to Human Resource materials. When a clearance is assigned to a user, the *classification* portion of the clearance label indicates the user's level of trust and the *compartment bits* typically indicate the department where that level of trust applies.

In contrast to label numbers, the compartment bit numbers do not indicate dominance. However, compartments with subcompartments form a hierarchy that can be used to indicate levels of trust, such as the bits for Highly Restricted including the bit defined for Restricted.

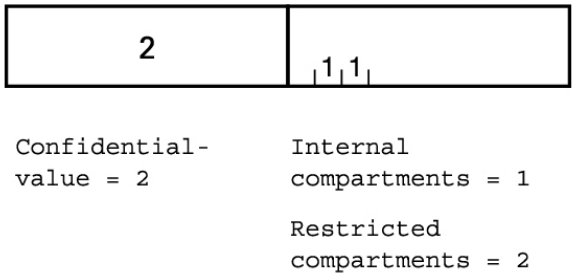
Each classification corresponds to a unique positive integer from 0 to 255. Higher numbers dominate lower numbers. A label **dominates** another label if its classification is at least equal to the other label's classification and its compartments include all the bits in the other label's compartments.

Each compartment corresponds to one or more bits. In Oracle Solaris, the number of available compartment bits is 256, but many thousands of compartments can be created from these bits. You can use compartment bits to define hierarchical, disjoint, and overlapping relationships, as described in Label Relationships. Oracle Solaris assigns bit numbers to the compartments that you name. You can change the bit assignments.

As the administrator, you name your classifications from the lowest classification to the highest and Oracle Solaris assigns the numbers. You can modify the number assignments to redefine the hierarchy. The classification numbers you can use range from 1 to 254.

In the following figure, the label has been assigned a classification of 2. The classification name is "Confidential - ". The compartment names are Internal and Restricted. The Confidential - Internal label uses the classification value and one compartment bit. The Confidential - Restricted label uses the classification value and two bits, compartments 1 and 2.

Sample Label Definitions



Label Relationships

Labels can have hierarchical relationships, disjoint relationships, and overlapping relationships. For a label encodings file that illustrates these relationships, see Example - Label Encodings File With Reused Compartment Bits.

- *Hierarchical* relationships are formed when a label dominates another label. A label dominates another label when its classification is at least equal to the other's classification and its compartments include all the bits in the other's compartments. For example, a classification that you created named Confidential that Oracle Solaris might represent internally as number 3 dominates a classification that you created named Public that is internally represented as 1.

  Compartments are represented as arbitrary numbers. Compartments can be hierarchical when the bits of a subcompartment are a subset of one or more other compartments. Subcompartments can also include their own subcompartments. These subcompartments can contain unique bits in addition to the subsets of the compartment bits. A simple example of a compartment and a subcompartment is Highly Restricted with the Restricted subcompartment. Internally, Oracle Solaris adds the Restricted bit to the Highly Restricted bit, so if the Restricted subcompartment is bit 2, Highly Restricted might be bits 2 and 3.
- *Disjoint* relationships are formed when labels with the same classification have different compartment bits. You can also specify that labels conflict. Disjoint labels are useful to isolate sensitive department information from personnel outside the department. For example, you might create the labels Confidential - Finance: Payroll and Confidential - Finance: Accounts to be disjoint.
- *Overlapping* relationships are formed when compartments share one or more bits but each compartment has at least one unique bit. Overlapping labels are useful to define an alias, such as an Information Technology alias for writers, course developers, web content providers, and editors.

Privileges for Translating Labels

Label translation occurs whenever programs manipulate labels. Labels are translated to and from the textual strings to the internal representation. For example, when a program such as `getlabel` obtains the label of a file, before the label can be displayed to the user, the internal representation of the label is translated into readable output, that is, into a textual string. When the `setlabel` program sets a label specified on the command line, the textual string (that is, the label's name) is translated into the label's internal representation. Oracle Solaris permits label translations only if the calling process's label dominates the label that is to be translated. If a process attempts to translate a label that the process's label does not

dominate, the translation is disallowed. The `sys_trans_label` privilege is required to override this restriction.

Labeled Files and Multilevel File Systems

Labeled files are files that your organization labels due to the sensitivity of their contents. Labeled files are in labeled file systems. Another name for a labeled file system is a **multilevel** file system.

Labeled file systems can have stricter requirements for encryption, auditing, and other security processes. The auditing of access to sensitive files is part of due diligence. The audit record includes both the label of the file and clearance of the active process. The audit service enables you to specify that file-read events are audited for labeled files only.

Labeled file systems complement encryption. Labeling protects data in mounted file systems, while encryption protects data in unmounted file systems, so archived labeled file systems should be encrypted.

By default, all file systems are unlabeled. In a multilevel file system, files can inherit their label from their directory or be assigned a label explicitly by a user whose process dominates the file label. No privilege can override the access policy specified by a label. You must be an administrator to create a labeled file system.

A user's clearance controls whether they can access a labeled file, upgrade or downgrade a file label, archive a multilevel file system, or restore it. The files that the user is operating on must be within the user's clearance. DAC permissions control whether the user can read, write, or execute the file. Note that discretionary access control (DAC) applies to all labeled files.

Sharing and Mounting Labeled File Systems

Labeled file systems can be shared and mounted. Without an explicit `labeled=on` option, only the `ADMIN_LOW` file systems are shared. With the explicit `labeled=on` keyword, users who are cleared at particular labels can access files at those labels.

Access to files on a remotely mounted labeled file system is enforced by the file server's policy. Access is based on the user's clearance as interpreted by the server. Access policy can either be stored locally on the file server or retrieved from a central LDAP repository. Users must have a clearance on the server that is equal to or higher than the files that they want to access. If the file system is not shared as a labeled file system, remote access is limited to `ADMIN_LOW` files, even by privileged users.

Only Oracle Solaris systems that support labeling can mount multilevel file systems. To prevent mount failures, set `canmount=off` for labeled file systems before booting into a non Oracle Solaris 11.4 system.

Protect Data With a Label Policy

The label policy on your computer system is an information protection policy that is enforced in software. For example, an information protection policy classifies its data according to sensitivity, value to the organization, and legal requirements related to confidentiality. Once identified, files that hold sensitive, valuable, or legally required information can be appropriately labeled. Labels such as Confidential - Restricted, and Confidential - Highly Restricted can isolate and protect information in every

department. You can create file systems or modify existing file systems to contain labeled data, and assign individual users the ability to access the sensitive files that they are responsible for.

Users, user processes, and other processes can access data whose label they dominate. How you label processes is part of your label policy. To create labeled processes, see Chapter 6, Labeling Processes for Data Loss Protection in *Securing Users and Processes in Oracle Solaris 11.4*.

Default Label Policy

This section describes the default label policy and considerations when developing a your label policy.

