man pages section 4: File Formats

Adobe PostScript ™

011029@2471

Contents

Preface

Overview

A man page is provided for both the naive user and the sophisticated user who is familiar with the Trusted Solaris operating environment and is in need of online information. A man page is intended to answer concisely the question "What does it do?" The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Trusted Solaris Reference Manual

In the AnswerBook2™ and online man command forms of the man pages, all man pages are available:

- Trusted Solaris man pages that are unique for the Trusted Solaris environment
- SunOS 5.8 man pages that have been changed in the Trusted Solaris environment
- SunOS 5.8 man pages that remain unchanged.

The printed manual, the *Trusted Solaris 8 Reference Manual* contains:

- Man pages that have been added to the SunOS operating system by the Trusted Solaris environment
- Man pages that originated in SunOS 5.8, but have been modified in the Trusted Solaris environment to handle security requirements.

Users of printed manuals need both manuals in order to have a full set of man pages, since the *SunOS 5.8 Reference Manual* contains the common man pages that are not modified in the Trusted Solaris environment.

Man Page Sections

The following contains a brief description of each section in the man pages and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.

- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.

- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.

- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume.

- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.

- Section 5 contains miscellaneous documentation such as character set tables.

- Section 6 contains available games and demos.

- Section 7 describes various special files that refer to specific hardware peripherals, and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.

- Section 9 provides reference information needed to write device drivers in the kernel operating systems environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).

- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer may include in a device driver.

- Section 9F describes the kernel functions available for use by device drivers.

- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the intro pages for more information and detail about each section, and man(1) for more information about man pages in general.

NAME
   This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS
   This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and

arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ]   The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . .   Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, ' "filename...".

| Separator. Only one of the arguments separated by this character can be specified at a time.

{ }   Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

PROTOCOL
This section occurs only in subsection 3R to indicate the protocol description file.

DESCRIPTION
This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.

IOCTL
This section appears on pages in Section 7 only. Only the device class which supplies appropriate parameters to the ioctl (2) system call is called ioctl and generates its own heading. ioctl calls for a specific device are listed alphabetically (on the man page for that specific device). ioctl calls are used for a particular class of devices all of which have an io ending, such as mtio(7I)

OPTIONS
This secton lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

OPERANDS
This section lists the command operands and describes how they affect the actions of the command.

OUTPUT
This section describes the output – standard output, standard error, or output files – generated by the command.

RETURN VALUES
If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged

paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.

ERRORS

On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE

This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality:

- Commands
- Modifiers
- Variables
- Expressions
- Input Grammar

EXAMPLES

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be root, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.

ENVIRONMENT VARIABLES

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

EXIT STATUS

This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.

FILES

This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

ATTRIBUTES

This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See `attributes`(5) for more information.

SUMMARY OF TRUSTED SOLARIS CHANGES
  This section describes changes to a Solaris item by Trusted Solaris software. It is present in man pages that have been modified from Solaris software.

SEE ALSO
  This section lists references to other man pages, in-house documentation and outside publications. The references are divided into two sections, so that users of printed manuals can easily locate a man page in its appropriate printed manual.

DIAGNOSTICS
  This section lists diagnostic messages with a brief explanation of the condition causing the error.

WARNINGS
  This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.

NOTES
  This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

BUGS
  This section describes known bugs and, wherever possible, suggests workarounds.

Introduction

NAME	Intro – introduction to file formats
DESCRIPTION	This section outlines the formats of various files. The C structure declarations for the file formats are given where applicable. Usually, the headers containing these structure declarations can be found in the directories /usr/include or /usr/include/sys. For inclusion in C language programs, however, the syntax #include <*filename.h*> or #include <sys/*filename.h*> should be used.

Because the operating system now allows the existence of multiple file system types, there are several instances of multiple manual pages with the same name. These pages all display the name of the FSType to which they pertain, in the form name_*fstype* at the top of the page. For example, fs_ufs(4).

TRUSTED SOLARIS DIFFERENCES

In the Trusted Solaris environment, these configuration files can be:

- Files that are unique to and originate in the Trusted Solaris environment, such as label_encodings(4).

- SunOS 5.8 configuration files that have been modified to work within Trusted Solaris security policy, such as proc(4). Man pages for modified files omit information that is not accurate for how the file is used within the Trusted Solaris environment. Modified man pages also describe new fields or entities.

- SunOS 5.8 files that remain unchanged from the Solaris 8 release, such as timezone(4).

Note – The printed *Trusted Solaris 8 4/01 Reference Manual* includes only those files that have been modified or originate in the Trusted Solaris environment. Printed versions of unchanged SunOS 5.8 man pages are found in the *SunOS 5.8 Reference Manual*. For more information on displaying manual pages, see Trusted Solaris Manual Page Display in Intro(1).

The Trusted Solaris operating environment is a security-enhanced version of the Solaris operating environment, a trusted desktop, the X window system, and the Solaris Management Console™ system administration tools. To preserve security attributes, configuration files are usually not edited using vi or another common editor. Rather, administrative roles edit the files using administrative graphical user interfaces. The GUIs audit all changes and preserve the required owner, group, permissions and sensitivity labels of the files.

RULES FOR INCLUDING LABELS IN A CONFIGURATION FILE

Follow the rules described here when entering labels in configuration files. When entering labels in GUIs, see Rules for the Display and Entering of Labels in Intro(1). When entering labels on the command line in a UNIX shell, follow the rules in Rules for the Display and Entering of Labels in Intro(1M).

Make sure that a program reading a configuration file can tell where the label starts and ends. Where the label is imbedded, as it is in the device_allocate(4) file, the only valid character to begin the label and terminate it is a semicolon (;). Most configuration files do not support label incrementations using plus or minus signs.

Configuration files are generally maintained at a sensitivity label of ADMIN_LOW. However, each site can choose whether to store labels in configuration files as text or as hexadecimal numbers, depending on the site's security policy, and the form used affects the sensitivity label at which the file should be stored. When labels are stored in human-readable form, the files that contain them must be protected at ADMIN_HIGH, so only administrative roles that have the ADMIN_HIGH label in their clearance can view the files. Also, if a file contains a collection of data written by all processes in the system (like the system log, /dev/kmem, and /dev/mem files) that file should be protected at the ADMIN_HIGH sensitivity label.

Note – Labels entered in text form must be quoted.

POLICY FOR SECURITY ATTRIBUTES ON CONFIGURATION FILES

The default user and group for configuration files are root and sys and default permissions are 00644. However, the security administrator should ensure that files that contain sensitivity information other than labels, such as those files that specify which activities are being audited, are not generally readable. These files should have more restrictive permissions, owner and group IDs, and possibly a protective label.

SEE ALSO

Trusted Solaris Administrator's Procedures, *Trusted Solaris Developer's Guide*

Intro(4)

File Formats

NAME	audit_class – Audit class definitions
SYNOPSIS	/etc/security/audit_class
DESCRIPTION	/etc/security/audit_class is a plain text system file that stores class definitions. Programs use the getauclassent(3BSM) routines to access this information.

The fields for each class entry are separated by colons. Each class entry is a bitmap and is separated from each other by a newline.

Each entry in the audit_class file has the form:

mask:*name*:*description*

The fields are defined as follows:

mask	The class mask.
name	The class name.
description	The description of the class.

The classes are user-configurable. Each class is represented as a bit in the class mask which is an unsigned integer. Thus, there are 32 different classes available, plus two meta-classes, all and no.

all represents a conjunction of all allowed classes, and is provided as a shorthand method of specifying all classes.

no is the "invalid" class, and any event mapped solely to this class will not be audited. (Turning auditing on to the all meta class will *not* cause events mapped solely to the no class to be written to the audit trail.)

EXAMPLES **EXAMPLE 1** Sample of an audit_class file

```
0x00000000:no:invalid class
0x00000001:fr:file read
0x00000002:fw:file write
0x00000004:fa:file attribute access
0x00000008:fm:file attribute modify
0x00000010:fc:file create
0x00000020:fd:file delete
0x00000040:cl:file close
0xffffffff:all:all classes
```

SUMMARY OF TRUSTED SOLARIS CHANGES By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration* for how to disable and enable auditing.

FILES /etc/security/audit_class   Audit class definitions.

SEE ALSO getauclassent(3BSM), audit_event(4)

NOTES It is possible to deliberately turn on the `no` class in the kernel, in which case the audit trail will be flooded with records for the audit event `AUE_NULL`.

NAME	audit_control – Control information for system audit daemon
SYNOPSIS	`/etc/security/audit_control`
DESCRIPTION	The `audit_control` file contains audit control information used by `auditd`(1M). Each line consists of a title and a string, separated by a colon. There are no restrictions on the order of lines in the file, although some lines must appear only once. A line beginning with '#' is a comment.

Directory definition lines list the directories to be used when creating audit files, in the order in which they are to be used. The format of a directory line is:

`dir:`*directory-name*

directory-name is where the audit files will be created. Any valid writable directory can be specified.

Unless explicitly told to look elsewhere, the `auditreduce`(1M) command by default looks for the audit trail in all directories named according to the following convention on the server on which the command is run. Therefore, this naming convention is recommended for directories in which audit-trail files are stored:

`/etc/security/audit/`*server*`[.`*number*`]/files`

server is the name of the audit server on which the audit files are stored. The optional *.number* is used when an audit server exports two or more audit partitions. For example, the audit server trustworthy exports `/etc/security/audit/trustworthy` and `/etc/security/audit/trustworthy.1`. For the current host to use both of these partitions, these lines must be added to the local `audit_control` file:

```
dir:/etc/security/audit/trustworthy/files
dir:/etc/security/audit/trustworthy.1/files
```

Audit data may be stored in directories with other names at the discretion of the site. Some sites may want to store each host's audit data in a separate subdirectory. The audit structure used will depend on each individual site. If the defined audit structure differs from `/etc/security/audit/*/files`, `auditreduce` needs to be given the new location of the audit trail explicitly as decribed in `auditreduce`(1M).

The audit threshold line specifies the percentage of free space that must be present in the file system containing the current audit file. The format of the threshold line is:

`minfree:`*percentage*

where *percentage* is indicates the amount of free space required. If free space falls below this threshold, the audit daemon `auditd`(1M) invokes the shell script `audit_warn`(1M). If no threshold is specified, the default is `0%`.

The audit flags line specifies the default system audit value. This value is combined with the user audit value read from audit_user(4) to form the process audit state. The user audit value overrides the system audit value. The format of a flags line is:

flags:*audit-flags*

where *audit-flags* specifies which event classes are to be audited. The character string representation of *audit-flags* contains a series of flag names, each one identifying a single audit class, separated by commas. A name preceded by minus (-) means that the class should be audited for failure only; successful attempts are not audited. A name preceded by plus (+) means that the class should be audited for success only; failing attempts are not audited. Without a prefix, the name indicates that the class is to be audited for both successes and failures. The special string all indicates that all events should be audited: -all indicates that all failed attempts are to be audited; +all, all successful attempts. The prefixes ^, ^-, and ^+ turn off flags specified earlier in the string (^- and ^+ for failing and successful attempts, ^ for both). They are typically used to reset flags.

The non-attributable flags line is similar to the flags line, but this one contain the audit flags that define what classes of events are audited when an action cannot be attributed to a specific user. The format of a naflags line is:

naflags:*audit-flags*

The flags are separated by commas, with no spaces.

The following table lists the predefined audit classes:

```
short name    long name     Short description
no      no_class        Null value for turning off event preselection
fr      Read of data, open for reading, etc.
fw      Write of data, open for writing, etc.
fa      Access of object attributes: stat, pathconf, etc.
fm      Change of object attributes: chown, flock, etc.
fc      Creation of object
fd      Deletion of object
cl      close(2) system call
pc      Process operations
nt      Network events: bind, connect, accept, etc.
ip      System V IPC operations
na      Non-attributable events
ad      Administrative actions: mount, exportfs, etc.
lo      Login and logout events
ap      Application auditing
ax      server
ss      system state
as      system-wide administration
aa      administration
ao      administration
ps      start/stop
pm      modify
io      ioctl(2) system call
```

```
fn     fcntl(2) system call
ot     Everything else
all    All flags set
```

Note that the classes are configurable; see `audit_class`(4).

EXAMPLES   **EXAMPLE 1** Sample /etc/security/audit_control file

Here is a sample `/etc/security/audit_control` file for the machine eggplant:

```
dir: /etc/security/jedgar/eggplant
dir: /etc/security/jedgar.aux/eggplant
#
# Last-ditch audit file system when jedgar fills up.
#
dir: /etc/security/global/eggplant
minfree: 20
flags: lo,ad,-all,^-fm
naflags: lo,ad
```

This identifies server `jedgar` with two file systems normally used for audit data, another server `global` used only when `jedgar` fills up or breaks, and specifies that the warning script is run when the file systems are 80% filled. It also specifies that all logins, administrative operations are to be audited (whether or not they succeed), and that failures of all types except failures to access object attributes are to be audited.

FILES
```
/etc/security/audit_control
/etc/security/audit_warn
/etc/security/audit/*/*/*
/etc/security/audit_user          Audit files
```

SUMMARY OF TRUSTED SOLARIS CHANGES   By default, the machine halts when audit files run out of disk space. The Trusted Solaris environment adds programming interfaces, audit tokens, audit classes, and audit events.

By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration* for how to disable and enable auditing.

Trusted Solaris 8 4/01 Reference Manual   `audit`(1M), `audit_warn`(1M), `auditd`(1M), `audit`(2), `getfauditflags`(3BSM), `audit.log`(4), `audit_class`(4), `audit_user`(4), *Trusted Solaris Audit Administration*

NAME | audit_data – Current information on audit daemon

SYNOPSIS | `/etc/security/audit_data`

DESCRIPTION | The `audit_data` file contains information about the audit daemon. The file contains the process ID of the audit daemon, and the pathname of the current audit log file. The format of the file is:

pid:*pathname*

Where *pid* is the process ID for the audit daemon, and *pathname* is the full pathname for the current audit log file.

EXAMPLES | **EXAMPLE 1** A sample audit_data file.

`64:/etc/security/audit/iedgar/19990506081249.19990506230945.eggplant`

SUMMARY OF TRUSTED SOLARIS CHANGES | By default, auditing is enabled in the Trusted Solaris environment. The `audit_data` file is protected at `ADMIN_HIGH`.

See *Trusted Solaris Audit Administration* for how to disable and enable auditing.

FILES | `/etc/security/audit_data`   Current information on audit daemon.

Trusted Solaris 8 4/01 Reference Manual | audit(1M), `auditd`(1M), audit(2), `audit.log`(4)

Trusted Solaris Audit Administration

NAME	audit_event – Audit event definition and class mapping file
SYNOPSIS	/etc/security/audit_event

DESCRIPTION /etc/security/audit_event is a plain text system file that stores event definitions and specifies the event-to-class mappings. Programs use the getauevent(3BSM) routines to access this information.

The fields for each event entry are separated by colons. Each event is separated from the next by a newline.

Each entry in the audit_event file has the form:

number:*name*:*description*: *flags*

The fields are defined as follows:

number	The event number.
name	The event name.
description	The description of the event.
flags	Flags specifying classes to which the event is mapped.

EXAMPLES **EXAMPLE 1** Some audit_event file entries

```
7:AUE_EXEC:exec(2):ps
79:AUE_OPEN_WTC:open(2) - write,creat,trunc:fc,fd,fw
6152:AUE_login:login - local:lo
6153:AUE_logout:logout:lo
6154:AUE_telnet:login - telnet:lo
6155:AUE_rlogin:login - rlogin:lo
```

FILES /etc/security/audit_event   Audit event definition and class mapping file.

SUMMARY OF TRUSTED SOLARIS CHANGES The Trusted Solaris environment adds audit events to the audit_event file, and remaps some audit events to audit classes that do not exist in the Solaris environment. Also, auditing is enabled by default. See *Trusted Solaris Audit Administration* for how to disable and enable auditing.

Trusted Solaris 8 4/01 Reference Manual getauevent(3BSM), audit_control(4)

Trusted Solaris Audit Administration

NAME	audit.log – audit trail file	

SYNOPSIS `#include <bsm/audit.h>`

`#include <bsm/audit_record.h>`

DESCRIPTION `audit.log` files are the depository for audit records stored locally or on an audit server. These files are kept in directories named in the file `audit_control`(4). They are named to reflect the time they are created and are, when possible, renamed to reflect the time they are closed as well. The name takes the form

yyyymmddhhmmss`.not_terminated.`*hostname*

when open or if the `auditd`(1M) terminated ungracefully, and the form

yyyymmddhhmmss`.`*yyyymmddhhmmss*`.`*hostname*

when properly closed. *yyyy* is the year, *mm* the month, *dd* day in the month, *hh* hour in the day, *mm* minute in the hour, and *ss* second in the minute. All fields are of fixed width.

The `audit.log` file begins with a standalone *file token* and typically ends with one also. The beginning file token records the pathname of the previous audit file, while the ending file token records the pathname of the next audit file. If the file name is `NULL` the appropriate path was unavailable.

The `audit.log` files contains audit records. Each audit record is made up of *audit tokens*. Each record contains a header token followed by various data tokens. Depending on the audit policy in place by `auditon`(2), optional other tokens such as trailers or sequences may be included.

The tokens are defined as follows:

The `file` token consists of:

```
token ID               1 byte
seconds of time        4 bytes
milliseconds of time   4 bytes
file name length       2 bytes
file pathname          N bytes + 1 terminating NULL byte
```

The `header` token consists of:

```
token ID               1 byte
record byte count       4 bytes
version #              1 byte    [2]
event type             2 bytes
event modifier         2 bytes
seconds of time        4 bytes/8 bytes (32-bit/64-bit value)
milliseconds of time   4 bytes/8 bytes (32-bit/64-bit value)
```

The expanded `header` token consists of:

```
toke ID                1 byte
record byte count      4 bytes
version #              1 byte     [2]
event type            2 bytes
event modifier        2 bytes
address type/length   4 bytes
machine address       4 bytes/16 bytes (IPv4/IPv6 address)
seconds of time       4 bytes/8 bytes (32/64-bits)
milliseconds of time  4 bytes/8 bytes (32/64-bits)
```

The trailer token consists of:

```
token ID               1 byte
trailer magic number   2 bytes
record byte count      4 bytes
```

The arbitrary data token is defined:

```
token ID               1 byte
how to print           1 byte
basic unit             1 byte
unit count             1 byte
data items             (depends on basic unit)
```

The in_addr token consists of:

```
token ID               1 byte
internet address       4 bytes
```

The expanded in_addr token consists of:

```
token ID               1 byte
IP address type/length 4 bytes
IP address             16 bytes
```

The ip token consists of:

```
token ID               1 byte
version and ihl        1 byte
type of service        1 byte
length                 2 bytes
id                     2 bytes
offset                 2 bytes
ttl                    1 byte
protocol               1 byte
checksum               2 bytes
source address         4 bytes
destination address    4 bytes
```

The expanded ip token consists of:

```
token ID               1 byte
version and ihl        1 byte
type of service        1 byte
length                 2 bytes
id                     2 bytes
offset                 2 bytes
ttl                    1 byte
```

```
 protocol               1 byte
 checksum               2 bytes
 address type/type      4 bytes
 source address         4 bytes/16 bytes (IPv4/IPv6 address)
 address type/length    4 bytes
 destination address    4 bytes/16 bytes (IPv4/IPv6 address)
```

The `iport` token consists of:

```
token ID               1 byte
port IP address        2 bytes
```

The `opaque` token consists of:

```
 token ID          char
 size              short
 data              char, size chars
```

The `path` token consists of:

```
 token ID          1 byte
 path length       2 bytes
 path              N bytes + 1 terminating NULL byte
```

The `process` token consists of:

```
 token ID             1 byte
 audit ID             4 bytes
 effective user ID    4 bytes
 effective group ID   4 bytes
 real user ID         4 bytes
 real group ID        4 bytes
 process ID           4 bytes
 session ID           4 bytes
 terminal ID
   port ID            4 bytes/8 bytes (32-bit/64-bit value)
   machine address    4 bytes
```

The expanded `process` token consists of:

```
 token ID             1 byte
 audit ID             4 bytes
 effective user ID    4 bytes
 effective group ID   4 bytes
 real user ID         4 bytes
 real group ID        4 bytes
 process ID           4 bytes
 session ID           4 bytes
 terminal ID
   port ID            4 bytes/8 bytes (32-bit/64-bit value)
   address type/length  4 bytes
   machine address    16 bytes
```

The `return` token consists of:

```
 token ID          1 byte
 error number      1 byte
 return value      4 bytes/8 bytes (32-bit/64-bit value)
```

The `subject` token consists of:

```
token ID                1 byte
audit ID                4 bytes
effective user ID       4 bytes
effective group ID      4 bytes
real user ID            4 bytes
real group ID           4 bytes
process ID              4 bytes
session ID              4 bytes
terminal ID
  port ID               4 bytes/8 bytes (32-bit/64-bit value)
  machine address       4 bytes
```

The expanded `subject` token consists of:

```
token ID                1 byte
audit ID                4 bytes
effective user ID       4 bytes
effective group ID      4 bytes
real user ID            4 bytes
real group ID           4 bytes
process ID              4 bytes
session ID              4 bytes
terminal ID
  port ID               4 bytes/8 bytes (32-bit/64-bit value)
  address type/length   4 bytes
  machine address      16 bytes
```

The `System V IPC` token consists of:

```
token ID                1 byte
object ID type          1 byte
object ID               4 bytes
```

The `text` token consists of:

```
token ID                1 byte
text length             2 bytes
text                    N bytes + 1 terminating NULL byte
```

The `attribute` token consists of:

```
token ID                1 byte
file access mode        4 bytes
owner user ID           4 bytes
owner group ID          4 bytes
file system ID          4 bytes
node ID                 8 bytes
device                  4 bytes/8 bytes (32-bit/64-bit)
```

The `groups` token consists of:

```
token ID                1 byte
number groups           2 bytes
group list              N * 4 bytes
```

The `System V IPC permission` token consists of:

```
token ID              1 byte
owner user ID         4 bytes
owner group ID        4 bytes
creator user ID       4 bytes
creator group ID      4 bytes
access mode           4 bytes
slot sequence #       4 bytes
key                   4 bytes
```

The `arg` token consists of:

```
token ID              1 byte
argument #            1 byte
argument value        4 bytes/8 bytes (32-bit/64-bit value)
text length           2 bytes
text                  N bytes + 1 terminating NULL byte
```

The `exec_args` token consists of:

```
token ID              1 byte
count                 4 bytes
text                  count null-terminated string(s)
```

The `exec_env` token consists of:

```
token ID              1 byte
count                 4 bytes
text                  count null-terminated string(s)
```

The `exit` token consists of:

```
token ID              1 byte
status                4 bytes
return value          4 bytes
```

The `socket` token consists of:

```
token ID              1 byte
socket type           2 bytes
remote port           2 bytes
remote Internet address 4 bytes
```

The expanded `socket` token consists of:

```
token ID              1 byte
socket type           2 bytes
local port            2 bytes
address type/length   4 bytes
local Internet address  4 bytes/16 bytes (IPv4/IPv6 address)
remote port           4 bytes
address type/length   4 bytes
remote Internet address 4 bytes/16 bytes (IPv4/IPv6 address)
```

The `seq` token consists of:

```
token ID              1 byte
sequence number       4 bytes
```

The `acl` token consists of

```
token ID              char
num of entries        int
   (following three fields repeated num times)
object type           int
uid/gid               int
permissions           short
```

The `clearance` token consists of

```
token ID              char
CLEARANCE
label ID              char
pad character         char
classification        short
compartments          8 ints
```

The `host` token consists of

```
token ID                  char
local Internet address    long
```

The `liaison` token consists of

```
token ID              char
liaison ID            int
```

The `priv` token consists of

```
token ID              char
succ/fail             char
priv. used            int
```

The `privilege` token consists of

```
token ID              char
type of set           char
priv. set             4 ints
```

The `slabel` token consists of

```
token ID              char
SLABEL
pad character         char
classification        short
compartments          8 ints
```

The `uauth` token consists of:

```
token ID              1 byte
text length           2 bytes
text                  N bytes + 1 terminating NULL byte
```

The `xatom` token consists of

```
token ID              char
string length         short
atom string           string length bytes
```

The `xcolormap` token consists of

```
token ID              char
XID                   int
creator UID           int
```

The `xcursor` token consists of

```
token ID              char
XID                   int
creator UID           int
```

The `xfont` token consists of

```
token ID              char
XID                   int
creator UID           int
```

The `xgc` token consists of

```
token ID              char
XID                   int
creator UID           int
```

The `xpixmap` token consists of

```
token ID              char
XID                   int
creator UID           int
```

The `xproperty` token consists of

```
token ID              char
XID                   int
creator UID           int
string length         short
string                string length bytes
```

The `xselect` token consists of

```
token ID              char
property length       short
property string       property length bytes
prop. type len.       short
```

```
                     prop type              prop. type len. bytes
                     data length            short
                     window data            data length bytes
```

The `xwindow` token consists of

```
    XID          int
    creator UID  int
```

SUMMARY OF TRUSTED SOLARIS CHANGES

These audit tokens have been added to the Trusted Solaris auditing module: `acl`, `clearance`, `host`, `liaison`, `priv`, `privilege`, `slabel`, `uauth`, `xatom`, `xcolormap`, `xcursor`, `xfont`, `xgc`, `xpixmap`, `xproperty`, `xselect`, and `xwindow`. Trusted Solaris auditing also uses the `auditwrite`(3TSOL) function instead of `au_to_*()` function calls to create audit tokens.

By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration* for how to disable and enable auditing.

Trusted Solaris 8 4/01 Reference Manual

`audit`(1M), `auditd`(1M), `audit`(2), `auditon`(2), `auditwrite`(3TSOL), `audit_control`(4)

SunOS 5.8 Reference Manual

`au_to`(3BSM)

NAME | audit_user – Per-user auditing data file

SYNOPSIS | `/etc/security/audit_user`

DESCRIPTION | `audit_user` is an access-restricted plain text system file that stores per-user auditing preselection data. The `audit_user` file can be used with other authorization sources, including the NIS+ `audit_user` table . Programs use the `getauusernam`(3BSM) to access this information.

The search order for `audit_user` sources follows the order specified for `passwd`(4) in the `nsswitch.conf`(4) file. No entry should be made for `audit_user`.

The fields for each user entry are separated by colons (`:`).. Each user is separated from the next by a newline. `audit_user` does not have general read permission.

Each entry in the `audit_user` database has the form:

username:always-audit-flags:never-audit-flags

The fields are defined as follows:

username	The user's login name.
always-audit-flags	Flags specifying event classes to *always* audit.
never-audit-flags	Flags specifying event classes to *never* audit.

For a complete description of the audit flags and how to combine them, see the `audit_control`(4) man page.

EXAMPLES | **EXAMPLE 1** Sample audit_user file.

```
other:lo,ad:io,cl
        freda:lo,ex,+fc,-fr,-fa:io,cl
        ethel:lo,ex,nt:io,cl
```

FILES |

`/etc/nsswitch.conf`	Configuration file for the name service switch
`/etc/security/audit_user`	Per-user auditing data file.
`/etc/passwd`	Per-machine user password file.

SUMMARY OF TRUSTED SOLARIS CHANGES | By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration* for how to disable and enable auditing.

Trusted Solaris 8 4/01 Reference Manual | `getauusernam`(3BSM), `audit_control`(4), `nsswitch.conf`(4)

Trusted Solaris Audit Administration

SunOS 5.8 Reference Manual | `passwd`(4)

NAME	auth_desc – Descriptions of defined authorizations
SYNOPSIS	`#include <tsol/auth.h>` (obsolete)
DESCRIPTION	This man page is obsolete. To see the definitions for authorizations, see the `Authorizations` tool in the `Solaris Management Console`'s `Rights Manager`.

NAME | auth_name – Authorization description database

SYNOPSIS | `/usr/lib/tsol/locale/`*locale*`/auth_name (obsolete)`

DESCRIPTION | The `auth_name` database and `<tsol/auth_name.h>` header file are replaced in Trusted Solaris 8 and later releases with the `auth_attr`(4) database. Programs can use the functions described in the `getauthattr`(3SECDB) man page to get information from the `auth_attr` database. See the *Trusted Solaris Transition Guide* for correspondences between old and new authorization names.

config.privs(4)

NAME	config.privs – List of window privileges that override system checks
SYNOPSIS	`/usr/openwin/server/tsol/config.privs`
DESCRIPTION	`config.privs` contains a list of all window privileges. `config.privs` lists each privilege in plain text, one per line, separated from the next by a new line. Lines preceded by a comment sign (#) are ignored.

Each privilege not preceded by a comment overrides system checks for that privilege. The security administrator can comment out privileges in the list, but cannot add new privileges.

By default, `config.privs` contains all the privileges that are allowed in the file: `win_colormap`, `win_config`, `win_dga`, `win_devices`, `win_fontpath`.

`config.privs` should have a sensitivity label of `ADMIN_LOW` with permission bits `664`, owner `root`, and group `bin`.

ATTRIBUTES See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxwplt

FILES `/usr/openwin/server/tsol/config.privs`
   List of window privileges that override system checks in the Trusted Solaris environment.

Trusted Solaris 8 `priv_desc`(4)
4/01 Reference
Manual `attributes`(5)
SunOS 5.8
Reference Manual

NAME	device_allocate – device allocate information file
SYNOPSIS	`/etc/security/device_allocate`
DESCRIPTION	The `device_allocate` file contains information about allocatable devices. Corresponding entries in `device_maps`(4) list the device special files associated with the allocatable device.

The `device_allocate` file contains information about allocatable devices. Corresponding entries in `device_maps`(4) list the device special files associated with the allocatable device.