After you install the `pkg:/system/file_labeling` package, you can customize your label policy, add labels to file systems, and assign clearances to users and SMF services. Before customizing your label policy, the default clearance is the highest label, `ADMIN_HIGH`, so access is not restricted by label.

```
$ svcs labeld:clearance
STATE          STIME    FMRI
online         Sep_25   svc:/system/labeld:clearance
```

Displaying Label and Policy Information

To view the policy details, use the `labelcfg info` command. For the steps that created this sample, see the `labelcfg`(8) man page.

```
# labelcfg info
title=Sample Information Protection Policy
classification=Public
    level=1
classification=Confidential -
    level=2
compartment=Highly Restricted
    bit=0
    subcompartments="Restricted"
    minclass=Confidential -
compartment=Restricted
    bit=1
    subcompartments="Internal"
    minclass=Confidential -
compartment=Internal
    bit=2
    minclass=Confidential -
min_label=Public
clearance=ADMIN_HIGH
```

Note that each classification has a numeric equivalent indicated by a `level` number. A higher classification has a higher `level` number. The compartments are differentiated by bits, so `bit` numbers do not indicate higher or lower. Classifications plus their compartments comprise the list of valid labels. When you list the labels, they display from highest label to lowest without displaying the `ADMIN_HIGH` or `ADMIN_LOW` label.

```
# labelcfg list
"Confidential - Highly Restricted"
"Confidential - Restricted"
```

```
"Confidential - Internal"
Public
```

The value of `clearance` in the encodings file applies to users or roles who do not have an explicit key-value setting for the `clearance` security attribute. The `root` role and the initial account that was created during the installation of Oracle Solaris have an explicit clearance, `ADMIN_HIGH`.

> **Caution:**
>
> Never change the explicit `ADMIN_HIGH` clearance of the `root` account.

User processes inherit the clearance of the user's primary login process. To view the clearance of your current process, type `plabel` in a terminal window. You have access to all labels from your clearance to `ADMIN_LOW`.

```
$ plabel
ADMIN_HIGH
```

Customizing a Label Policy

Your label policy protects data during use, just as encryption protects data at rest. The overall process is:

1. Separate sensitive data.
2. Limit access to the data to specific individuals or groups.
3. Monitor the data during use.
4. Archive the data such that machine operators, IT personnel, and users who can assume the `root` role cannot view the information in the files through normal operations.

To configure labeling, you install the labeling package, then configure the labels to satisfy the security requirements of your organization. When configuring a label policy, you supply a minimum label, a maximum label (or **clearance**) for users, and a hierarchy of labels. You can also define disjoint label relationships. At login, the processes of users to whom you assigned a higher clearance start at that clearance. Then, sensitive data that is labeled at a high label can be accessed only by those users whose processes are running at the higher clearance.

You can either use one of the supplied policies, which are sufficient for testing and demonstrations or create your own label policy specific to your organization's requirements regarding its sensitive information.

When creating a label policy, cover the following issues:

- Identify the sensitivity of the data

  For example, credit cards and health records might be considered highly sensitive information, vendor discounts might be sensitive information, vendor visits might be internal information, and marketing announcements would be public information.
- Identify the departments of your organization that handle sensitive data

For example, regulatory bodies require companies that handle credit cards to protect the credit card details and transaction details. Departments of the company that handle credit cards would need labeled file systems, and individual users and roles who are permitted to view the credit card details or handle disputes about credit card use would need sufficient clearance.

- Identify users or roles in each department of your organization whom you trust to handle sensitive material

  For example, you might allow some people in receivables to view credit card information but not others. Those individuals or groups who can modify information would need clearance to do so, as would those who need to view the information.

- Identify departments that should not see information from other departments

  For example, perhaps the executive board should not be able to see credit card information. For highly sensitive information, each department of the company would need its own compartment, for example, Confidential - Highly Restricted(Exec) and Confidential - Highly Restricted(Payments), where Payments handlers do not have access to Executive discussions and Executive users do not have access to payment details. In each group, the information being protected is of high value.

- Identify services that should be protected by a label

  For example, you might protect applications that contain information of high value, such as internal browser interface applications or FTP services.

  See Example - Protecting the FTP Service With a Label in *Securing Users and Processes in Oracle Solaris 11.4*.

Oracle Solaris simplifies the creation of a label policy. As you enter your labels, the software provides the numbers that create the hierarchy of labels as well as the numbers for the compartments that separate departments of your organization. You provide the names that you want, starting at the lowest label. Public or Internal are possible lowest labels. See Configuring Labels on an Oracle Solaris System for a detailed description of the tasks involved in creating and maintaining a custom label policy.

About Hardening Labeled File Systems

Although you can restrict access to sensitive data to users and roles based on their clearances, a label policy does not prevent modifying the underlying configuration, loading untrusted software, or modifying the kernel. It also does not prevent cleared users or roles from copying labeled data to unlabeled directories. Hardening can limit these dangers.

You can put the following protections in place:

- Create non-global zones where selected users are granted a higher clearance than their clearance outside the zone.
- Restrict access to a labeled zone to users who have been delegated the `login` authorization for that zone.
- Make the configuration of the zone immutable.
- Import one or more labeled file systems read-write into the zone from the global zone. The label of each top-level directory is also the minimum label at which data can be written to each file system.
- Remove any network interfaces to prevent leakage outside the zone.

With these protections, when users log in to the zone, their clearance is raised to the value specified in the zone's `user_attr` file. Although users might not be authorized to set individual file labels, all files in the imported file systems are automatically labeled based on their containing directories. Also, although users cannot be prevented from copying files into unlabeled file systems, unlabeled data does not leak outside the zone. The labeled data is available outside of the zone only to users and roles with sufficient clearance.

About Installing a Customized Labels Package

After you create and test the label policy for your site, you can install it as an Image Packaging System (IPS) package on your systems. The package you create must contain the encodings file. Also, the encodings file name must be the value of the `labeld/encodings_file` property of the `labeld:clearance` service in the service manifest.

Use the `labelcfg -e` command to place the active encodings file as the value of the `labeld/encodings_file` property in the service manifest.

```
# labelcfg -e /etc/security/tsol/site-enc
# svccfg -s labeld:clearance listprop labeld/encodings_file
labeld/encodings_file astring     /etc/security/tsol/site-enc
```

File labels and user clearances are stored as hexadecimal strings that encode the classifications and compartments. After installing the package, assigning new names to existing classifications does not affect the internal label representations, although renaming might be confusing for users. Adding additional classifications and compartments does not invalidate existing settings, either. However, do not remove classifications or compartments that are assigned to users or files because existing labels that used those classifications and compartments would then display as hexadecimal strings.

For information about package creation and testing, see Chapter 2, Packaging Software With IPS in *Packaging and Delivering Software With the Image Packaging System in Oracle Solaris 11.4*. For more information about package delivery and installation, see *Updating Systems and Adding Software in Oracle Solaris 11.4* and *Creating a Custom Oracle Solaris 11.4 Image*.

Ideas for Using Labeled File Systems for Data Loss Protection

Labeled file systems protect sensitive files from inadvertent or malicious tampering. You can use labeled file systems in the following ways:

- *Restrict access to core files* – Store core files in labeled file systems so that access to these core files requires label dominance. You can use the `%l` format specification to specify the directory pathname corresponding to the label of the process generating the core file. For more information, see the labeling examples on the `coreadm`(8) man page.
- *Restrict access to audit files* – Store audit files in labeled file systems. A labeled audit trail reduces access to the audit trail, including access to the contents of higher-labeled processes. Access to the audit trail will require label dominance. See How to Create a Labeled Audit Trail.

- *Restrict access to selected directories* – Users can set `TMPDIR` to a labeled directory under their home directory. Similarly, you can configure the `vim` editor so that the backup and swap directories are labeled.
- *Restrict access to DTrace probes* – Running DTrace on a labeled process requires process dominance. For information about DTrace probes, see the `dtrace`(8) man page.
- *Restrict access to database data and configuration* – Make Oracle database instances more robust by assigning a label to the `$ORACLE_HOME` directory to protect the data and configuration files from rogue administrators. An administrator, including `root`, whose process does not dominate the database label would be unable to access the directory. Such labeling provides an extra level of security beyond encryption. For example, another user assuming the `root` role would be unable to change or remove files in `$ORACLE_HOME`.
- *Restrict modification of system configuration* – Make the system configuration immutable by configuring the labeled system with the `fixed-configuration immutable` policy. An immutable policy prevents `root` from altering the labeled configuration. For more information, see the `zonecfg`(8) man page. When an immutable policy is in effect, changes to any method or `sysconfig` properties of any SMF service, including the clearance of the service, requires a clearance of the `ADMIN_HIGH` label from the requesting client. See How to Enforce a Fixed Configuration for a Labeled File System.