This file is normally created using the `mkdevdb`(1M) command, run by the `init.d`(4) scripts during a system's initial bootload or when the system is booted with the `-r` (reconfigure) option. The `mkdevdb` command creates a set of entries for the system's audio and removable media devices.

The preferred method of modifying the `device_allocate` file is to use the Device Administration dialog box of the Device Allocation Manager.

Each device is represented by a one-line entry of the form:

device-name;*device-type*;*attributes*;reserved;*device-authorization*;*device-clean*

where

device-name   is the name used to identify the device for allocations. The allocation name is an arbitrary text string, containing no embedded white space or non-printable characters. Note, however, that the `init.d`(4) scripts assume that the allocation names will not be changed for entries they created using `mkdevdb`(1M). If these entries are renamed, the `init.d` scripts will create new (and possibly conflicting) entries when the system is rebooted with the `-r` option. Also, the `/etc/security/lib/device_clean` script depends on the names of disk devices having the names assigned by `mkdevdb`.

device-type   is the generic device type, used to identify and group together devices of like type. This field is an arbitrary text string, containing no embedded white space or non-printable characters.

attributes   is a colon-separated string of *key=value* pairs.

device-authorization   is a comma-separated list of authorizations. A user must have at least one of these authorizations to allocate the device. In place of the authorization list, this field may contain an `*` to indicate that the device is not allocatable, or an `@` to indicate that no explicit authorization is needed to allocate the device.

	An optional colon (:) plus a second list of authorizations may be used to provide different authorizations for allocations from the trusted path (primarily through the Device Allocation Manager) and for allocations that do not come from the trusted path (primarily by command-line use of the `allocate(1)` command). The syntax for this form of the authorizations field is `tp_auths:nontp_auths`. If a device allocation request comes from the trusted path, the user must have one of the authorizations specified in `tp_auths`. For requests not from the trusted path, the user must have one of the authorizations specified in `nontp_auths`. Either of these may be `*` or `@`.
device-clean	is the path of a device cleaning program to be run any time the device is allocated or deallocated. The cleaning program ensures that all usable data is purged from the physical drive before it is reused.
	The device cleaning program may interact with the user via prompts and responses on `stdout/stdin`.
	An alternate version of the cleaning program for use in a windowing environment may be supplied by using the same path with the suffix `.windowing` appended. The windowing version may use the window system to interact with the user via dialogs.

Lines in `device_allocate` can end with a \ to continue an entry on the next line.

Leading and trailing blanks are allowed in any of the fields.

The recommended method of modifying the `device_allocate` file is through the Add Allocatable Device action and the Device Allocation Manager. A designated administrative role uses the Add Allocatable Device action to add a device with default attributes. The Device Allocation Manager's Configure dialog box is used for modifications to a device. These tools handle the formatting of entries (including translation of plain text sensitivity labels to hex), and audit all changes. They preserve the correct permissions, ownership, and label of the `device_allocate` file.

EXAMPLES

EXAMPLE 1 Sample Device Allocate File

```
# Allow local (trusted path) allocation of audio to any user,
# Disallow all remote (non-trusted path) allocation of audio.

audio; \
     audio; \
     minsl=0x00000000000000000000000000000000000000000\
          00000000000000000000000000000: \
     maxsl=0x7fffffffffffffffffffffffffffffffffffffff\
          fffffffffffffffffffffffffffffff; \
```

EXAMPLE 1 Sample Device Allocate File   *(Continued)*

```
    reserved; \
    @:*; \
    /etc/security/audio_clean_wrapper; \

# Allow tape drive use by users with either the
# solaris.device.allocate or com.xyzcompany.tape authorization.

mag_tape_0; \
    st; \
    minsl=0x0000000000000000000000000000000000000000\
        0000000000000000000000000000: \
    maxsl=0x7fffffffffffffffffffffffffffffffffffffff\
        ffffffffffffffffffffffffffff; \
    reserved; \
    solaris.device.allocate,com.xyzcompany.tape; \
    /etc/security/lib/disk_clean; \

# Allow CD use by anyone at [SECRET] or above.

cdrom_0; \
    sr; \
    minsl=0x00050c0000000000000000000000000000000000\
        0000000000000003fffffffffffff0: \
    maxsl=0x7fffffffffffffffffffffffffffffffffffffff\
        ffffffffffffffffffffffffffff; \
    reserved; \
@; \
/etc/security/lib/disk_clean;
```

FILES   /etc/security/device_allocate   Administrative file defining parameters for
                                                 device allocation.

SUMMARY OF TRUSTED SOLARIS CHANGES

Devices are labeled, and by default require authorization for allocating and
deallocating. The authorization field can optionally specify separate authorizations for
allocations made from the trusted path and allocations not made from the trusted
path. Special entries for framebuffer and printers are used by the window system and
the printing system.

Trusted Solaris 8 4/01 Reference Manual

allocate(1), deallocate(1), list_devices(1), mkdevdb(1M),
device_deallocate(4), device_maps(4)

SunOS 5.8 Reference Manual

auth_attr(4)

NOTES

A special entry for the framebuffer device is used to specify the minimum, and
maximum labels at which users may log in to the workstation. This entry is used by
the window system rather than by the allocate(1) command.

Special entries for printers are used to specify the minimum and maximum labels at
which users may submit print requests for a printer. The device-name field contains
the name of the printer. This entry is used by the printing system rather than by the

device_allocate(4)

allocate(1) command. There need not be a corresponding entry in the
`device_maps` file; if it exists, its contents are ignored by the printing system. Serial
line entries may be similarly specified.

NAME | device_deallocate – Device deallocate file

SYNOPSIS | `/etc/security/device_deallocate`

DESCRIPTION | The `device_deallocate` file can contain device deallocation information for allocatable devices. Its entries parallel those of the `device_allocate`(4) file. An entry for a device has the form:

device-name ; *system-boot* ; *user-logout* ;

A backslash (\) at the end of a line continues the next line as part of the current entry. Leading and trailing blanks are allowed in any of the fields.

device-name　　The name of the device. This must match the name of the device in the `device_allocate`(4) file.

system-boot　　Specifies what to do when the named device is found during system boot in an allocated state. This field may be one of these keywords:

DEALLOCATE　　　　　　Deallocate the device.

NO_ACTION　　　　　　Leave the device in the allocated state.

user-logout　　Specifies what to do when a user logs out from the window system.

The *user-logout* action applies to any form of logout from the window system, whether initiated by the user, an administrator, or the system. This includes logout due to a system shutdown. It does not apply to other types of logouts, such as exiting from an rlogin, telnet or ftp session, or exiting from a role.

The *user-logout* applies to devices that are allocated by the user who is logging out from the window system. It applies regardless of whether the user allocated the device from the window session or by some other means (such as from a telnet session or a cron_job). If the device is allocated by a different user or by a role, it remains allocated.

This field may be one of these keywords:

DEALLOCATE　　　　　　Deallocate the device.

NO_ACTION　　　　　　Leave the device in the allocated state.

If a device does not have an entry in the `device_deallocate` file, the default action is NO_ACTION for both system-boot and user-logout. `device_deallocate` should be at a sensitivity label of ADMIN_LOW with permission bits 644, owner root, and group sys.

NOTES   The preferred method of modifying this file is by use of the Device
Administration function of the Device Allocation Manager.

EXAMPLES   **EXAMPLE 1** Deallocating the st0 device upon boot

st0;DEALLOCATE;NO_ACTION;

This entry causes the st0 device to be deallocated at system boot. No action is taken
at the time of logout from the window system.

EXAMPLE 2 Deallocating the CD-ROM device only upon login session termination

cdrom_0;NO_ACTION;DEALLOCATE;

This entry causes the cdrom_0 device to be deallocated when the user who allocated
it logs out from the window system. It will also be deallocated when the system is
rebooted, since system shutdown forcibly logs out all users, so there is no functional
difference between this entry and cdrom_0;DEALLOCATE;DEALLOCATE.

FILES   /etc/security/device_allocate
  Administrative file defining parameters for device allocation.

/etc/security/device_deallocate
  Administrative file defining parameters for device deallocation.

Trusted Solaris 8   allocate(1), deallocate(1), list_devices(1), remove_allocatable(1M),
4/01 Reference   device_allocate(4)
Manual

NAME | device_maps – maps allocatable devices to device special files

SYNOPSIS | /etc/security/device_maps

DESCRIPTION | The device_maps file maps each allocatable device to the set of device special files that are associated with the device.

This file is normally created using the mkdevdb(1M) command, run by the init.d(4) scripts during a system's initial bootload or when the system is booted with the -r (reconfigure) option. The mkdevdb command creates a set of entries for the system's audio and removable media devices.

The preferred method of modifying the device_maps file is to use the Device Administration dialog box of the Device Allocation Manager.

Each device is represented by a one-line entry of the form:

device-name : *device-type* : *device-list*

where

device-name is the allocation name of the physical device. This must match the name given in the device's device_allocate(4) entry.

device-type is the generic device type. This must match the type given for the device in the device_allocate(4) file.

device-list is a list of device special files under /dev that are associated with the physical drive. This field contains device special file pathnames separated by white spaces, or a list generator enclosed in backquotes (").

Lines in device_maps can end with a \ to continue an entry on the next line. Leading and trailing blanks are allowed in any of the fields.

EXAMPLES | **EXAMPLE 1** Sample device_maps entries

```
# Audio
audio:\
   audio:\
     '/etc/security/lib/audio_devlist'
# CD-ROM drive
cdrom_0:\
   sr:\
     /dev/dsk/c0t2d0s0 /dev/dsk/c0t2d0s1 /dev/dsk/c0t2d0s2 \
     /dev/dsk/c0t2d0s3 /dev/dsk/c0t2d0s4 /dev/dsk/c0t2d0s5 \
     /dev/dsk/c0t2d0s6 /dev/dsk/c0t2d0s7 /dev/rdsk/c0t2d0s0 \
     /dev/rdsk/c0t2d0s1 /dev/rdsk/c0t2d0s2 /dev/rdsk/c0t2d0s3 \
     /dev/rdsk/c0t2d0s4 /dev/rdsk/c0t2d0s5 /dev/rdsk/c0t2d0s6 \
     /dev/rdsk/c0t2d0s7
```

FILES | /etc/security/device_allocate
Administrative file defining parameters for device allocation.

device_maps(4)

/dev
   Directory containing logical device name links to device special files under
   /devices.

/devices
   Directory containing all device special files, named to reflect their system bus
   addresses.

Trusted Solaris 8
4/01 Reference
Manual

allocate(1), deallocate(1), list_devices(1), dminfo(1M), mkdevdb(1M)

NAME | device_policy – device policy file

DESCRIPTION | The security policy for device files can differ from that for regular files and is configured through the `device_policy` database file. Rebooting the system in multiuser mode is required to effect the file's contents. Each entry in the file consists of one or more lines and represents the device policy configuration for one or more device files. A backslash (\) at the end of a line continues the next line as part of the current entry. A pound sign (#) as the first character of a line indicates a comment line, which is ignored. Each entry is of the form:

name:minor_name policy_type=value policy_type=value ...

name is the name of a device driver.

minor_name is the actual name of a minor node, or a string of shell metacharacters that represent several minor nodes. See `sh`(1).

If two or more entries match a device, `devpolicy`(1M) uses the first matching entry. For example, for the following `device_policy` entries, the policy for `/dev/ptyp0` will differ from the policy for other `pty` devices.

```
#
# device_policy file
#
ptc: typ0    data_mac_policy=DR_MAC_EQ,DW_MAC_EQ
#
ptc:*        data_mac_policy=DR_MAC_ANY,DW_MAC_ANY
```

policy_type=value specifies a policy for the device nodes. There are four policy types: `data_mac_policy`, `attr_mac_policy`, `open_priv`, and `str_type`. The policy types and their allowed values are described below.

data_mac_policy type | This policy type specifies what a process's sensitivity label must be to have access to the device. The specified policy is enforced by the `open`(2) and `access`(2) system calls. The value for this type is a comma-separated pair of values: a read-MAC value and a write-MAC value:

The read-MAC values are:

DR_MAC_ANY   Process may have any SL.

DR_MAC_EQ   Process SL must be equal to device SL.

DR_MAC_NEQ   Process SL must not equal device SL.

DR_MAC_NEVER   Device is not read accessible.

DR_MAC_SDOM   Process SL must dominate device SL.

DR_MAC_ODOM   Process SL must be dominated by device SL.

The write-MAC values are:

DW_MAC_ANY	Process may have any SL.
DW_MAC_EQ	Process SL must equal device SL.
DW_MAC_NEQ	Process SL must not equal device SL.
DW_MAC_NEVER	Device is not write accessible.
DW_MAC_SDOM	Process SL must dominate device SL.
DW_MAC_ODOM	Process SL must be dominated by device SL.

The optional read-MAC-modifier or write-MAC-modifier value is:

MOD_AUTO_ALLOC	Automatically allocate the device on behalf of the opening process.
MOD_GETDEVLABEL	Get label directly from device. This is used only for console-related pseudo-devices, such as /dev/console or /dev/syslog.

The default policy is

```
data_mac_policy=DR_MAC_EQ,DW_MAC_EQ
```

attr_mac_policy type   This policy type specifies how to handle access to the device's attributes by the operations acl(2), chmod(2), chown(2), and stat(2). The value for this type is a comma-separated set of values: a read-MAC value, a write-MAC value, and an optional read-MAC modifier:

The read-MAC values are:

DR_MAC_ANY	Process may have any SL.
DR_MAC_EQ	Process SL must equal device SL.
DR_MAC_NEQ	Process SL must not equal device SL.
DR_MAC_NEVER	Device is not read accessible.
DR_MAC_SDOM	Process SL must dominate device SL.
DR_MAC_ODOM	Process SL must be dominated by device SL.

The write-MAC values are:

DW_MAC_ANY	Process may have any SL.
DW_MAC_EQ	Process SL must equal device SL.
DW_MAC_NEQ	Process SL must not equal device SL.
DW_MAC_NEVER	Device is not write accessible.
DW_MAC_SDOM	Process SL must dominate device SL.
DW_MAC_ODOM	Process SL must be dominated by device SL.

The optional read-MAC-modifier value is:

MOD_FABRICATE   Return fabricated device attributes to the reading process. Fabrication is designed for a process that walks down an array of BSD-style pty's until it encounters an accessible pty (indicated by getting device attributes) or the end of the array.

The default policy is:

```
attr_mac_policy=DR_MAC_SDOM,DW_MAC_EQ
```

open_priv type   This policy type specifies a privilege required to open the device. The specified privilege is required in addition to the data MAC policy. Privilege names can be in upper or lower case; or an integer ordinal can be used. For example,

```
open_priv=sys_devices
```

The default policy is:

```
open_priv=none
```

str_type type   The streams type, meaningful only for streams devices, specifies how the kernel streams head should control streams messages. The value can be one of these keywords:

DSTR_LOOP   Loop type stream. Unlabeled streams control messages are allowed. Unlabeled data messages are not allowed.

DSTR_NET   Network type Stream. Unlabeled Stream messages are not allowed.

DSTR_DEV   Device type Stream. Unlabeled Stream messages are allowed.

An example is:

```
str_type=DSTR_NET
```

The default policy is:

```
str_type=STR_DEV
```

ATTRIBUTES   See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu

device_policy(4)

EXAMPLES | **EXAMPLE 1** A complete policy — Sample

mm:kmem   \
data_mac_policy=DR_MAC_EQ,DW_MAC_EQ   \
attr_mac_policy=DR_MAC_SDOM,DW_MAC_EQ
mm:null   \
data_mac_policy=DR_MAC_ANY,DW_MAC_ANY   \
attr_mac_policy=DR_MAC_SDOM,DW_MAC_EQ

FILES | /etc/security/tsol/device_policy
Device policy file.

Trusted Solaris 8 | devpolicy(1M)
4/01 Reference
SunOS 5.8 | sh(1), attributes(5)
Reference Manual

NAME | exec_attr – execution attributes database

SYNOPSIS | `/etc/security/exec_attr`

DESCRIPTION | `/etc/security/exec_attr` is a local database that specifies the execution attributes associated with rights profiles. The `exec_attr` file can be used with other sources for rights profiles, including the `exec_attr` NIS map and NIS+ table. Programs use the `getexecattr(3SECDB)` routines to access this information.

The search order for multiple rights profile sources is specified in the `/etc/nsswitch.conf` file, as described in the `nsswitch.conf(4)` man page. The search order follows the entry for `prof_attr(4)`.

A rights profile is a logical grouping of authorizations, CDE actions, and commands that is interpreted by a profile shell to form a secure execution environment. The shells that interpret profiles are `pfcsh`, `pfksh`, and `pfsh`. See the `pfexec(1)` man page. Each user's account is assigned zero or more profiles in the `user_attr(4)` database file.

Each entry in the `exec_attr` database consists of one line of text containing seven fields separated by colons (`:`). Line continuations using the backslash (`\`) character are permitted. The basic format of each entry is:

name:*policy*:*type*:*res1*:*res2*:*cmdid*:*attr*

name:*policy*:*type*:*res1*:*res2*:*actid;argclass;argtype;argmode;argcount*:*attr*

name	The name of the profile. Profile names are case-sensitive.
policy	The policy that is associated with the profile entry. The only valid policies are `suser` and `tsol`.
type	The type of object defined in the profile. There are two valid types: `cmd` and `act`.
res1	Reserved for future use.
res2	Reserved for future use.
cmdid	A string that uniquely identifies the command described by the profile or an asterisk (`*`) used as a wildcard. *cmdid* is either the full path to the command or a wildcard indicating all commands. You can also use a wildcard with a pathname to indicate all files in a particular directory. To specify arguments, the pathname should point to a shell script written to execute the command with the desired arguments.
actid	A string that uniquely identifies the CDE action described by the profile or an asterisk (`*`) used as a wildcard. If an individual action is specified, there are four additional semicolon-separated fields

used to define an argument for the action. These fields can be
empty but the semicolons are required.

argclass	Specifies the argument class (for example, FILE or SESSION.) Corresponds to ARG_CLASS for CDE actions.
argtype	Specifies the data type for the argument. Corresponds to ARG_TYPE for CDE actions.
argmode	Specifies read or write mode for the argument. Corresponds to ARG_MODE for CDE actions.
argcount	Specifies the number of arguments that the action can accept. Corresponds to ARG_COUNT for CDE actions.

attr   An optional list of semicolon-separated (;) key-value pairs that
describe the security attributes to apply to the object upon
execution. Zero or more keys may be specified. The list of valid
keywords depends on the policy enforced. The following
keywords are valid: `privs`, `clearance`, `label`, `euid`, `uid`,
`egid`, and `gid`.

The `privs` key contains a comma-separated list of privilege
numbers that will be effective when the command or action is run.

The `clearance` key contains the maximum label at which the
process can run.

The `label` key contains the minimum label at which the process
can run.

`euid` and `uid` contain a single user name or a numeric user ID.
Commands designated with `euid` run with the effective UID
indicated, which is similar to setting the setuid bit on an
executable file. Commands designated with `uid` run with both the
real and effective UIDs. Setting `uid` may be more appropriate than
setting the `euid` on privileged shell scripts.

`egid` and `gid` contain a single group name or a numeric group ID.
Commands designated with `egid` run with the effective GID
indicated, which is similar to setting the setgid bit on a file.
Commands designated with `gid` run with both the real and
effective GIDs. Setting `gid` may be more appropriate than setting
`guid` on privileged shell scripts.

EXAMPLES

EXAMPLE 1 Using effective user and group IDs

The following example shows how the `audit` command in the Audit Control profile is specified to execute with an effective user ID of root (`0`) and effective group ID of bin (`3`):

```
Audit Control:suser:cmd:::/etc/init.d/audit:euid=0;egid=3
```

EXAMPLE 2 Applying Privileges to a CDE Action

The following example shows how the Tar action in the Media Backup profile is specified to execute with a set of privileges. (Note that privilege names are mapped to integer values in `/usr/include/sys/tsol/priv_names.h`.)

```
Media Backup:tsol:act:::Tar;*;TAR,MAGTAPE;*;>0:privs=1,4,5,8,10,11,12,19,71;
```

FILES

`/etc/nsswitch.conf`	Configuration file for the name service switch.
`/etc/user_attr`	Local source of extended attributes associated with users and roles.
`/etc/security/exec_attr`	Local source for execution attributes associated with rights profiles.

CAVEATS

When deciding which authorization source to use (see DESCRIPTION), keep in mind that NIS+ provides stronger authentication than NIS.

Because the list of legal keys is likely to expand, any code that parses this database must be written to ignore unknown key-value pairs without error. When any new keywords are created, the names should be prefixed with a unique string, such as the company's stock symbol, to avoid potential naming conflicts.

The following characters are used in describing the database format and must be escaped with a backslash if used as data: colon (`:`), semicolon (`;`), equals (`=`), and backslash (`\`).

SUMMARY OF TRUSTED SOLARIS CHANGES

In the Trusted Solaris environment, the `exec_attr` file contains actions (including four arguments) as well as commands. In addition, both actions and commands can have privileges, clearances, and labels as security attributes.

Trusted Solaris 8 4/01 Reference Manual

`auths`(1), `profiles`(1), `roles`(1), `getauusernam`(3BSM), `getauthattr`(3SECDB), `prof_attr`(4), `priv_desc`(4)

SunOS 5.8 Reference Manual

`makedbm`(1M), `getexecattr`(3SECDB), `getprofattr`(3SECDB), `getuserattr`(3SECDB), `kva_match`(3SECDB)

NAME | logindevperm, fbtab – login-based device permissions

SYNOPSIS | `/etc/logindevperm`

DESCRIPTION | The `/etc/logindevperm` file contains information that is used by `login`(1) and `ttymon`(1M) to change the owner, group, and permissions of devices upon logging into or out of a console device. By default, this file contains lines for the keyboard, mouse, audio, and frame buffer devices.

In the Trusted Solaris environment, `logindevperm` entries are not needed for the keyboard, mouse, and frame buffer devices, because sensitivity labels on these devices prevent access by user processes. Device allocation based on `allocate`(1) is the preferred method of setting device ownership and permissions on other devices, such as audio.

The owner of the devices listed in `/etc/logindevperm` is set to the owner of the console by `login`(1). The group of the devices is set to the owner's group specified in `/etc/passwd`. The permissions are set as specified in `/etc/logindevperm`.

Fields are separated by TAB and/or SPACE characters. Blank lines and comments can appear anywhere in the file; comments start with a hashmark, ' # ', and continue to the end of the line.

The first field specifies the name of a console device (for example, /dev/console). The second field specifies the permissions to which the devices in the *device_list* field (third field) will be set. A *device_list* s a colon-separated list of device names. A device entry that is a directory name and ends with "/*" specifies all entries in the directory (except "." and ".."). For example, "/dev/fbs/*" specifies all frame buffer devices.

Once the devices are owned by the user, their permissions and ownership can be changed using `chmod`(1) and `chown`(1), as with any other user-owned file.

Upon logout the owner and group of these devices will be reset by `ttymon`(1M) to owner `root` and root's group as specified in `/etc/passwd` (typically `other`). The permissions are set as specified in the `/etc/logindevperm` file.

FILES | `/etc/passwd`   File that contains user group information.

SUMMARY OF TRUSTED SOLARIS CHANGES | The use of `logindevperm` is not supported, and the default `/etc/logindevperm` file has all entries commented out.

Trusted Solaris 8 4/01 Reference Manual | `allocate`(1), `chmod`(1), `chown`(1), `login`(1), `device_allocate`(4), `device_deallocate`(4)

SunOS 5.8 Reference Manual | `ttymon`(1M), `passwd`(4)

NOTES | `/etc/logindevperm` provides a superset of the functionality provided by `/etc/fbtab` in SunOS 4.x releases.

NAME | inetd.conf – Internet servers database

SYNOPSIS | `/etc/inet/inetd.conf`

`/etc/inetd.conf`

DESCRIPTION | The `inetd.conf` file contains the list of servers that `inetd`(1M) invokes when it receives an Internet request over a socket. Each server entry is composed of a single line of the form:

service-name endpoint-type protocol wait-status uid server-program \
server-arguments

Fields are separated by either SPACE or TAB characters. A # (number sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search this file.

service-name
: The name of a valid service listed in the `services` file. For RPC services, the value of the *service-name* field consists of the RPC service name or program number, followed by a / (slash) and either a version number or a range of version numbers (for example, `rstatd/2-4`).

endpoint-type
: Can be one of:

`stream`	For a stream socket
`dgram`	For a datagram socket
`raw`	For a raw socket
`seqpacket`	For a sequenced packet socket
`tli`	For all TLI endpoints

protocol
: Must be a recognized protocol listed in the file `/etc/inet/protocols`. For RPC services, the field consists of the string `rpc` followed by a / (slash) and either a * (asterisk), one or more nettypes, one or more netids, or a combination of nettypes and netids. Whatever the value, it is first treated as a nettype. If it is not a valid nettype, then it is treated as a netid. For example, `rpc/*` for an RPC service using all the transports supported by the system (the list can be found in the `/etc/netconfig` file), equivalent to saying `rpc/visible rpc/ticots` for an RPC service using the Connection-Oriented Transport Service.

wait-status
: `nowait` for all but "single-threaded" datagram servers — servers which do not release the socket until a timeout occurs. These must have the status `wait`. Do not configure `udp` services as `nowait`. This will cause a

race condition where the inetd program selects on the socket and the server program reads from the socket. Many server programs will be forked and performance will be severely compromised.

A new option exists for udp servers. The -poly option, is similar to the -wait option except that -poly allows a separate server to be started at each sensitivity label. This option is allowed only for udp servers.

If the server program should inherit the trusted path attribute, the *wait-status* field should include the keyword trusted, separated from other keywords in the field by a comma. If the keyword is not present, the trusted path attribute will not be propagated to the server.

If the server program should inherit audit characteristics from the client, the *wait-status* field should include the keyword setaudit, separated from other keywords in the field by a comma. If the setaudit keyword is present, the audit ID, audit terminal ID, and audit preselection mask of the client will be transferred to the server.

uid   The user ID under which the server should run. This allows servers to run with access privileges other than those for root. If the server should run with the ID of the client making the call to the server, a keyword of CLIENT should be entered in the *uid* field. The CLIENT keyword is allowed only for nowait servers. If the CLIENT keyword is present the user ID, group ID, and supplementary groups of the client will be transferred to the server.

server-program   Either the pathname of a server program to be invoked by inetd to perform the requested service, or the value internal if inetd itself provides the service.

server-arguments   If a server must be invoked with command line arguments, the entire command line (including argument 0) must appear in this field (which consists of all remaining words in the entry). If the server expects inetd to pass it the address of its peer (for compatibility with 4.2BSD executable daemons), then the first argument to the command should be specified as '%A'. No more than five arguments are allowed in this field.

SUMMARY OF TRUSTED SOLARIS CHANGES

The *wait-status* field is extended to allow a `trusted` keyword to specify that the trusted path attribute should be passed to the server by `inetd`. If you want a server to run with the audit characteristics of the client, the *wait-status* field can now contain a keyword of `setaudit`.

If you want a `nowait` server to run with the user ID of the client, the *uid* field can now contain a keyword of `CLIENT`.

The `-poly` option has been added for `udp` servers. The option is similar to the `-wait` option except that `-poly` allows a separate server to be started at each sensitivity label.

FILES

`/etc/netconfig`	Network configuration file
`/etc/inet/protocols`	Internet protocols
`/etc/inet/services`	Internet network services

Trusted Solaris 8 4/01 Reference Manual `in.tftpd(1M)`, `inetd(1M)`

SunOS 5.8 Reference Manual `rlogin(1)`, `rsh(1)`, `services(4)`

NOTES

`/etc/inet/inetd.conf` is the official SVR4 name of the `inetd.conf` file. The symbolic link `/etc/inetd.conf` exists for BSD compatibility.

NAME | inittab – Script for init

DESCRIPTION | The file /etc/inittab controls process dispatching by init. The processes most typically dispatched by init are daemons.