Configuring Labels on an Oracle Solaris System

In Oracle Solaris, labels facilitate data loss protection, which is a requirement of certain standards, such as PCI-DSS. For an overview of the steps to use labels for data loss protection, see Overall Process for Configuring Labeling.

Configuring labels involves the following tasks:

- Creating labels – Initially Configuring Labels in Oracle Solaris
- Hardening labeled file systems – Further Hardening Labeled File Systems
- Backing up and archiving labeled file systems – Maintaining Labeled File Systems
- Testing labels – Viewing and Testing Sample Label Encodings Files

For more examples, see Example - Label Encodings File With Reused Compartment Bits and Chapter 6, Labeling Processes for Data Loss Protection in *Securing Users and Processes in Oracle Solaris 11.4*.

Overall Process for Configuring Labeling

Because Oracle Solaris does not ship sensitive data, all files after installation are at the same label. You should apply labels to your files that contain sensitive information, such as financial data and personnel data.

Perform the following tasks to configure labeled file systems for sensitive data:

1. Install the `file_labeling` package

   This package is not part of a group installation. For the procedure, see How to Install Labels in Oracle Solaris.

2. Determine the coverage of your label policy

   For considerations, see Customizing a Label Policy.

3. Create the label policy

   You set the default clearance for SMF services and customize a label encodings file. Oracle Solaris provides two sample encodings files. You can copy and modify one of these files, or create an encodings file from scratch.

4. Create labeled file systems

   An upper bound label is dynamically computed for each labeled file system. Whenever a file is upgraded, the new label is combined with the current upper bound. A labeled file system retains its upper bound even if all labeled files are reset or removed.

5. Assign labels, called clearances, to users whose clearances should differ from the default

   Administrators assign higher clearances to the few users who have the authority to access labeled files. On a system with labeled files, only a user whose clearance dominates files in the labeled file system can view or modify those dominated files. The administrator might also assign a lower clearance to guest users.

6. Authorize selected users to upgrade or downgrade files.

   By default, only the `root` role can change the label of a file. The Object Label Management profile grants both upgrade and downgrade rights. You can also authorize users or roles to only upgrade or only downgrade information.

7. Configure the auditing of labeled files

   Actions on sensitive files need to be monitored. For most file systems, file-read audit events are not preselected because they add many unimportant events to the audit trail. However, for labeled files, file-read events can be important. Options to the audit service enable you to preselect file-read audit events for labeled files only, thus auditing actions that are important with respect to labeled files but are not important for unlabeled files. Additionally, the `file_labeling` package includes the `/usr/demo/tsol` script that you can use to display daily audit records for local files. To protect the audit trail from snooping, you can create a labeled file system for the audit trail.

8. Reboot to start user processes and SMF services at the new clearances.

9. Test the configuration.

After configuring labeled file systems, you can harden the configuration, as described in About Hardening Labeled File Systems.

1. Create labeled file systems for audit records and for core files.
2. Create a zone for labeled file systems.
3. Create zone login accounts for users who can access the labeled files in the zone.
4. Make the zone immutable.

Initially Configuring Labels in Oracle Solaris

The procedures in this section install and configure a customized label policy. They include assigning labels to users and file systems. To harden this initial setup, see About Hardening Labeled File Systems.

How to Install Labels in Oracle Solaris

You must be the initial user or an administrator with the Software Installation rights profile. The `root` role has all of these rights. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

1. **Install the `pkg:/system/file_labeling` package.**

   ```
   # pkg install file_labeling
   ```

2. **Verify that the `labeld:clearance` service is enabled.**

   ```
   $ svcs labeld:clearance
   STATE          STIME    FMRI
   online         Nov_18   svc:/system/labeld:clearance
   ```

3. **View the current label policy by using the `labelcfg list` command.**

   To view the policy details, use the `labelcfg info` command. For more information, see the `labelcfg`(8) man page. For sample output, see Viewing and Testing Sample Label Encodings Files.

How to Configure Your Label Policy

Complete a label policy assessment. To determine which labels to create, see Configuring Labels on an Oracle Solaris System.

You must be assigned the Object Label Management rights profile or be in the `root` role. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

Defining a label policy is the first step in data loss protection. Later you will assign labels to file systems, and assign selected users a clearance that is higher than the default to view sensitive files.

This procedure uses the following configuration parameters:

- Encodings file = `site-enc`
- Minimum label (Lower bound of user labels) = Public
- Next higher classification = Confidential
- Confidential label hierarchy = Confidential Internal Use Only, Confidential Restricted, Confidential Highly Restricted
- Clearance (Upper bound of user labels) = Confidential Internal Use Only

1. **As `root`, assign the `ADMIN_LOW` clearance as the default clearance for all SMF services.**

   If you are using the `account-policy` service, use the first option. For more information, see `account-policy`(8S) man page.

   - Modify the `login_policy/clearance` security attribute in SMF.

     Follow the How to Set Account Locking for All Logins in *Securing Users and Processes in Oracle Solaris 11.4* procedure, and substitute `login_policy/clearance` for the property in the procedure.

- DEPRECATED: Comment out the original line in the `policy.conf` file and add the `ADMIN_LOW` clearance.

```
# pfedit /etc/security/policy.conf
...
## Highest label at which SMF services run by default.
## For services that must run at a higher label, set a higher clearance
## on their start and restart methods.
#CLEARANCE=ADMIN_HIGH
CLEARANCE=ADMIN_LOW
...
```

2. **Create an encodings file.**

   You can modify the `label_encodings.compliance` or `label_encodings.default` files in the `/etc/security/tsol` directory or create a new encodings file. The following command creates an encodings file from scratch.

```
# labelcfg -e /etc/security/tsol/site-enc
labelcfg:site-enc>
```

3. **Title the label policy.**

```
labelcfg:site-enc> set title="Name Label Policy"
```

4. **Define the labels you will use at your site to protect data.**

   Start with the lowest classification, which is typically the `Public` classification.

```
labelcfg:site-enc> add classification="Public"
labelcfg:Public> set shortname="P"
labelcfg:Public> end
```

   Because public information is public throughout the organization, this label does not require compartments.

5. **Define the next higher classification.**

```
labelcfg:site-enc> add classification="Confidential"
labelcfg:Confidential> set shortname="Conf"
labelcfg:Confidential> end
```

   Add compartments to this classification to indicate levels of confidentiality from company-internal to very restricted.

6. **Create the lowest Confidential label by defining the classification's first compartment.**

```
labelcfg:site-enc> add compartment="Internal Use Only"
labelcfg:Internal Use Only> set minclass="Confidential"
labelcfg:Internal Use Only> end
```

   `minclass` indicates that this compartment cannot be used by the `Public` classification.

7. **Define the next higher label.**

   This label is higher because its compartment bits include the `Internal Use Only` compartment bits.

```
labelcfg:site-enc> add compartment="Restricted"
labelcfg:Restricted> set minclass="Confidential"
labelcfg:Restricted> set subcompartments="Internal Use Only"
labelcfg:Restricted> end
```

8. **Define the next higher label and set Restricted as its subcompartment.**

```
labelcfg:site-enc> add compartment="Highly Restricted"
labelcfg:Highly Restricted> set minclass="Confidential"
labelcfg:Highly Restricted> set subcompartments=Restricted
labelcfg:Highly Restricted> end
```

9. **Define the `min_label` value.**

```
labelcfg:site-enc> set min_label=Public
```

Choose a label that is suitable for the organization, such as `Public`. This label is the lower bound for all processes.

10. **Define the clearance and commit the label policy.**

```
labelcfg:site-enc> set clearance="Confidential Internal Use Only"
labelcfg:site-enc> commit
```

This label is the default clearance for all user processes. Only users to whom you explicitly assign a higher label can access sensitive files.

11. **Display the details of your label hierarchy.**

```
labelcfg:site-enc> info
title=Organization's Label Policy
classification=Public
    level=1
classification=Confidential
    level=2
compartment=Highly Restricted
    bit=2
    subcompartments="Restricted"
    minclass=Confidential
compartment=Restricted
    bit=1
    minclass=Confidential
compartment=Internal Use Only
    bit=0
    minclass=Confidential
min_label=Public
clearance=Confidential Internal Use Only
labelcfg:site-enc> exit
```

12. **Save your work into a flat file.**

The `export` subcommand produces output that can be used as input to the `labelcfg` command to create the exported label policy. In this example, the administrator saves the file to a secure directory.

```
# labelcfg export -f /opt/adminfiles/site-enc-export1
```

Next Steps

If you have disjoint labels to define, you can do so now. For an example, see Example - Label Encodings File With Reused Compartment Bits and the `labelcfg`(8) man page.

How to Assign a Label to a File System

Create an encodings file. You must have logged out and logged back in. You also must be a user who can assume the `root` role. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

To create a labeled file system, you enable the `multilevel` ZFS property. This action can be performed at any time during the lifetime of a ZFS dataset.

1. **Verify that your label policy is in effect.**

```
$ labelcfg list
list-of-labels
$ labelcfg info clearance
clearance
$ plabel
clearance
```

   The *clearance* value returned by these two commands should be identical. If the values differ, you did not commit the value of `clearance` when you edited the encodings file or you have not logged out and logged back in.

2. **Assume the `root` role.**

```
$ su - root
Password:
#
```

3. **Modify or create the ZFS datasets that will contain sensitive, labeled files.**

   - To modify an existing file system and set a label on the mount point:

```
# zfs set -o multilevel=on -o rpool/existing-fs
# setlabel "label" /existing-fs-mountpoint
```

     For example, to label the `/export/home` directory:

```
# zfs set -o multilevel=on -o rpool/export/home
# setlabel "Conf - Internal Use Only" /export/home
```

   - To create a labeled file system, mount it, and set a label on the mount point:

> **Tip:**
>
> For additional protection, encrypt every new multilevel file system.

```
# zfs create -o multilevel=on -o encryption=on rpool/labeled-fs
# zfs set =/mountpoint rpool/labeled-fs
# setlabel "label" /mountpoint
```

     For example, you could label a directory that contains files for company-wide distribution.

```
# zfs create -o multilevel=on -o encryption=on rpool/ftp-files
# zfs set =/ftpsource rpool/ftp-files
# setlabel "Conf - Internal Use Only" /ftpsource
```

4. **Verify that the file system is labeled.**

```
# getlabel /mountpoint
label
```

5. **Share the file system over NFS as a labeled file system.**

   If you do not share a labeled file system with the `share.nfs.labeled=on` option, the files whose labels are higher than `ADMIN_LOW` cannot be accessed.

> **Tip:**
>
> To minimize the risk of identity spoofing, specify an NFS security option with the labeled option. See the nfssec(7) man page.

```
# zfs share -o nfs=on -o share.nfs.labeled=on -o share.nfs.sec=krb5 rpool/labeled-fs
```

6. **View the upper bound of the file system.**

   The value of the `mlslabel` property is the upper bound of the file system and cannot be lowered.

```
# zfs get mlslabel
NAME                              PROPERTY  VALUE                  SOURCE
...
rpool/VARSHARE/zones              mlslabel  none                       -
rpool/dump                        mlslabel  -                          -
rpool/export                      mlslabel  none                       -
rpool/export/home                 mlslabel  Conf - Internal Use Only   -
```

   If higher-labeled files are added, the upper bound is raised to the label of the higher files. A labeled file system retains its label even if all labeled files are reset or removed.

7. **Assign clearances that are higher than the default clearance to trusted users and trusted roles.**

```
# usermod -K clearance="higher-than-default-clearance" trusted-user1
# rolemod -K clearance="higher-than-default-clearance" trusted-role1
```

> **Tip:**
>
> To enable a user to run a command at a clearance higher than the user's assigned clearance, see Enabling Access to Labeled Files in *Securing Users and Processes in Oracle Solaris 11.4* and How to Assign Clearances to Users in *Securing Users and Processes in Oracle Solaris 11.4*.

8. **Assign clearances that are lower than the default clearance to guest users.**

```
# usermod -K clearance=Public guest
```

9. **Configure the auditing of sensitive files by enabling the `labeled-only` audit policy, then set the appropriate audit flags.**

   This policy enables you to audit file-read events and set the audit flags for labeled files.

```
# auditconfig -setpolicy +labeled-only
# auditconfig -setflags fr,fw,fm,dc,fd,ex,lo
```

   When you enable the `fr` audit class when the `labeled-only` policy is in effect, only labeled files are audited for file read. Regular files are not.

Example 3-1 Finding Files of a Specified Label

The following script finds all files of a specified label.

```
#!/bin/sh
# Find all files whose label matches $1
```

```
zfs list -Ho multilevel,mounted,mountpoint -t filesystem -r rpool|\
while read multilevel mounted mountpt;do
    if [ $multilevel == on -a $mounted == yes ];then
        for file in $(find $mountpt -print); do
            label=$(getlabel $file 2>/dev/null|cut -d: -f2|\
                grep -i "$1" 2>/dev/null)
            if [[ -n $label ]]; then
                echo $file
                echo '\t'$label
            fi
        done
    fi
done
```

Further Hardening Labeled File Systems

Although applying labels can prevent inadvertent access by privileged users, the label configuration can be compromised by modifying the underlying configuration, loading untrusted software, or modifying the kernel. To improve the robustness of the label policy, you can isolate labeled file systems in zones, use immutable zones, and store your audit records and core dumps in a labeled file system.

How to Enforce a Fixed Configuration for a Labeled File System

You must assume the `root` role. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

This procedure helps prevent malicious or inadvertent modification of the system configuration.

1. **Add the labeled file system to the global zone, then make the global zone immutable.**

   ```
   # zonecfg -z global
   zonecfg:global> add dataset
   zonecfg:global:dataset> set name=rpool/labeled-fs
   zonecfg:global:dataset> end
   zonecfg:global> set file-mac-profile=fixed-configuration
   zonecfg:global> exit
   ```

   The fixed configuration also prevents modifications to the SMF method and `sysconfig` group properties. The client process must be running at the `ADMIN_HIGH` clearance or using the Trusted Path to modify the configuration. Also, access to the kernel by using the `mdb -k` command requires an `ADMIN_HIGH` process. The `mdb -K` command can succeed only by using the Trusted Path. For more information, see the `mdb`(1) man page. For information about Trusted Path and zone configuration, see Administering an Immutable Zone by Using the Trusted Path Domain in *Creating and Using Oracle Solaris Zones*, and the `zlogin`(1) and `zonecfg`(8) man pages.