The inittab file is composed of entries that are position dependent and have the following format:

id:*rstate*:*action*:*process*

Each entry is delimited by a newline; however, a backslash (\) preceding a newline indicates a continuation of the entry. Up to 512 characters for each entry are permitted. Comments may be inserted in the *process* field using the convention for comments described in sysh(1M). . There are no limits (other than maximum entry size) imposed on the number of entries in the inittab file. The entry fields are:

id
> One or two characters used to uniquely identify an entry.

rstate
> Define the run level in which this entry is to be processed. Run-levels effectively correspond to a configuration of processes in the system. That is, each process spawned by init is assigned a run level(s) in which it is allowed to exist. The run levels are represented by a number ranging from 0 through 6. For example, if the system is in run level 1, only those entries having a 1 in the *rstate* field are processed.
>
> When init is requested to change run levels, all processes that do not have an entry in the *rstate* field for the target run level are sent the warning signal SIGTERM and allowed a 5-second grace period before being forcibly terminated by the kill signal SIGKILL. The *rstate* field can define multiple run levels for a process by selecting more than one run level in any combination from 0 through 6. If no run level is specified, then the process is assumed to be valid at all run levels 0 through 6.
>
> There are three other values, a, b and c, which can appear in the *rstate* field, even though they are not true run levels. Entries which have these characters in the *rstate* field are processed only when an init or telinit process requests them to be run (regardless of the current run level of the system). See init(1M). These differ from run levels in that init can never enter run level a, b or c. Also, a request for the execution of any of these processes does not change the current run level. Furthermore, a process started by an a, b or c command is not killed when init changes levels. They are killed only if their line in inittab is marked off in the *action* field, their line is deleted entirely from inittab, or init goes into single-user state.

action
> Key words in this field tell init how to treat the process specified in the *process* field. The actions recognized by init are as follows:

respawn
   If the process does not exist, then start the process; do not wait for its
   termination (continue scanning the inittab file), and when the process dies,
   restart the process. If the process currently exists, do nothing and continue
   scanning the inittab file.

wait
   When init enters the run level that matches the entry's *rstate*, start the process
   and wait for its termination. All subsequent reads of the inittab file while
   init is in the same run level cause init to ignore this entry.

once
   When init enters a run level that matches the entry's *rstate*, start the process,
   do not wait for its termination. When it dies, do not restart the process. If init
   enters a new run level and the process is still running from a previous run level
   change, the program is not restarted.

boot
   The entry is to be processed only at init's boot-time read of the inittab file.
   init is to start the process and not wait for its termination; when it dies, it does
   not restart the process. In order for this instruction to be meaningful, the *rstate*
   should be the default or it must match init's run level at boot time. This action
   is useful for an initialization function following a hardware reboot of the system.

bootwait
   The entry is to be processed the first time init goes from single-user to
   multi-user state after the system is booted. (If initdefault is set to 2, the
   process runs right after the boot.) init starts the process, waits for its
   termination and, when it dies, does not restart the process.

powerfail
   Execute the process associated with this entry only when init receives a power
   fail signal, SIGPWR (see signal(3C)).

powerwait
   Execute the process associated with this entry only when init receives a power
   fail signal, SIGPWR, and wait until it terminates before continuing any
   processing of inittab.

off
   If the process associated with this entry is currently running, send the warning
   signal SIGTERM and wait 5 seconds before forcibly terminating the process with
   the kill signal SIGKILL. If the process is nonexistent, ignore the entry.

ondemand
   This instruction is really a synonym for the respawn action. It is functionally
   identical to respawn but is given a different keyword in order to divorce its
   association with run levels. This instruction is used only with the a, b or c
   values described in the *rstate* field.

initdefault

    An entry with this action is scanned only when `init` is initially invoked. `init`
    uses this entry to determine which run level to enter initially. It does this by
    taking the highest run level specified in the *rstate* field and using that as its initial
    state. If the *rstate* field is empty, this is interpreted as `0123456` and `init` will
    enter run level `6`. This will cause the system to loop (it will go to firmware and
    reboot continuously). Additionally, if `init` does not find an `initdefault` entry
    in `inittab`, it requests an initial run level from the user at reboot time.

sysinit

    Entries of this type are executed before `init` tries to access the console (that is,
    before the Console Login: prompt). It is expected that this entry will be used
    only to initialize devices that `init` might try to ask the run level question. These
    entries are executed and `init` waits for their completion before continuing.

process

    Specify a command to be executed. The entire `process` field is prefixed with `exec`
    and passed to a forked `sh` as `sh −c 'exec command'`. For this reason, any legal `sh`
    syntax can appear in the *process* field.

SUMMARY OF
TRUSTED
SOLARIS
CHANGES

The Trusted Solaris environment uses the `sysh` shell.

`init(1M)`, `sysh(1M)`, `exec(2)`, `open(2)`

`who(1)`, `ttymon(1M)`, `signal(3C)`

NAME | label_encodings – label encodings file

SYNOPSIS | /etc/security/tsol/label_encodings

DESCRIPTION | The label_encodings file is a standard encodings file of security labels that are used to control the conversion of human-readable labels into an internal format, the conversion from the internal format to a human-readable canonical form, and the construction of banner pages for printed output. In the Trusted Solaris environment, the label_encodings file is protected at the label admin_high. The file should be edited and checked by the security administrator using the Check Label Encodings action in the System_Admin folder in the Application Manager.

EXTENDED DESCRIPTION | In addition to the required sections of the label encodings file described in *Compartmented Mode Workstation Labeling: Encodings Format*, the Trusted Solaris environment accepts optional local extensions. These extensions provide various translation options and an association between character-coded color names and sensitivity labels.

The optional local extensions section starts with the LOCAL DEFINITIONS: keyword and is followed by zero or more of the following unordered statements:

ADMIN LOW NAME=*name*
> The string *name* is accepted as an alternate name for the ADMIN_LOW label when translating from character-coded to binary form. The string *name* is the string returned when translating the ADMIN_LOW label from binary to character-coded form. If this option is not specified, ADMIN_LOW is used.

> Note that use of this option could lead to interoperability problems with machines which do not have the same alternate name.

ADMIN HIGH NAME=*name*
> The string *name* is accepted as an alternate name for the ADMIN_HIGH label when translating from character-coded form to binary form. The string *name* is the string returned when translating the ADMIN_HIGH label from binary to character-coded form. If this option is not specified, ADMIN_HIGH is used.

> Note that use of this option could lead to interoperability problems with machines which do not have the same alternate name.

DEFAULT LABEL VIEW IS EXTERNAL
> Unless otherwise specified, when an ADMIN_HIGH or ADMIN_LOW binary label is translated to a character-coded label, the character-coded label will be in external form. In external form ADMIN_HIGH is demoted to the maximum label and ADMIN_LOW is promoted to the minimum label. If this option is not specified, the external label view applies.

DEFAULT LABEL VIEW IS INTERNAL
> Unless otherwise specified, when an ADMIN_HIGH or ADMIN_LOW binary label is translated to a character-coded label, the character-coded label will be in internal form. In internal form, ADMIN_HIGH is represented by the string ADMIN_HIGH and

ADMIN_LOW is represented by the string ADMIN_LOW. If this option is not specified, the external label view applies.

DEFAULT FLAGS= *value*
This option represents a default GFI Flags= keyword value to be used if no flags are specified as a parameter to the translation. Caution must be taken when defining a DEFAULT FLAGS= *value* that the appropriate Flags= *value*s have been provided. A non-zero value also implies that label validation during translation from binary to character-coded form is not done. If this option is not specified, the default value is 0 (zero).

FORCED FLAGS= *value*
This option represents a GFI Flags= keyword value to be used in all translations. Caution must be taken when defining a FORCED FLAGS= value that the appropriate Flags= values have been provided. A non-zero value also implies that label validation during translation from binary to character-coded form is not done. If this option is not specified, the default value is 0 (zero).

CLASSIFICATION NAME= *name*
This option specifies the string to be displayed in the Label builder GUI for the title of the Classification names section. Specifying a NULL value for *name* leaves the section without a title. If this option is not specified, the default value is CLASSIFICATION.

COMPARTMENTS NAME= *name*
This option specifies the string to be displayed in the label builder GUI for the title of the *Compartments Word* section. Specifying a NULL value for *name* leaves the section without a title. If this option is not specified, the default value is COMPARTMENTS.

DEFAULT USER SENSITIVITY LABEL= *sensitivity label*
This option specifies the sensitivity label to use as the user's minimum sensitivity label if none is defined for the user in the administrative databases. The default value is the MINIMUM SENSITIVITY LABEL= value from the ACCREDITATION RANGE: section of the label encodings file.

DEFAULT USER CLEARANCE= *clearance*
This option specifies the clearance to use as the user's clearance if none is defined for the user in the administrative databases. The default value is the MINIMUM CLEARANCE= value from the ACCREDITATION RANGE: section of the label encodings file.

The final part of the LOCAL DEFINITIONS: section defines the character-coded color names to be associated with various words, sensitivity labels, or classifications. This section supports the bltocolor(3TSOL) function. It consists of the COLOR NAMES: keyword and is followed by zero or more color-to-label assignments. Each statement has one of the following two syntaxes:

word= *word value*; color= *color value*;

label= *label value*; color= *color value*;

where *color value* is a character–coded color name to be associated with the word *word value*, sensitivity label *label value*, or classification *label value*.

The character–coded color name *color value* for a label is determined by the order of entries in the COLOR NAMES: section that make up the label. If a label contains a word *word value* that is specified in this section, the *color value* of the label is the one associated with the first *word value* specified. If no specified word *word value* is contained in the label, the *color value* is the one associated with an exact match of a *label value*. If there is no exact match, the *color value* is the one associated with the first specified *label value* whose classification matches the classification of the label.

ATTRIBUTES   See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsr

EXAMPLES   **EXAMPLE 1** A Sample LOCAL DEFINITIONS: Section

```
LOCAL DEFINITIONS:
ADMIN LOW NAME= LoLo;   * It is strongly advised not to use this option
ADMIN HIGH NAME= HiHi;  * It is strongly advised not to use this option

DEFAULT LABEL VIEW IS INTERNAL;

DEFAULT FLAGS= 0x4;
FORCED FLAGS= 0;

CLASSIFICATION NAME=;  * No Classification name title
COMPARTMENTS NAME=;    * No Compartments word title

DEFAULT USER SENSITIVITY LABEL= C A;
DEFAULT USER CLEARANCE LABEL= S ABLE;


COLOR NAMES:

label= Admin_Low;    color= Pale Blue;
label= unclassified; color= light grey;
word= Project A;     color= bright blue;
label= c;            color= sea foam green;
label= secret;       color= #ff0000;        * Hexadecimal RGB value
word= Hotel;         color= Lavender;
word= KeLO;          color= red;
label= TS;           color= khaki;
label= TS Elephant;  color= yellow;
label= Admin_High;   color= shocking pink;
```

FILES   /etc/security/tsol/label_encodings
The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system. It is protected at the label admin_high.

DIAGNOSTICS   The following diagnostics are in addition to those found in Appendix A of *Compartmented Mode Workstation Labeling: Encodings Format*:

Admin_High color already assigned as XXX.
A color has already been defined for the ADMIN_HIGH label. Another cannot be defined.

Admin_Low color already assigned as XXX.
A color has already been defined for the ADMIN_LOW label. Another cannot be defined.

Can't allocate NNN bytes for ADMIN HIGH NAME=
The system cannot dynamically allocate the memory it needs to process the ADMIN_HIGH NAME= option.

Can't allocate NNN bytes for ADMIN LOW NAME=
The system cannot dynamically allocate the memory it needs to process the ADMIN_LOW NAME= option.

Can't allocate NNN bytes for CLASSIFICATION NAME=
The system cannot dynamically allocate the memory it needs to process the CLASSIFICATION NAME= option.

Can't allocate NNN bytes for COMPARTMENTS NAME=
The system cannot dynamically allocate the memory it needs to process the COMPARTMENTS NAME= option.

Can't allocate NNN bytes for color name XXX.
The system cannot dynamically allocate the memory it needs to store color name XXX.

Can't allocate NNN bytes for color names table.
The system cannot dynamically allocate the memory it needs to process the COLOR NAMES: section.

Can't allocate NNN bytes for color table entry.
The system cannot dynamically allocate the memory it needs to process a Color Table entry.

Can't allocate NNN bytes for color word entry.
The system cannot dynamically allocate the memory it needs to process a Color Word entry.

Can't allocate NNN bytes for DEFAULT USER.
The system cannot dynamically allocate the memory it needs to process the DEFAULT USER.

`DEFAULT USER CLEARANCE= XXX is not in canonical form. Is YYY what is intended?`
This error occurs if the clearance specified, while understood, is not in canonical form. This additional canonicalization check ensures that no errors are made in specifying the clearance.

`DEFAULT USER SENSITIVITY LABEL= XXX is not in canonical form. Is YYY what is intended?`
This error occurs if a sensitivity label specified, while understood, is not in canonical form. This additional canonicalization check ensures that no errors are made in specifying the sensitivity label.

`Duplicate ADMIN HIGH NAME= ignored.`
More than one `ADMIN HIGH NAME=` option was encountered. All but the first are ignored.

`Duplicate ADMIN LOW NAME= ignored.`
More than one `ADMIN LOW NAME=` option was encountered. All but the first are ignored.

`Duplicate CLASSIFICATION NAME= ignored.`
More than one `CLASSIFICATION NAME=` option was encountered. All but the first are ignored.

`Duplicate COMPARTMENTS NAME= ignored.`
More than one `COMPARTMENTS NAME=` option was encountered. All but the first are ignored.

`Duplicate DEFAULT USER CLEARANCE= ignored.`
More than one `DEFAULT USER CLEARANCE=` option was encountered. All but the first are ignored.

`Duplicate DEFAULT USER SENSITIVITY LABEL= ignored.`
More than one `DEFAULT USER SENSITIVITY LABEL=` option was encountered. All but the first are ignored.

`End of File not found where expected. Found instead: XXX.`
The noted extraneous text was found when the end of label encodings file was expected.

`End of File or LOCAL DEFINITIONS: not found. Found instead: XXX.`
The noted extraneous text was found when the `LOCAL DEFINITIONS:` section or end of label encodings file was expected.

`Found color XXX without associated label.`
The color XXX was found, however it had no label or word associated with it.

`Invalid color label XXX.`
The label XXX cannot be parsed.

`Invalid DEFAULT USER CLEARANCE XXX.`
The DEFAULT USER CLEARANCE XXX cannot be parsed.

```
Invalid DEFAULT USER SENSITIVITY LABEL XXX.
```
The DEFAULT USER SENSITIVITY LABEL XXX cannot be parsed.

```
Label preceding XXX did not have a color specification.
```
A label or word was found without a matching color name.

```
MARKINGS NAME= ignored.
```
This option is obsolete and ignored.

```
Word XXX not found as a valid Sensitivity Label word.
```
The word XXX was not found as a valid word for a sensitivity label.

Trusted Solaris 8 4/01 Reference Manual

`bcltobanner`(3TSOL), `blinset`(3TSOL), `bltocolor`(3TSOL), `bltos`(3TSOL), `blvalid`(3TSOL), `labelinfo`(3TSOL), `labelvers`(3TSOL), `stobl`(3TSOL), `chk_encodings`(1M)

Trusted Solaris Label Administration

Defense Intelligence Agency document DDS-2600-6216-93, *Compartmented Mode Workstation Labeling: Encodings Format*, September 1993.

SunOS 5.8 Reference Manual

`attributes`(5)

WARNINGS

Creation of and modification to the label encodings file should only be undertaken with a thorough understanding not only of the concepts in *Compartmented Mode Workstation Labeling: Encodings Format* but also of the details of the local labeling requirements.

The following warnings are paraphrased from *Compartmented Mode Workstation Labeling: Encodings Format*.

Take extreme care when modifying a label encodings file that is already loaded and running in a Trusted Solaris environment. Once the system runs with the label encodings file, many objects are labeled with sensitivity labels that are well formed with respect to the loaded label encodings file. If the label encodings file is subsequently changed, it is possible that the existing labels will no longer be well-formed. Changing the bit patterns associated with words causes existing objects whose labels contain the words to have possibly invalid labels. Raising the minimum classification or lowering the maximum classification associated with words will likely cause existing objects whose labels contain the words to no longer be well-formed.

Changes to a current encodings file that has already been used should be limited only to adding new classifications or words, changing the names of existing words, or modifying the local extensions. As described in *Compartmented Mode Workstation Labeling: Encodings Format*, it is important to reserve extra inverse bits when the label encodings file is first created to allow for later expansion of the label encodings file to incorporate new inverse words. If an inverse word is added that does not use reserved inverse bits, all existing objects in the environment will erroneously have labels that include the new inverse word.