2. **To further protect the kernel, enable verified boot by using the Oracle Integrated Lights Out Manager (ILOM).**

   For kernel zones, use the `zonecfg` command. For more information about verified boot, see Using Verified Boot in *Securing Systems and Attached Devices in Oracle Solaris 11.4*.

Example 3-2 Making an Immutable Global Zone That Contains a Labeled File System Immutable

In this example, the administrator specifies that three labeled file systems on a system cannot be reconfigured except through the Trusted Path.

```
# zonecfg -z global
zonecfg:global> add dataset
zonecfg:global:dataset> set name=rpool/Internal
zonecfg:global:dataset> set name=rpool/Restricted
zonecfg:global:dataset> set name=rpool/HighlyRestricted
zonecfg:global:dataset> end
zonecfg:global> set file-mac-profile=fixed-configuration
zonecfg:global> exit
```

How to Isolate a Labeled File System in a Zone

You have enabled a label encodings file that includes the Confidential - Restricted label.

Although access to sensitive data can be restricted to users and roles based on their clearances, a label policy does not prevent cleared users or roles from copying labeled data to unlabeled directories. One way to prevent such data loss is to enable Trusted Extensions. However, you can also use standard Oracle Solaris Zones to provide an additional layer of protection.

1. **Create a labeled file system.**

   ```
   # zfs create -o multilevel=on -o encryption=on \
   -o mountpoint=/mountpoint rpool/mountpoint
   # chmod 777  /mountpoint
   # setlabel "Confidential - Restricted" /mountpoint
   ```

2. **Configure and install a zone to import */mountpoint*.**

   a. **Make the zone a solaris zone and remove the network connection.**

      ```
      # zonecfg -z zonename
      zonecfg> create -b
      zonecfg> set brand=solaris
      zonecfg> remove anet
      ```

   b. **Authorize a user to log in to the zone.**

      ```
      zonecfg> add admin
      zonecfg> set user=username
      zonecfg> set auths=login
      zonecfg> end
      ```

      When authorized users log in to the zone, their clearance is raised to the value specified in the zone's `user_attr` file.

   c. **Mount the labeled file as a loopback file system with read-write permissions.**

      ```
      zonecfg> add fs
      zonecfg> set dir=/mountpoint
      zonecfg> set special=/mountpoint
      zonecfg> set type=lofs
      zonecfg> add options rw
      zonecfg> end
      zonecfg> exit
      ```

   d. **Install the zone.**

```
# zoneadm -z zonename install
```

Although users might not be authorized to set individual file labels, all files in the imported file systems would be automatically labeled based on their containing directories. Also, although users cannot be prevented from copying files into unlabeled file systems, unlabeled data cannot leak outside the zone. The labeled data is available outside of the zone to users and roles with sufficient clearance.

3. **In the zone context, assign a higher clearance to the authorized user.**

```
# zlogin zonename
# usermod -K clearance="specific-higher-clearance" username
```

The user can now log in to the zone and run at the higher clearance. In the global zone, the user's clearance is the default.

```
username@global$ pfbash ; plabel
default-user-clearance
```

4. **Log in as the user to the zone where the user has a higher clearance, open a profile shell. and confirm the higher clearance.**

```
username@global$ zlogin -l username
zonename
username@zonename$ pfbash ; plabel
specific-higher-clearance
```

Example 3-3 Isolating Labeled File Systems in a Zone

This example describes how to create a labeled zone where authorized users can work on `Confidential - Restricted` files.

1. The administrator creates a labeled file system that can contain files at different labels and a zone to mount it. In the mounting zone, the administrator authorizes users to log in, adds the labeled file system, and then installs the zone.

```
# zfs create -o multilevel=on -o encryption=on -o mountpoint=/multi-r rpool/
multi-r
# chmod 777  /multi-r
# setlabel "Confidential - Restricted" /multi-r

# zonecfg -z restricted
zonecfg> create -b
zonecfg> set brand=solaris
zonecfg> remove anet
zonecfg> add admin
zonecfg> set user=user1
zonecfg> set auths=login
zonecfg> end
zonecfg> add fs
zonecfg> set dir=/multi-r
zonecfg> set special=/multi-r
zonecfg> set type=lofs
zonecfg> add options rw
zonecfg> end
zonecfg> exit
# zoneadm -z restricted install
```

2. The administrator assigns the user a higher clearance in the zone context.

```
# zlogin restricted
# usermod -K clearance="Confidential - Restricted" user1
```

3. The administrator tests the user's clearance in the zone context.

```
user1@global$ pfbash ; plabel
Confidential - Internal
user1@global$ zlogin -l user1 restricted
user1@restricted$ pfbash ; plabel
Confidential -  Restricted
```

User `user1` has a higher clearance in the `restricted` zone.

4. After halting the zone, the administrator gives other trusted users access to the `restricted` zone, then boots the zone.

```
# zoneadm -z restricted shutdown
# zonecfg -z restricted
zonecfg> set admin
zonecfg> set user=user2
zonecfg> set auths=login
zonecfg> set user=user3
zonecfg> set auths=login
zonecfg> commit
zonecfg> end
zonecfg> exit
# zoneadm -z restricted boot
```

How to Create a Labeled Audit Trail

You must be in the `root` role.

You create a file system at the highest label, `ADMIN_HIGH`, for the audit trail. All audit events, labeled and not labeled, are then recorded and stored at that label.

1. **Create a file system at `ADMIN_HIGH` for the audit files.**

```
# zfs create -o multilevel=on -o encryption=on rpool/VARSHARE/audit_high
# setlabel ADMIN_HIGH /var/audit_high
```

2. **Add the `audit_high` directory to the list of `audit_binfile` plugins.**

```
# auditconfig -setplugin audit_binfile active "p_dir=/var/audit_high"
```

3. **Run the auditing process at that label.**

```
# svccfg -s auditd
> setprop start/clearance = astring: ADMIN_HIGH
> exit
```

4. **Read the audit service changes into the kernel and restart the service.**

```
# audit -t
# audit -s
```

5. **Create a Labeled Audit Review rights profile and assign it to the users who review audit records.**

   a. **Use the Audit Review profile as the template.**

```
# profiles -p "Audit Review"
profiles:Audit Review> set name="Labeled Audit Review"
profiles:Labeled Audit Review> set desc="Review Labeled Audit Trail"
profiles:Labeled Audit Review> select cmd=/usr/sbin/auditreduce
profiles:Labeled Audit Review:auditreduce> set clearance="ADMIN_HIGH"
profiles:Labeled Audit Review:auditreduce> end
profiles:Labeled Audit Review> select cmd=/usr/sbin/praudit
profiles:Labeled Audit Review:praudit> set clearance="ADMIN_HIGH"
profiles:Labeled Audit Review:praudit> end
```

```
profiles:Labeled Audit Review> commit
profiles:Labeled Audit Review> exit
```

The Labeled Audit Review profile inherits the existing security attributes of the selected commands. The commands retain their assigned privileges and EUIDs.

b. **Verify that the commands are running at the ADMIN_HIGH clearance and retain any security attributes from the original rights profile.**

```
# profiles -p "Labeled Audit Review" "select cmd=/usr/sbin/auditreduce ; info; end;"
    id=/usr/sbin/praudit
    euid=0
    clearance=ADMIN_HIGH
```

c. **Assign the rights profile to users who can review the audit trail by typing one of the following commands:**

```
# usermod -K profiles+="Labeled Audit Review" user-who-reviews-audit-trail

# usermod -K auth_profiles+="Labeled Audit Review" user-who-reviews-audit-trail
```

The commands in the Labeled Audit Review rights profile will run at the `ADMIN_HIGH` label when the user runs the commands in a profile shell, as in `pfexec praudit`. The clearance of the assigned user does not change but the command processes run at the label specified in the profile.