NOTES Defining the label encodings file is a three-step process. First, the set of human-readable labels to be represented must be identified and understood. The definition of this set includes the list of classifications and other words used in the human-readable labels, relations between and among the words, classification restrictions associated with use of each word, and intended use of the words in mandatory access control and labeling system output. Next, this definition is associated with an internal format of integers, bit patterns, and logical relationship statements. Finally, a label encodings file is created. The *Compartmented Mode Workstation Labeling: Encodings Format* document describes the second and third steps, and assumes that the first has already been performed.

Information labels (ILs) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any ILs on communications and files from systems running earlier releases as ADMIN_LOW.

Even though ILs are not supported, a valid IL section is still required in the label encodings file. A copy of the contents of the SL section may be used to create a valid IL section.

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL]. However, the IL component is fixed at ADMIN_LOW.

As a result, Trusted Solaris 7 and later releases have the following characteristics:

- ILs do not display in window labels; SLs (Sensitivity Labels) display alone within brackets.

- ILs do not float.

- Setting an IL on an object has no effect.

- Getting an object's IL will always return ADMIN_LOW.

- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting ILs are always ADMIN_LOW, and cannot be set on any objects.

NAME	logindevperm, fbtab – login-based device permissions
SYNOPSIS	`/etc/logindevperm`
DESCRIPTION	The `/etc/logindevperm` file contains information that is used by `login`(1) and `ttymon`(1M) to change the owner, group, and permissions of devices upon logging into or out of a console device. By default, this file contains lines for the keyboard, mouse, audio, and frame buffer devices.

In the Trusted Solaris environment, `logindevperm` entries are not needed for the keyboard, mouse, and frame buffer devices, because sensitivity labels on these devices prevent access by user processes. Device allocation based on `allocate`(1) is the preferred method of setting device ownership and permissions on other devices, such as audio.

The owner of the devices listed in `/etc/logindevperm` is set to the owner of the console by `login`(1). The group of the devices is set to the owner's group specified in `/etc/passwd`. The permissions are set as specified in `/etc/logindevperm`.

Fields are separated by TAB and/or SPACE characters. Blank lines and comments can appear anywhere in the file; comments start with a hashmark, ' # ', and continue to the end of the line.

The first field specifies the name of a console device (for example, `/dev/console`). The second field specifies the permissions to which the devices in the *device_list* field (third field) will be set. A *device_list* s a colon-separated list of device names. A device entry that is a directory name and ends with "/*" specifies all entries in the directory (except "." and ".."). For example, "/dev/fbs/*" specifies all frame buffer devices.

Once the devices are owned by the user, their permissions and ownership can be changed using `chmod`(1) and `chown`(1), as with any other user-owned file.

Upon logout the owner and group of these devices will be reset by `ttymon`(1M) to owner `root` and root's group as specified in `/etc/passwd` (typically `other`). The permissions are set as specified in the `/etc/logindevperm` file.

FILES	`/etc/passwd` File that contains user group information.
SUMMARY OF TRUSTED SOLARIS CHANGES	The use of `logindevperm` is not supported, and the default `/etc/logindevperm` file has all entries commented out.
Trusted Solaris 8 4/01 Reference Manual	`allocate`(1), `chmod`(1), `chown`(1), `login`(1), `device_allocate`(4), `device_deallocate`(4)
SunOS 5.8 Reference Manual	`ttymon`(1M), `passwd`(4)
NOTES	`/etc/logindevperm` provides a superset of the functionality provided by `/etc/fbtab` in SunOS 4.x releases.

NAME | mnttab – Mounted file system table

DESCRIPTION | The file /etc/mnttab is the file system that provides read-only access to the table of mounted file systems for the current host. /etc/mnttab is read by programs using the routines described in getmntent(3C). Mounting a file system adds an entry to this table. Unmounting removes an entry from this table. Remounting a file system causes the information in the mounted file system table to be updated to reflect any changes caused by the remount. The list is maintained by the kernel in order of mount time. That is, the first mounted file system is first in the list and the most recently mounted file system is last. When mounted on a mount point the file system appears as a regular file containing the current mnttab information.

Each entry is a line of fields separated by spaces in the form:

special   *mount_point*   *fstype*   *options*   *time*

where

special | The name of the resource to be mounted.

mount_point | The pathname of the directory on which the filesystem is mounted.

fstype | The file system type of the mounted file system.

options | The mount options. (See respective mount file system man page below in SEE ALSO.)

time | The time at which the file system was mounted.

Examples of entries for the *special* field include the pathname of a block-special device, the name of a remote file system in the form of *host:pathname*, or the name of a *swap file* (for example, a file made with mkfile(1M)).

IOCTLS | The following ioctl(2) calls are supported:

MNTIOC_NMOUNTS | Returns the count of mounted resources in the current snapshot in the uint32_t pointed to by *arg*.

MNTIOC_GETDEVLIST | Returns an array of uint32_t's that is twice as long as the length returned by MNTIOC_NMOUNTS. Each pair of numbers is the major and minor device number for the file system at the corresponding line in the current /etc/mnttab snapshot. *arg* points to the memory buffer to receive the device number information.

MNTIOC_SETTAG | Sets a tag word into the options list for a mounted file system. A tag is a notation that will appear in the options string of a mounted file system but it is not recognized or interpreted by the file system code. *arg* points to a filled in mnttagdesc structure, as shown in the following example:

```
uint_t  mtd_major; /* major number for mounted fs */
uint_t  mtd_minor; /* minor number for mounted fs */
char    *mtd_mntpt; /* mount point of file system */
char    *mtd_tag;  /* tag to set/clear */
```

If the tag already exists then it is marked as set but not re-added. Tags can be at most `MAX_MNTOPT_TAG` long.

MNTIOC_CLRTAG	Marks a tag in the options list for a mounted file system as not set. *arg* points to the same structure as `MNTIOC_SETTAG`, which identifies the file system and tag to be cleared.

ERRORS

EFAULT	The arg pointer in an `MNTIOC_` ioctl call pointed to an inaccessible memory location or a character pointer in a `mnttagdesc` structure pointed to an inaccessible memory location.
EINVAL	The tag specified in a `MNTIOC_SETTAG` call already exists as a file system option, or the tag specified in a `MNTIOC_CLRTAG` call does not exist.
ENAMETOOLONG	The tag specified in a `MNTIOC_SETTAG` call is too long or the tag would make the total length of the option string for the mounted file system too long.

WARNINGS

The mnttab file system provides the previously undocumented dev=*xxx* option in the option string for each mounted file system. This is provided for legacy applications that might have been using the dev=information option.

Using dev=*option* in applications is strongly discouraged. The device number string represents a 32-bit quantity and might not contain correct information in 64-bit environments.

Applications requiring device number information for mounted file systems should use the getextmntent(3C) interface, which functions properly in either 32- or 64-bit environments.

FILES

/etc/mnttab
  Usual mount point for mnttab file system

/usr/include/sys/mntio.h
  Header file that contains IOCTL definitions

SUMMARY OF TRUSTED SOLARIS CHANGES

The /etc/mnttab file must have a sensitivity label of ADMIN_LOW and an owner UID of 0.

Trusted Solaris 8 4/01 Reference Manual

mount_hsfs(1M), mount_nfs(1M), mount_pcfs(1M), mount_ufs(1M), mount(1M), read(2), stat(2),

SunOS 5.8 Reference Manual

mkfile(1M), mount_cachefs(1M), ioctl(2), poll(2), getmntent(3C)

NOTES The snapshot of the mnttab information is taken any time a read(2) is performed at offset 0 (the beginning) of the mnttab file. The file modification time returned by stat(2) for the mnttab file is the time of the last change to mounted file system information. A poll(2) system call requesting a POLLRDBAND event can be used to block and wait for the system's mounted file system information to be different from the most recent snapshot since the mnttab file was opened.

NAME	nca.if – the NCA configuration file that specifies physical interfaces
SYNOPSIS	`/etc/nca/nca.if`
DESCRIPTION	Specify the physical interfaces for which the Solaris Network Cache and Accelerator ("NCA") feature will be configured in the `nca.if` configuration file. List the physical interfaces in the file, one per line. To configure NCA to listen on all physical interfaces present on the system backed by a `hostname.{interface_name}`, then list only an asterik ("`*`") in `nca.if`.

Note – The NCA is disabled in the Trusted Solaris environment.

When ncakmod(1) is invoked during system boot, it will attempt to ifconfig(1M) each physical interface specified in the `nca.if` file. Note that there must be an accompanying `hostname.{interface_name}` file and an entry in `/etc/hosts` for the contents of `hostname.{interface_name}`.

You must reboot in order to implement changes to the `nca.if` file.

IA	**EXAMPLE 1** IA: `nca.if` on IA

The following is an example of an `nca.if` file that would be used on an IA system:

```
iprb1
iprb6
iprb8
```

SPARC: SPARC	**EXAMPLE 2** `nca.if` on SPARC

The following is an example of an `nca.if` file that would be used on a SPARC system:

```
hme2
hme3
hme4
```

All Platforms	**EXAMPLE 3** Configuring NCA to Listen on All Physical Interfaces

The following example shows the contents of an `nca.if` file that would be used to configure either platform to listen on all physical interfaces present on the system:

```
*
```

FILES	`/etc/nca/nca.if`	Lists the physical interfaces on which NCA will run.
	`/etc/hostname.{}{0-9}`	Lists all physical interfaces configured on the server.
	`/etc/hosts`	Lists all host names associated with the server. Entries in this file must match with entries in `/etc/hostname.{}{0-9}` for NCA to function.

ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWncar
Interface Stability	Evolving

The Network Cache and Accelerator kernel module is not supported in the Trusted Solaris environment.

ifconfig(1M),nca(1)

attributes(5)

System Administration Guide, Volume 3

NAME | nsswitch.conf – configuration file for the name service switch

SYNOPSIS | `/etc/nsswitch.conf`

DESCRIPTION | The operating environment uses a number of databases of information about hosts, ipnodes, users (`passwd/shadow`), and groups. Data for these can come from a variety of sources: host-names and host-addresses, for example, may be found in `/etc/hosts`, NIS, NIS+, LDAP, or DNS. Zero or more sources may be used for each database; the sources and their lookup order are specified in the `/etc/nsswitch.conf` file.

The following databases use the switch file:

Database	*Used By*
`aliases`	`sendmail`(1M)
`auth_attr`	`getauthnam`(3SECDB)
`automount`	`automount`(1M)
`bootparams`	`rpc.bootparamd`(1M)
`ethers`	`ethers`(3SOCKET)
`group`	`getgrnam`(3C)
`hosts`	`gethostbyname`(3NSL). See `Interaction with netconfig`.
`ipnodes`	`getipnodebyname`(3SOCKET)
`netgroup`	`innetgr`(3C)
`netmasks`	`ifconfig`(1M)
`networks`	`getnetbyname`(3SOCKET)
`passwd`	`getpwnam`(3C), `getspnam`(3C), `getauusernam`(3BSM), `getusernam`(3SECDB)
`printers`	`lp`(1), `lpstat`(1), `cancel`(1), `lpr`(1B), `lpq`(1B), `lprm`(1B), `in.lpd`(1M), `lpadmin`(1M), `lpget`(1M), `lpset`(1M)
`prof_attr`	`getprofname`(3SECDB), `getexecprof`(3SECDB)
`protocols`	`getprotobyname`(3SOCKET)
`publickey`	`getpublickey`(3NSL), `secure_rpc`(3NSL)
`rpc`	`getrpcbyname`(3NSL)
`sendmailvars`	`sendmail`(1M)
`services`	`getservbyname`(3SOCKET). See `Interaction with netconfig`.

Database	Used By
tnrhdb	tnrhdb(4)
tnrhtp	tnrhtp(4)

The following sources may be used:

Source	Uses
files	/etc/hosts, /etc/passwd, /etc/inet/ipnodes, /etc/shadow
nis	NIS(YP)
nisplus	NIS+
ldap	LDAP
dns	Valid only for hosts; uses the Internet Domain Name Service.
compat	Valid only for passwd and group; implements "+" and "-". See Interaction with +/- syntax.
user	Valid only for printers; implements support for ${HOME}/.printers.
xfn	Valid only for printers; implements support for FNS printer contexts. Provided to allow transition away from FNS printer contexts.

There is an entry in /etc/nsswitch.conf for each database. Typically these entries will be simple, such as "protocols: files" or "networks: files nisplus". However, when multiple sources are specified, it is sometimes necessary to define precisely the circumstances under which each source will be tried. A source can return one of the following codes:

Status	Meaning
SUCCESS	Requested database entry was found.
UNAVAIL	Source is not configured on this system or internal failure.
NOTFOUND	Source responded "no such entry"
TRYAGAIN	Source is busy or not responding, might respond to retries.

For each status code, two actions are possible:

Action	*Meaning*
continue	Try the next source in the list.
return	Return now.

Additionally, for TRYAGAIN only, the following actions are possible:

Action	*Meaning*
forever	Retry the current source forever.
n	Retry the current source *n* more times, where *n* is an integer between 0 and MAX_INT (that is, 2.14 billion). After *n* retries has been exhausted, the action will continue to the next source.

The complete syntax of an entry is:

```
<entry>     ::= <database> ":" [<source>
[<criteria>]]*
<criteria>  ::= "[" <criterion>+ "]"
<criterion> ::= <status> "=" <action>
<status>    ::= "success" | "notfound" | "unavail" | "tryagain"
```

For every status except TRYAGAIN, the action syntax is:

```
<action>    ::= "return"  | "continue"
```

For the TRYAGAIN status, the action syntax is:

```
<action>    ::= "return"  | "continue" | "forever" | <n>
<n>         ::= 0...MAX_INT
```

Each entry occupies a single line in the file. Lines that are blank, or that start with white space, are ignored. Everything on a line following a # character is also ignored; the # character can begin anywhere in a line, to be used to begin comments. The <database> and <source> names are case-sensitive, but <action> and <status> names are case-insensitive.

The library functions contain compiled-in default entries that are used if the appropriate entry in nsswitch.conf is absent or syntactically incorrect.

The default criteria for DNS and the NIS server in "DNS-forwarding mode" (and DNS server not responding or busy) is [SUCCESS=return NOTFOUND=continue UNAVAIL=continue TRYAGAIN=continue].

The default criteria for all other sources is [SUCCESS=return NOTFOUND=continue UNAVAIL=continue TRYAGAIN=forever].

The default, or explicitly specified, criteria are meaningless following the last source in an entry; and they are ignored, since the action is always to return to the caller irrespective of the status code the source returns.

Interaction with netconfig

In order to ensure that they all return consistent results, gethostbyname(3NSL), getipnodebyname(3SOCKET), getservbyname(3SOCKET), and netdir_getbyname(3NSL) functions are all implemented in terms of the same internal library function. This function obtains the system-wide source lookup policy for hosts, ipnodes, and services based on the inet family entries in netconfig(4) and uses the switch entries only if the netconfig entries have a "-" in the last column for nametoaddr libraries. See the NOTES section in gethostbyname(3NSL) and getservbyname(3SOCKET) for details.

Interaction with NIS+ NIS/YP-compatibility Mode

The NIS+ server can be run in "YP-compatibility mode", where it handles NIS (YP) requests as well as NIS+ requests. In this case, the clients get much the same results (except for getspnam(3C)) from the "nis" source as from "nisplus"; however, "nisplus" is recommended instead of "nis".

Interaction with server in DNS-forwarding Mode

The NIS (YP) server can be run in "DNS-forwarding mode", where it forwards lookup requests to DNS for host-names and -addresses that do not exist in its database. In this case, specifying "nis" as a source for "hosts" is sufficient to get DNS lookups; "dns" need not be specified explicitly as a source.

In SunOS 5.3 (Solaris 2.3) and compatible versions, the NIS+ server in "NIS/YP-compatibility mode" can also be run in "DNS-forwarding mode" (see rpc.nisd(1M)). Forwarding is effective only for requests originating from its YP clients; "hosts" policy on these clients should be configured appropriately.

Interaction with Password Aging

When password aging is turned on, only a limited set of possible name services are permitted for the passwd: *database* in the /etc/nsswitch.conf file:

passwd:	files
passwd:	files nis
passwd:	files nisplus
passwd:	files ldap
passwd:	compat
passwd_compat:	nisplus
passwd_compat:	ldap

Any other settings will cause the passwd(1) command to fail when it attempts to change the password after expiration and will prevent the user from logging in. These are the *only* permitted settings when password aging has been turned on. Otherwise, you can work around incorrect passwd: lines by using the -r repository argument to the passwd(1) command and using passwd -r repository to override the nsswitch.conf settings and specify in which name service you want to modify your password.

Interaction with +/- syntax	Releases prior to SunOS 5.0 did not have the name service switch but did allow the user some policy control. In /etc/passwd one could have entries of the form *+user* (include the specified user from NIS passwd.byname), *-user* (exclude the specified user) and + (include everything, except excluded users, from NIS passwd.byname). The desired behavior was often "everything in the file followed by everything in NIS", expressed by a solitary + at the end of /etc/passwd. The switch provides an alternative for this case ("passwd: files nis") that does not require + entries in /etc/passwd and /etc/shadow (the latter is a new addition to SunOS 5.0, see shadow(4)).

If this is not sufficient, the NIS/YP compatibility source provides full +/- semantics. It reads /etc/passwd for getpwnam(3C) functions and /etc/shadow for getspnam(3C) functions and, if it finds +/- entries, invokes an appropriate source. By default, the source is "nis", but this may be overridden by specifying "nisplus" or "ldap" as the source for the pseudo-database passwd_compat.

Note that for every /etc/passwd entry, there should be a corresponding entry in the /etc/shadow file.

The NIS/YP compatibility source also provides full +/- semantics for group; the relevant pseudo-database is group_compat.

Useful Configurations	The compiled-in default entries for all databases use NIS (YP) as the enterprise level name service and are identical to those in the default configuration of this file:

passwd:	files nis
group:	files nis
hosts:	nis [NOTFOUND=return] files
ipnodes:	nis [NOTFOUND=return] files
networks:	nis [NOTFOUND=return] files
protocols:	nis [NOTFOUND=return] files
rpc:	nis [NOTFOUND=return] files
ethers:	nis [NOTFOUND=return] files
netmasks:	nis [NOTFOUND=return] files
bootparams:	nis [NOTFOUND=return] files
publickey:	nis [NOTFOUND=return] files
netgroup:	nis
automount:	files nis
aliases:	files nis
services:	files nis

sendmailvars:	files
printers:	user files nis nisplus xfn
auth_attr	files nis
prof_attr	files nis

The policy "nis [NOTFOUND=return] files" implies "if nis is UNAVAIL, continue on to files, and if nis returns NOTFOUND, return to the caller; in other words, treat nis as the authoritative source of information and try files only if nis is down. This, and other policies listed in the default configuration above, are identical to the hard-wired policies in SunOS releases prior to 5.0.

If compatibility with the +/- syntax for passwd and group is required, simply modify the entries for passwd and group to:

passwd:	compat
group:	compat

If NIS+ is the enterprise level name service, the default configuration should be modified to use nisplus instead of nis for every database on client machines. The file /etc/nsswitch.nisplus contains a sample configuration that can be copied to /etc/nsswitch.conf to set this policy.

If LDAP is the enterprise level name service, the default configuration should be modified to use ldap instead of nis for every database on client machines. The file /etc/nsswitch.ldap contains a sample configuration that can be copied to /etc/nsswitch.conf to set this policy.

If the use of +/- syntax is desired in conjunction with nisplus, use the following four entries:

passwd:	compat
passwd_compat:	nisplus OR ldap
group:	compat
group_compat:	nisplus OR ldap

In order to get information from the Internet Domain Name Service for hosts that are not listed in the enterprise level name service, NIS+ or LDAP, use the following configuration and set up the /etc/resolv.conf file (see resolv.conf(4) for more details):

hosts:	nisplus dns [NOTFOUND=return] files

or

hosts:	ldap dns [NOTFOUND=return] files

Enumeration - | Many of the databases have enumeration functions: passwd has getpwent(), hosts
getXXXent() | has gethostent(), and so on. These were reasonable when the only source was
files but often make little sense for hierarchically structured sources that contain
large numbers of entries, much less for multiple sources. The interfaces are still
provided and the implementations strive to provide reasonable results, but the data
returned may be incomplete (enumeration for hosts is simply not supported by the
dns source), inconsistent (if multiple sources are used), formatted in an unexpected
fashion (for a host with a canonical name and three aliases, the nisplus source will
return four hostents, and they may not be consecutive), or very expensive
(enumerating a passwd database of 5,000 users is probably a bad idea). Furthermore,
multiple threads in the same process using the same reentrant enumeration function
(getXXXent_r() are supported beginning with SunOS 5.3) share the same
enumeration position; if they interleave calls, they will enumerate disjoint subsets of
the same database.

In general, the use of the enumeration functions is deprecated. In the case of passwd,
shadow, and group, it may sometimes be appropriate to use fgetgrent(),
fgetpwent(), and fgetspent() (see getgrnam(3C), getpwnam(3C), and
getspnam(3C), respectively), which use only the files source.

FILES | A source named *SSS* is implemented by a shared object named nss_*SSS*.so.1 that
resides in /usr/lib.

/etc/nsswitch.conf	Configuration file.
/usr/lib/nss_compat.so.1	Implements "compat" source.
/usr/lib/nss_dns.so.1	Implements "dns" source.
/usr/lib/nss_files.so.1	Implements "files" source.
/usr/lib/nss_nis.so.1	Implements "nis" source.
/usr/lib/nss_nisplus.so.1	Implements "nisplus" source.
/usr/lib/nss_ldap.so.1	Implements "ldap" source.
/usr/lib/nss_user.so.1	Implements "user" source.
/usr/lib/nss_xfn.so.1	Implements "xfn" source.
/etc/netconfig	Configuration file for netdir(3NSL) functions that redirects hosts/devices policy to the switch.
/etc/nsswitch.files	Sample configuration file that uses "files" only.
/etc/nsswitch.nis	Sample configuration file that uses "files" and "nis".
/etc/nsswitch.nisplus	Sample configuration file that uses "files" and "nisplus".

`/etc/nsswitch.ldap`	Sample configuration file that uses "files" and "ldap".
`/etc/nsswitch.dns`	Sample configuration file that uses "files" and "dns" (but only for hosts:).

SUMMARY OF TRUSTED SOLARIS CHANGES

The following Trusted Solaris network files have been added: `tnrhdb` and `tnrhtp`.

In the default Trusted Solaris environment, an administrative role uses the Name Service Switch action in the System_Admin folder in the Application Manager to edit the `nsswitch.conf` file. This file should not be edited directly.

Trusted Solaris 8 4/01 Reference Manual

passwd(1), automount(1M), ifconfig(1M), rpc.bootparamd(1M), rpc.nisd(1M), sendmail(1M), getauusernam(3BSM), getauthnam(3SECDB), resolv.conf(4)

SunOS 5.8 Reference Manual

ldap(1), nis+(1), getgrnam(3C), getnetgrent(3C), getpwnam(3C), getspnam(3C), gethostbyname(3NSL), getpublickey(3NSL), getrpcbyname(3NSL), netdir(3NSL), secure_rpc(3NSL), getexecprof(3SECDB), getprofnam(3SECDB), getusernam(3SECDB), ethers(3SOCKET), getipnodebyname(3SOCKET), getnetbyname(3SOCKET), getprotobyname(3SOCKET), getservbyname(3SOCKET), netconfig(4), ypfiles(4)

NOTES

Within each process that uses `nsswitch.conf`, the entire file is read only once; if the file is later changed, the process will continue using the old configuration.

Programs that use the `getXXbyYY()` functions cannot be linked statically since the implementation of these functions requires dynamic linker functionality to access the shared objects `/usr/lib/nss_SSS.so.1` at run time.

The use of both `nis` and `nisplus` as sources for the same database is strongly discouraged since both the name services are expected to store similar information and the lookups on the database may yield different results depending on which name service is operational at the time of the request. The same applies for using `ldap` along with `nis` or `nisplus`.

Misspelled names of sources and databases will be treated as legitimate names of (most likely nonexistent) sources and databases.

The following functions do *not* use the switch: fgetgrent(3C), fgetpwent(3C), fgetspent(3C), getpw(3C), putpwent(3C), shadow(4).

NAME | policy.conf – Configuration file for security policy

SYNOPSIS | `/etc/security/policy.conf`

DESCRIPTION | The `policy.conf` file provides the security policy configuration for user-level attributes. Each entry consists of a key/value pair in the form:

key=value

The key/value pair must appear on a single line, and the key must start the line. Lines starting with # are taken as comments and ignored. Option name comparisons are case-insensitive.

The following keys are defined:

AUTHS_GRANTED
Specifies the default set of authorizations granted to all users. This entry is interpreted by `chkauthattr`(3SECDB). The value is one or more comma-separated authorizations defined in `auth_attr`(4).

IDLECMD=logout|lock
Specifies the action to take after the user has been idle for `IDLETIME` minutes. The default value is `lock`.

IDLETIME=*minutes*
Specifies the number of minutes before the specified `IDLECMD` gets executed. Any integer value between 1 and 120 is valid. The default value is `30` minutes.

LABELVIEW=hidesl|showsl
Specifies whether window labels are visible to the user (`showsl`), or not visible (`hidesl`). The default value is `showsl`.

LOCK_AFTER_RETRIES=yes|no
Specifies whether or not an account is locked after the count of failed logins for a user equals or exceeds the allowed number of retries as defined by `RETRIES` in `/etc/default/login`. The default value for users is `yes`. The default value for roles is `no`.

PASSWORD=auto|manual
Specifies how the user's password is changed. If `auto` is specified, the user is given a list of random passwords from which to choose. If `manual` is specified, the user creates a password. The default value is `manual`.

PROFS_GRANTED
Specifies the default set of profiles granted to all users. This entry is interpreted by `chkauthattr`(3SECDB) and `getexecuser`(3SECDB). The value is one or more comma-separated profiles defined in the `prof_attr`(4) file.

EXAMPLES | **EXAMPLE 1** Defining a key/value pair

AUTHS_GRANTED=com.sun.date

EXAMPLE 1 Defining a key/value pair   *(Continued)*

FILES

/etc/user_attr	Defines extended user attributes.
/etc/security/auth_attr	Defines authorizations.
/etc/security/prof_attr	Defines profiles.
/etc/security/policy.conf	Defines policy for the system.

SUMMARY OF TRUSTED SOLARIS CHANGES

The IDLECMD, IDLETIME, LABELVIEW, LOCK_AFTER_RETRIES, and PASSWORD keys are added.

Trusted Solaris 8 4/01 Reference Manual

chkauthattr(3SECDB), prof_attr(4), user_attr(4)

SunOS 5.8 Reference Manual

pfexec(1), getexecuser(3SECDB), auth_attr(4)

NAME | priv_desc – descriptions of defined privileges

SYNOPSIS | `#include <tsol/priv.h>`

DESCRIPTION | Every defined privilege has a manifest constant for use in programs, a name for use in user interfaces, and a description displayed by certain administrative tools. When a process has a privilege in its *effective* set, that process has the power to bypass security policy and perform the task allowed by that privilege.

Privilege Descriptions | The following section gives the manifest constant, name, and description for each privilege defined on this system.

PRIV_FILE_AUDIT

Name   file_audit

Allows a process to get or set a file's or directory's audit preselection information. The audit preselection information may override the preselection information associated with a process' access to a file or directory. Allows a process to get or set a file's or directory's public object flag. The public object flag may override the successful read/search access preselection information associated with a process' access to a file or directory. Allows a process to write to or modify a file or directory without the file's or directory's audit preselection information or public object flag being cleared.

PRIV_FILE_CHOWN

Name   file_chown

Allows a process to change a file's owner user ID. Allows a process to change a file's group ID to one other than the process' effective group ID or one of the process' supplemental group IDs.

PRIV_FILE_DAC_EXECUTE

Name   file_dac_execute

Allows a process to execute an executable file whose permission bits or ACL do not allow the process execute permission.

PRIV_FILE_DAC_READ

Name   file_dac_read

Allows a process to read a file or directory whose permission bits or ACL do not allow the process read permission.

PRIV_FILE_DAC_SEARCH

Name   file_dac_search

Allows a process to search a directory whose permission bits or ACL do not allow the process search permission.

PRIV_FILE_DAC_WRITE

Name   file_dac_write

Allows a process to write a file or directory whose permission bits or ACL do not allow the process write permission.

PRIV_FILE_DOWNGRADE_SL

Name   file_downgrade_sl

Allows a process to set the Sensitivity Label of a file or directory to a Sensitivity Label that does not dominate the existing Sensitivity Label.

PRIV_FILE_FILE_LOCK

Name   file_lock

Allows a process to get accurate lock information for a file lock that it does not hold.

PRIV_FILE_MAC_READ

Name   file_mac_read

Allows a process to read a file or directory whose Sensitivity Label is not dominated by the process' Sensitivity Label. Allows a process to get accurate file attributes of a file or directory whose Sensitivity Label is not dominated by the process' Sensitivity Label. Allows a process, when upgraded directory names are hidden, to get directory entries whose Sensitivity Label is not dominated by the process' Sensitivity Label.

PRIV_FILE_MAC_SEARCH

Name   file_mac_search

Allows a process to search a directory whose Sensitivity Label is not dominated by the process' Sensitivity Label.

PRIV_FILE_MAC_WRITE

Name   file_mac_write

Allows a process to write a file or directory whose Sensitivity Label does not dominate the process' Sensitivity Label, or whose Sensitivity Label dominates the process' Clearance.

PRIV_FILE_OWNER

Name   file_owner

Allows a process which is not the owner of a file to modify that file's access and modification times, audit preselection attributes, privileges, or downgrade labels. Allows a process which is not the owner of a directory to modify that directory's access and modification times or downgrade labels. Allows a process which is not the owner of a file or directory to remove or rename a file or directory whose parent directory has the ''save text image after execution'' (sticky) bit set. Allows a process which is not the owner of a file to mount a ''namefs'' upon that file. (Does not apply to setting access permission bits or ACLs.)