Maintaining Labeled File Systems

After configuring labeled file systems and their users, you maintain the systems by monitoring audit logs and archiving the file systems. Periodically, you need to update the users who can access sensitive files. You can also store an export of the label policy.

> **Note:**
>
> To transfer or archive a labeled file system, your clearance must dominate the value of the `mlslabel` property of the file system.

- Transfer files to a new labeled file system by running the `tar` command with the `-T` option.

  In the following example, *fromdir* is the root of the existing file system and *todir* is the root of the new file system.

  ```
  $ pfbash
  $ cd fromdir; tar -cTf - . | (cd todir; tar xTp -)
  ```

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

- Archive the file systems by using the `zfs send` and `archiveadm` commands.

  These commands preserve the labels of the files. For more information, see the `zfs`(8) and `archiveadm`(8) man pages.

```
$ pfexec zfs send -r labeled-filesystem

$ pfexec archiveadm labeled-filesystem
```

- Export and store the commands that re-create your encodings file.

  This file can be imported to create your encodings file on a test system, for example.

  ```
  # labelcfg -f enc-file-commands
  ```

  For example, to save the committed encodings file to an administrative directory:

  ```
  # labelcfg -f /opt/adminfiles/site-enc-commands
  ```

Viewing and Testing Sample Label Encodings Files

The examples in this section illustrate how to view and use the label policies that Oracle Solaris provides. The encodings files are simple. The default encodings file illustrates a three-level label hierarchy. The compliance encodings file illustrates disjoint labels. The simplest way to test a label policy is to commit the policy, then create a multilevel ZFS dataset and a few users with clearances that enable them to create and view files in the dataset.

Testing Labeling by Using the Default Encodings File

1. To test labels, you must set the `clearance` value to a user label in the default encodings file.

   The following commands list the available labels, set a new clearance value, and display the new clearance.

   ```
   # cp label_encodings.default label_encodings.default.orig
   # labelcfg list
   "Confidential - Highly Restricted"
   "Confidential - Restricted"
   "Confidential - Internal"
   Public
   # labelcfg 'set clearance="Confidential - Internal"'
   # labelcfg info clearance
   clearance=Confidential - Internal
   ```

   > **Note:**
   >
   > The clearance value is typed within double quotes because it contains spaces. However, the shell interprets the double quotes and then removes them. When you surround the subcommand with single quotes, the shell removes the single quotes and leaves the double quotes.

2. Create a labeled file system, mount it, and enable DAC access to any user.

   ```
   # zfs create -o multilevel=on -o encryption=on rpool/defaultenc
   # zfs set mountpoint=/defaultenc rpool/defaultenc
   # cd / ; cd rpool
   # chmod 777 defaultenc
   ```

3. Create test users at different clearances. Users who are created without a clearance inherit the default, Confidential - Internal.

```
# useradd -m /export/home -K clearance="Confidential - Highly Restricted"
high1
# useradd -m /export/home -K clearance="Confidential - Restricted" rest1
# useradd -m /export/home test1
```

4. Reboot, then test.

   For various items to test, see How to Verify User Access to Labeled Files in *Securing Users and Processes in Oracle Solaris 11.4*.

5. After the testing is complete, you can delete the labeled dataset, delete the users with high clearances, and enable the default label encodings file.

```
# zfs destroy rpoot/defaultenc
# userdel -r high1 ; usermod -r rest1 ; usermod -r test1
# labelcfg -e /etc/security/tsol/label_encodings.default.orig commit
```

Example 3-4 Customizing a Test Label Policy

In this example, you modify the existing template with the name of your organization. This example calls the organization `ExampleCo`.

```
# cd /etc/security/tsol
# cp label_encodings.default label_encodings.exampleco
# labelcfg -e label_encodings.exampleco
labelcfg:label_encodings.exampleco> set title="Data Protection Policy for
ExampleCo"
labelcfg:label_encodings.exampleco> select classification="Confidential -"
labelcfg:Confidential -> set shortname="Conf ExampleCo -"
labelcfg:Confidential -> end
labelcfg:label_encodings.exampleco> set clearance="Conf ExampleCo - Internal"
labelcfg:label_encodings.exampleco> commit
labelcfg:label_encodings.exampleco> list
"Conf ExampleCo - Highly Restricted"
"Conf ExampleCo - Restricted"
"Conf ExampleCo - Internal"
Public
labelcfg:label_encodings.exampleco> info clearance
clearance=Conf ExampleCo - Internal
labelcfg:label_encodings.exampleco> exit
```

After you commit this label policy, regular users at login would be operating at the Conf ExampleCo - Internal label. Only users whom you configure with an explicit higher clearance can access sensitive files labeled as Conf ExampleCo - Restricted or Conf ExampleCo - Highly Restricted.

Testing Labeling by Using the Compliance Encodings File

The sample compliance encodings file contains disjoint labels. Disjoint labels can prevent users from seeing department-private information. In the sample `label_encodings.compliance` file, the Health Records and Payment Data departments are disjoint. This policy isolates payment data from health records, both of which are highly restricted information. The policy is enabled by the `commit` command.

1. Commit then view the label encodings file.

```
# cd /etc/security/tsol
# cp label_encodings.compliance label_encodings.compliance.orig
# labelcfg -e label_encodings.compliance commit
# labelcfg info
```

```
title=Sample Data Protection Policy
classification=Public
    level=1
classification=Confidential
    level=2
compartment=Highly Restricted
    subcompartments="Payment Data,Health Records"
    minclass=Confidential
compartment=Payment Data
    bit=0
    subcompartments="Internal Use Only"
    conflicts="Health Records"
    minclass=Confidential
compartment=Health Records
    bit=1
    subcompartments="Internal Use Only"
    conflicts="Payment Data"
    minclass=Confidential
compartment=Internal Use Only
    bit=2
    minclass=Confidential
min_label=Public
clearance=Confidential Internal Use Only
```

2. List the available labels.

```
# labelcfg list
"Confidential Highly Restricted"
"Confidential Payment Data"
"Confidential Health Records"
"Confidential Internal Use Only"
Public
```

3. Create a labeled file system for payment data and health records.

   Add a few payment files and health files, correctly labeled.

4. Create users with different clearances.

   For example, assign the Confidential Highly Restricted clearance to a user who can access everything, the Confidential Payment Data clearance to a user who can handle only payment data, and the Confidential Health Records to a user who can handle only health records. A user with the Confidential Internal Use Only clearance should not be able to see any payment or health information.

5. Reboot, then test.

Example - Label Encodings File With Reused Compartment Bits

The number of compartments that can be defined in a label encodings file is much greater than the number of compartment bits. In Oracle Solaris, the number of available compartment bits is 256, but many thousands of compartments can be created from these bits. Two compartment properties make this possible: `subcompartments` and `conflicts`.

The `subcompartments` property acts like an `include` statement in C. It specifies that the bits of an existing compartment are included in the current compartment. You can define hierarchies of subcompartments to create arbitrary levels of nesting. For example, the following `labelcfg` subcommands create three compartments. The Internal compartment gets bit 0, the Restricted compartment gets bits 0 and 1, and Engineering gets bits 0, 1, and 2.

```
 add compartment=Internal
    set bit=0
    end
 add compartment=Restricted
    set bit=1
    set subcompartments=Internal
    end
 add compartment=Engineering
    set bit=2
    set subcompartments=Restricted
    end
```

If you do not specify a bit value, the next available bit is used. You can use the `clear bit` subcommand to prevent the assignment of a unique bit value. The `clear bit` subcommand is useful when creating an alias for a combination of subcompartments. In this example, the `All BUs` compartment is an alias for all the subcompartments and its bit is cleared.

The `conflicts` property specifies the compartments that are mutually exclusive with the current compartment. Labels that contain conflicting compartments cannot be applied to files or clearances. The `list` subcommand only shows valid compartment combinations.

> **Tip:**
>
> Although compartments typically have unique bit values, you can assign the same bit values to conflicting compartments because conflicting compartments cannot be combined into a valid label.

In this example, multiple business units in a corporation are each assigned one unique bit, and then share the remaining bits for their own projects or departments. The six business units are each assigned unique bits, 2 through 7, and share bits 8 and 9 because the business units are exclusive. The use of 7 bits creates over twenty-five distinct labels.

The bit assignments are as follows:

- 2 - Engineering (8 - Software, 9 - Hardware)
- 3 - Operations (8 - Information Technology, 9 - Maintenance)
- 4 - Human Resources (8 - Benefits, 9 - Personal Information)
- 5 - Legal (8 - Patents, 9 - Compliance)
- 6 - Finance (8 - Payroll, 9 - Accounts)
- 7 - Mergers and Acquisitions (8 - Robots, 9 - Widgets)

This example shows the export file of this encodings file.

```
$ labelcfg -e corporate_encodings info
set title="Corporate Example's Information Protection Policy"
 add classification=Public
    set level=1
    end
 add classification="Confidential -"
    set level=2
    end
```

```
 add compartment=Internal
    set bit=0
    set minclass="Confidential -"
    end
 add compartment=Restricted
    set bit=1
    set subcompartments="Internal"
    end
 add compartment=Engineering
    set bit=2
    set subcompartments="Restricted"
    set prefix="Business Units:"
    end
 add compartment=Operations
    set bit=3
    set subcompartments="Restricted"
    set prefix="Business Units:"
    end
add compartment="Human Resources"
    set shortname=HR
    set bit=4
    set subcompartments="Restricted"
    set prefix="Business Units:"
    end
 add compartment=Legal
    set bit=5
    set subcompartments="Restricted"
    set prefix="Business Units:"
    end
 add compartment=Finance
    set bit=6
    set subcompartments="Restricted"
    set prefix="Business Units:"
    end
 add compartment="Mergers and Acquisitions"
    set shortname=M&A
    set bit=7
    set subcompartments="Restricted"
    set prefix="Business Units:"
    end
 add compartment="All BUs"
    clear bit
    set subcompartments="Mergers and
Acquisitions,Legal,Operations,Finance,Engineering,Human Resources"
    set prefix="Business Units:"
    end
 add compartment=Software
    set bit=8
    set conflicts="All BUs"
    set subcompartments="Engineering"
    set prefix="Example Engineering:"
    end
 add compartment=Hardware
    set bit=9
    set subcompartments="Engineering"
    set conflicts="All BUs"
    set prefix="Example Engineering:"
    end
 add compartment="Information Technology"
    set shortname=IT
    set bit=8
```

```
   set subcompartments="Operations"
   set conflicts="All BUs"
   set prefix="Example Operations:"
   end
add compartment=Maintenance
   set bit=9
   set subcompartments="Operations"
   set conflicts="All BUs"
   set prefix="Example Operations:"
   end
add compartment=Patents
   set bit=8
   set subcompartments="Legal"
   set conflicts="All BUs"
   set prefix="Example Legal:"
   end
add compartment=Compliance
   set bit=9
   set subcompartments="Legal"
   set conflicts="All BUs"
   set prefix="Example Legal:"
   end
add compartment=Robots
   set bit=8
   set subcompartments="Mergers and Acquisitions"
   set conflicts="All BUs"
   set prefix="Example M&A:"
   end
add compartment=Widgets
   set bit=9
   set subcompartments="Mergers and Acquisitions"
   set conflicts="All BUs"
   set prefix="Example M&A:"
   end
add compartment=Benefits
   set bit=8
   set subcompartments="Human Resources"
   set conflicts="All BUs"
   set prefix="Example HR:"
   end
add compartment="Personal Information"
   set bit=9
   set subcompartments="Human Resources"
   set conflicts="All BUs"
   set prefix="Example HR:"
   end
add compartment=Payroll
   set bit=8
   set subcompartments="Finance"
   set conflicts="All BUs"
   set prefix="Example Finance:"
   end
add compartment=Accounts
   set bit=9
   set subcompartments="Finance"
   set conflicts="All BUs"
   set prefix="Example Finance:"
   end
add compartment="Highly Restricted"
   clear bit
   set subcompartments="All Bus,Hardware,Software"
```

```
   end
select compartment="Mergers and Acquisitions"
   set conflicts="All BUs"
   end
select compartment=Legal
   set conflicts="All BUs"
   end
select compartment=Operations
   set conflicts="All BUs"
   end
select compartment=Finance
   set conflicts="All BUs"
   end
select compartment="Human Resources"
   set conflicts="All BUs"
   end
select classification=Public
   set valid=""
   end
select classification="Confidential -"
   set invalid=""
   end
set min_label=Public
set clearance="Confidential - Internal"
```

The resulting encodings file creates 22 compartments by using only 10 bits (0-9). The remaining 246 bits could be shared to create unique hierarchies within each business unit. This example shows the corresponding list of valid labels.

```
$ labelcfg -e corporate_encodings list
 "Confidential - Highly Restricted"
 "Confidential - Business Units: All BUs"
 "Confidential - Example Engineering: Software/Hardware"
 "Confidential - Example Engineering: Software"
 "Confidential - Example Engineering: Hardware"
 "Confidential - Business Units: Engineering"
 "Confidential - Example Operations: IT/Maintenance"
 "Confidential - Example Operations: IT"
 "Confidential - Example Operations: Maintenance"
 "Confidential - Business Units: Operations"
 "Confidential - Example HR: Benefits/Personal Information"
 "Confidential - Example HR: Benefits"
 "Confidential - Example HR: Personal Information"
 "Confidential - Business Units: HR"
 "Confidential - Example Legal: Patents/Compliance"
 "Confidential - Example Legal: Patents"
 "Confidential - Example Legal: Compliance"
 "Confidential - Business Units: Legal"
 "Confidential - Example Finance: Payroll/Accounts"
 "Confidential - Example Finance: Payroll"
 "Confidential - Example Finance: Accounts"
 "Confidential - Business Units: Finance"
 "Confidential - Example M&A: Robots/Widgets"
 "Confidential - Example M&A: Robots"
 "Confidential - Example M&A: Widgets"
 "Confidential - Business Units: M&A"
 "Confidential - Restricted"
 "Confidential - Internal"
 Public
```

Label Man Pages

The following man pages support labeling in Oracle Solaris. The description includes links to examples or explanations of these features.

Label Man Page
Purpose and Links to Additional Information

`atohexlabel`(8)
Converts a human-readable label to its internal text equivalent.
For an example, see How to Obtain the Hexadecimal Equivalent for a Label in *Trusted Extensions Configuration and Administration*.

`blcompare`(3TSOL)
Compares binary labels.

`blminmax`(3TSOL)
Determines the bound of two labels.

`chk_encodings`(8)
Checks the label encodings file syntax.
For examples, see How to Debug a label_encodings File in *Trusted Extensions Label Administration* and How to Check and Install Your Label Encodings File in *Trusted Extensions Configuration and Administration*.

`fgetlabel`(2)
Gets the file's label

`getlabel`(1)
Displays the label of the selected files or directories.
For an example, see How to Display the Labels of Mounted Files in *Trusted Extensions Configuration and Administration*.

`getlabel`(2)
Gets the label of a file

`getplabel`(3TSOL)
Gets the label of a process

`getuserrange`(3TSOL)
Gets the label range of a user

`hextoalabel`(8)
Converts an internal text label to its human-readable equivalent
For an example, see How to Obtain a Readable Label From Its Hexadecimal Form in *Trusted Extensions Configuration and Administration*.

`labelcfg`(8)
Configures labels can modify the `label_encodings` file

`label_encodings`(5)
Describes the label encodings file

`label_to_str`(3TSOL)
Converts labels to human-readable strings

`labels`(7)
Describes label attributes

`m_label`(3TSOL)
Allocates and frees resources for a new label

`plabel`(1)
Gets the label of a process

`setlabel`(1)
Relabels the selected item. Requires the `solaris.label.file.downgrade` or `solaris.label.file.upgrade` authorization. These authorizations are in the Object Label Management rights profile.

`str_to_label`(3TSOL)
Parses human-readable strings to a label

`updatehome`(1)
Updates the home directory copy and link files for the current label
See How to Configure Startup Files for Users in Trusted Extensions in *Trusted Extensions Configuration and Administration*.

4 Verifying File Integrity by Using BART

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.

Compliance assessments can also help you track differences in critical files over time. For more information, see *Oracle Solaris 11.4 Compliance Guide*.

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.

> **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. To create a manifest, use the `bart create` command, as described in How to Create a Control Manifest.

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

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 and the `bart_rules`(5) man page. To create a rules file, see How to Customize a BART Report by Using a Rules File.

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
Create a custom BART manifest.	Generates a list of information about specific files that are installed on a system.	How to Customize a Manifest
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 How to Compare Manifests From Different Systems
(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 How to Customize a BART Report by Using a Rules File

BART Security Considerations

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 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.

How to Create a Control Manifest

You must assume the `root` role. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

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. To understand the format conventions, see Explanation of the BART Manifest Format.

> **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.

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 Explanation of the BART Manifest Format.

3. **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 4-1 Explanation of the BART Manifest Format

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

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

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

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

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

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

How to Customize a Manifest

You must assume the `root` role. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

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.

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. **Save the manifest in a protected directory for future use.**

   For an example, see Step 3 in How to Create a Control Manifest.

> **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

You must assume the `root` role. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

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

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

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

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

   For an example, see Step 3 in How to Create a Control Manifest.

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 4-2 Tracking File Changes for the Same System Over Time

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

- Create a control manifest.

```
# cd /var/adm/logs/manifests
# bart create -R /etc > system1.control.090713
! Version 1.1
! HASH SHA256
! Saturday, September 07, 2013 (11:11:17)
```

```
# Format:
#fname D size mode acl dirmtime uid gid
#fname P size mode acl mtime uid gid
#fname S size mode acl mtime uid gid
#fname F size mode acl mtime uid gid contents
#fname L size mode acl lnmtime uid gid dest
#fname B size mode acl mtime uid gid devnode
#fname C size mode acl mtime uid gid devnode
/.cpr_config F 2236 100644 owner@:read_data/write_data/append_data/read_xattr/wr
ite_xattr/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchr
onize:allow,group@:read_data/read_xattr/read_attributes/read_acl/synchronize:all
ow,everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
4e271c59 0 0 3ebc418eb5be3729ffe7e54053be2d33ee884205502c81ae9689cd8cca5b0090
/.login F 1429 100644 owner@:read_data/write_data/append_data/read_xattr/write_x
attr/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize
:allow,group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow,ev
eryone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
4bf9d6d7 0 3 ff6251a473a53de68ce8b4036d0f569838cff107caf1dd9fd04701c48f09242e
.
.
.
```

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

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

- Compare the manifests.

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

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

How to Compare Manifests From Different Systems

You must assume the `root` role. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

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.

1. **Create a control manifest.**

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

   For the options, see the bart(8) man page.

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

   For an example, see Step 3 in How to Create a Control Manifest.

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

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

4. **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 4-3 Identifying a Suspect File in the `/usr/bin` Directory

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

- Create a control manifest.

```
# bart create -R /usr/bin > control-manifest.090713
! Version 1.1
! HASH SHA256
! Saturday, September 07, 2013 (11:11:17)
# Format:
#fname D size mode acl dirmtime uid gid
#fname P size mode acl mtime uid gid
#fname S size mode acl mtime uid gid
#fname F size mode acl mtime uid gid contents
#fname L size mode acl lnmtime uid gid dest
#fname B size mode acl mtime uid gid devnode
#fname C size mode acl mtime uid gid devnode
/2to3 F 105 100555 owner@:read_data/read_xattr/write_xattr/execute/
read_attribut
es/write_attributes/read_acl/write_acl/write_owner/
synchronize:allow,group@:read
_data/read_xattr/execute/read_attributes/read_acl/
synchronize:allow,everyone@:re
ad_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow
4bf9d261 0
2 154de3e7bdfd0d57a074c9fae0896a9e2e04bebfe5e872d273b063319e57f334
/7z F 509220 100555 owner@:read_data/read_xattr/write_xattr/execute/
read_attribu
tes/write_attributes/read_acl/write_acl/write_owner/
```

```
synchronize:allow,group@:rea
d_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow,everyone@:r
ead_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow 4dadc48a 0
2 3ecd418eb5be3729ffe7e54053be2d33ee884205502c81ae9689cd8cca5b0090
...
```

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

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

- Copy the manifests to the same location.

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

- Compare the manifests.

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

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

How to Customize a BART Report by Specifying File Attributes

You must assume the `root` role. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

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

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

You must assume the `root` role. For more information, see Using Your Assigned Administrative Rights in *Securing Users and Processes in Oracle Solaris 11.4*.

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.

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. **Save the manifest in a protected directory for future use.**

   For an example, see Step 3 in How to Create a Control Manifest.

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

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

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

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

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

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

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

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

- Compare the manifests by using the same rules file.

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

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

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

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

BART Manifests, Rules Files, and Reports

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

BART Manifest File Format

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

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`(5) 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 4-5 Rules File Format

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

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

> **Note:**
>
> All directives are read in order. Later directives can override earlier directives.

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

BART Rules File Attributes

The `CHECK` and `IGNORE` statements define which file attributes to track or ignore. The metadata that begins each manifest lists the attribute *keywords* per file type. See Explanation of the BART Manifest Format.

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`(5) 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.

For more information, see the `bart`(8) man page.

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.

classification

The hierarchical component of a clearance or a label. A classification indicates a hierarchical level of security, for example, `RESTRICTED` or `PUBLIC`.

compartment

A nonhierarchical component of a label that is used with the classification component to form a clearance or a label. A compartment represents a collection of information, such as would be used by an engineering department or a multidisciplinary project team.

clearance

The upper limit of the set of labels at which a user can work. The lower limit is the minimum label that is assigned by the security administrator.

label

A security identifier that is assigned to an object. The label is based on the level at which the information in that object should be protected. Labels are defined in the label_encodings file.

label_encodings file

The label configuration file that defines the label hierarchy, the default user clearance, and other aspects of labels.

minimum label

The lower bound of a user's labels and the lower bound of the system's labels. The minimum label is the label of the user's processes at login. The sensitivity label that is specified in the

minimum label field by the security administrator in the `label_encodings` file sets the lower bound for the system.

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`(7) 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

Symbols

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z