PRIV_FILE_SETDAC

Name   file_setdac

Allows a process which is not the owner of a file or directory to modify that file's or directory's permission bits or ACL.

PRIV_FILE_SETID

Name   file_setid

Allows a process to change the ownership of a file or write to a file without the set-user-ID and set-group-ID bits being cleared. Allows a process to set the set-user-ID bit on a file whose owner is not the process' effective user. Allows a process to set the set-group-ID bit on a file whose group is not the process' effective group or one of the process' supplemental groups.

PRIV_FILE_SETPRIV

Name   file_setpriv

Allows a process to set the privilege sets on an executable file that the process owns. Allows a process to write to an executable file without the file's allowed and forced privilege sets being emptied.

PRIV_FILE_UPGRADE_SL

Name   file_upgrade_sl

Allows a process to set the Sensitivity Label of a file or directory to a Sensitivity Label that dominates the existing Sensitivity Label.

PRIV_IPC_DAC_READ

Name   ipc_dac_read

Allows a process to read a System V IPC Message Queue, Semaphore Set, or Shared Memory Segment whose permission bits do not allow the process read permission.

PRIV_IPC_DAC_WRITE

Name   ipc_dac_write

Allows a process to write a System V IPC Message Queue, Semaphore Set, or Shared Memory Segment whose permission bits do not allow the process write permission.

PRIV_IPC_MAC_WRITE

Name   ipc_mac_write

Allows a process to write a System V IPC Message Queue, Semaphore Set, or Shared Memory Segment whose Sensitivity Label does not dominate the process' Sensitivity Label, or whose Sensitivity Label dominates the process' Clearance.

PRIV_IPC_OWNER

Name   ipc_owner

Allows a process which is not the owner of a System V IPC Message Queue, Semaphore Set, or Shared Memory Segment to remove, change ownership of, or change permission bits of the Message Queue, Semaphore Set, or Shared Memory Segment.

PRIV_NET_BROADCAST

Name   net_broadcast

Allows a process to send broadcast or multicast packets. Because broadcast packets are delivered to all machines on the local network, they are not labeled.

PRIV_NET_DOWNGRADE_SL

Name   net_downgrade_sl

Allows a process to specify a Sensitivity Label for data being written or to set the network endpoint default Sensitivity Label to an Sensitivity Label which does not dominate the process' Sensitivity Label.

PRIV_NET_MAC_READ

Name   net_mac_read

Allows a process to bind to or accept with a multi-level port. Binding to a multi-level port allows the process to read all data sent to that port socket for which there is not a bound single level port that matches the Sensitivity Label of the data. Accepting with a multi-level port allows a process to receive all data sent to that connected port. (There can be no single level connected port for the accept to succeed.) Allows a process to create a multi-level RPC port mapping.

PRIV_NET_PRIVADDR

Name   net_privaddr

Allows a process to bind to a privileged port number. The privilege port
numbers are 1-1023 (the traditional UNIX privileged ports) and 6000-6002
(the XSun server ports). Privileged port numbers include the Internet
reserved (well known) port numbers.

PRIV_NET_RAWACCESS

Name   net_rawaccess

Allows a process to have direct access to the network layer. Direct access to
the network layer bypasses network labeling. Auditing is not bypassed.

PRIV_NET_REPLY_EQUAL

Name   net_reply_equal

Allows a process to reply with the Sensitivity Label of the last packet
received rather than its own Sensitivity Label. A combination of
net_mac_read and net_reply_equal allow unmodified programs to
successfully receive and reply at all Sensitivity Labels. This privilege exists
for unmodified program compatibility and is not used by modified Trusted
Solaris programs.

PRIV_NET_SETCLR

Name   net_setclr

Allows a process to specify a Clearance for data being written or to set the
network endpoint default Clearance to a value different from the process'
Clearance.

PRIV_NET_SETID

Name   net_setid

Allows a process to specify an effective user ID, effective group ID, or set of
supplemental groups for data being written or to set the network endpoint
default effective user ID, effective group ID, or set of supplemental groups
to values different from the process' values. Allows a process which is not
the owner of a RPC port mapping to remove the mapping.

PRIV_NET_SETPRIV

Name   net_setpriv

Allows a process to specify the effective privilege set for data being written
or to set the network endpoint default effective privilege set to privileges
contained in the process' permitted privilege set.

PRIV_NET_UPGRADE_SL

Name   net_upgrade_sl

Allows a process to specify a Sensitivity Label for data being written or to set the network endpoint default Sensitivity Label to a Sensitivity Label which dominates the process' Sensitivity Label.

PRIV_PROC_AUDIT_APPL

Name   proc_audit_appl

Allows a process to generate audit records with an audit event outside the Trusted Solaris TCB event number range. Allows a process to get its own audit preselection information.

PRIV_PROC_AUDIT_TCB

Name   proc_audit_tcb

Allows a process to generate audit records with an audit event within the Trusted Solaris TCB event number range. Allows a process to get its own audit preselection information.

PRIV_PROC_CHROOT

Name   proc_chroot

Allows a process to change its root directory.

PRIV_PROC_DEBUG_NONTRANQUIL

Name   proc_debug_nontranquil

Allows a process to retain access to a process object when that process object changes its Sensitivity Label. Mandatory Access Control is enforced on the new Sensitivity Label.

Note – This privilege is intended to be used to debug processes that change their Sensitivity Labels and not for other purposes.

PRIV_PROC_DUMPCORE

Name   proc_dumpcore

Allows a TCB process to execute a new program which is set-user-ID, set-group-ID, or permits the use of privilege to have a ''core'' file created for it when taking the default action for SIGQUIT, SIGILL, SIGTRAP, SIGABRT, SIGEMT, SIGFPE, SIGBUS, SIGSEGV, SIGSYS, SIGXCPU, or SIGXFSZ signals. Allows a TCB process to have a ''core'' file created for it when taking the default action for SIGQUIT, SIGILL, SIGTRAP, SIGABRT, SIGEMT, SIGFPE, SIGBUS, SIGSEGV, SIGSYS, SIGXCPU, or SIGXFSZ

signals.

PRIV_PROC_MAC_READ

Name   proc_mac_read

Allows a process to read another process whose Sensitivity Label is not dominated by the reading process' Sensitivity Label.

PRIV_PROC_MAC_WRITE

Name   proc_mac_write

Allows a process to write another process whose Sensitivity Label does not dominate the writing process' Sensitivity Label, or whose Sensitivity Label dominates the writing process' Clearance.

PRIV_PROC_NODELAY

Name   proc_nodelay

Allows a process to not be delayed when doing operations that are identified as covert channels.

PRIV_PROC_OWNER

Name   proc_owner

Allows a process to read from and write to another process with a different process owner. Allows a process to bind a process to a CPU with a different process owner. Allows a process to open a process whose program file is set-user-ID or set-group-ID, or has the use of privilege.

PRIV_PROC_SETCLR

Name   proc_setclr

Allows a process to set its Clearance to a Clearance that is not equal to the process' current Clearance.

PRIV_PROC_SETID

Name   proc_setid

Allows a process to set its user or group IDs to one different from its current effective, real, or saved IDs. Allows a process to set its supplemental group IDs. Allows a process to set the process group of a controlling terminal to one not in the process' process group. Allows a process to set the window size on a terminal not in its session.

PRIV_PROC_SETSL

Name   proc_setsl

Allows a process to set its Sensitivity Label to a Sensitivity Label that is not
equal to the process' current Sensitivity Label.

PRIV_PROC_TRANQUIL

Name   proc_tranquil

Allows a process to set the Sensitivity Label of an object to a Sensitivity
Label that is not equal to the current Sensitivity Label when the object is in
use by another process.

PRIV_SYS_AUDIT

Name   sys_audit

Allows a process to start the (kernel) audit daemon. Allows a process to
view and set the audit state (audit user ID, audit terminal ID, audit session
ID, audit preselection mask). Allows a process to turn off and on auditing.
Allows a process to configure the audit parameters (cache and queue sizes,
event to class mappings, policy options).

PRIV_SYS_BOOT

Name   sys_boot

Allows a process to halt, re-boot, or suspend a Trusted Solaris machine.

PRIV_SYS_CONFIG

Name   sys_config

Allows a process to lock into memory and unlock from memory a memory
mapped file or Shared Memory Segment. Allows a process to change the
scheduling priority of a process not owned by this process, or increase this
process' priority. Allows a process to increase its resource or process limits.
Allows a process to set the "save text image after execution" (sticky) bit on
executable files. Allows a process to turn on and off accounting. Allows a
process to change the machine time of day clock. Allows a process to
change the machine high resolution timer clock. Allows a process to
reconfigure scheduling classes. Allows a process to create and delete (hard)
links to directories. Allows a process to place a processor on-line or off-line.
Allows a process to modify kernel driver statistics values.

PRIV_SYS_CONSOLE

Name   sys_console

Allows a process to redirect console output to another device.

PRIV_SYS_DEVICES

Name   sys_devices

Allows a process to create device special files. Allows a process to use `mknod(2)` to create directory and regular files. Allows a process to revoke all access to a device special file. Allows a process to reassign a controlling terminal from one process to another. Allows a process to open a terminal already exclusively opened. Allows a process to revoke access to its controlling terminal. Allows a process to enable or disable keyboard abort processing. Allows a process to map frame buffer devices into its address space. Allows a process to enable or disable a disk's write-check capability. Allows a process to load a kernel loadable driver. Allows a process to control the Floating Point Accelerator. Allows a process to configure autopush STREAMS modules. Allows a process to configure the device driver policy table. Allows a process to successfully call a third party loadable module that calls DDI `drv_priv`.

PRIV_SYS_FS_CONFIG

Name   sys_fs_config

Allows a process to manipulate filesystem locks. Allows a process to set/clear the automatic update (delayed I/O) state of a filesystem. Allows a process to get meta disk allocation information. Allows a process to open a specified inode in a filesystem. Allows a process to set the last access time of a file system object.

PRIV_SYS_IPC_CONFIG

Name   sys_ipc_config

Allows a process to increase the size of a System V IPC Message Queue buffer.

PRIV_SYS_MAXPROC

Name   sys_maxproc

Allows a process to create processes when the maximum number of processes for this process' owning user is exceeded. Allows a process to create the last available process in the system.

PRIV_SYS_MINFREE

Name   sys_minfree

Allows a process to write to a filesystem whose available storage space is below the minimum allowed.

PRIV_SYS_MOUNT

Name   sys_mount

Allows a process to mount filesystems which are restricted from being
freely mounted. Such filesystems include those of type ufs, nfs, tmpfs,
procfs, ... Allows a process to remount the root filesystem. Allows a process
to add and remove swap filesystems. Allows a process to determine the
users of a filesystem.

PRIV_SYS_NET_CONFIG

Name   sys_net_config

Allows a process to configure a machine's network interfaces and routes.
Allows a process to set a machine's host and domain names. Allows a
process to set a machine's kerberos realm. Allows a process to load and
unload host type, accreditation, and default information. Allows a process
direct access to network devices. Allows a process to set endpoint names.
Allows a process to use the rpcmod STREAMS module.

PRIV_SYS_NFS

Name   sys_nfs

Allows a process to start a kernel NFS daemon. Allows a process to start
and stop a kernel NFS lock manager daemon. Allows a process to export
directories for use by NFS clients. Allows a process to retrieve the NFS file
handle for a path name. Allows a process to revoke NFS RPC credentials
for a client it does not own.

PRIV_SYS_SUSER_COMPAT

Name   sys_suser_compat

Allows a process to successfully call a third party loadable module that
calls the kernel suser() function to check for allowed access. This privilege
exists only for third party loadable module compatibility and is not used
by Trusted Solaris.

PRIV_SYS_SYSTEM_DOOR

Name   sys_system_door

Allows a process to create a door that can be opened by processes at any
Sensitivity Label.

PRIV_SYS_TRANS_LABEL

Name   sys_trans_label

Allows a process to translate labels to and from ''external string form'' that
are not dominated by the process' Sensitivity Label.

PRIV_WIN_COLORMAP

Name   win_colormap

Allows a process to override colormap restrictions. Allows a process to install or remove colormaps. Allows a process to retrieve colormap cell entries allocated by other processes.

PRIV_WIN_CONFIG

Name   win_config

Allows a process to configure or destroy resources that are permanently retained by the X server. Allows a process to use SetScreenSaver to set the screen saver timeout value. Allows a process to use ChangeHosts to modify the display access control list. Allows a process to use GrabServer. Allows a process to use the SetCloseDownMode request which may retain window, pixmap, colormap, property, cursor, font, or graphic context resources.

PRIV_WIN_DAC_READ

Name   win_dac_read

Allows a process to read from a window resource that it does not own (has a different user ID.

PRIV_WIN_DAC_WRITE

Name   win_dac_write

Allows a process to write to or create a window resource that it does not own (has a different user ID). A newly created window property is created with the window's user ID.

PRIV_WIN_DEVICES

Name   win_devices

Allows a process to perform operations on window input devices. Allows a process to get and set keyboard and pointer controls. Allows a process to modify pointer button and key mappings.

PRIV_WIN_DGA

Name   win_dga

Allows a process to use the direct graphics access (DGA) X protocol extensions. Direct process access to the frame buffer is still required. Thus the process must have MAC and DAC privileges that allow access to the frame buffer, or the frame buffer must be allocated to the process.

PRIV_WIN_DOWNGRADE_SL

Name   win_downgrade_sl

Allows a process to set the Sensitivity Label of a window resource to a
Sensitivity Label that does not dominate the existing Sensitivity Label.

PRIV_WIN_FONTPATH

Name   win_fontpath

Allows a process to set a font path.

PRIV_WIN_MAC_READ

Name   win_mac_read

Allows a process to read from a window resource whose Sensitivity Label
is not equal to the process Sensitivity Label.

PRIV_WIN_MAC_WRITE

Name   win_mac_write

Allows a process to write to create a window resource whose Sensitivity
Label is not equal to the process Sensitivity Label. A newly created window
property is created with the window's Sensitivity Label.

PRIV_WIN_SELECTION

Name   win_selection

Allows a process to request inter-window data moves without the
intervention of the selection arbitrator.

PRIV_WIN_UPGRADE_SL

Name   win_upgrade_sl

Allows a process to set the Sensitivity Label of a window resource to a
Sensitivity Label that dominates the existing Sensitivity Label.

FILES /usr/lib/tsol/locale/locale/priv_name
Privileges descriptions

</usr/include/sys/tsol/priv_names.h>
Manifest constant and ID value definitions

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu

Trusted Solaris 8 Intro(2), getfpriv(2), setfpriv(2), priv_to_str(3TSOL),
4/01 Reference set_effective_priv(3TSOL), priv_name(4), priv_macros(5)
Manual

priv_desc(4)

Trusted Solaris administrator's document set, Trusted Solaris Developer's Guide

attributes(5)

NAME | priv_name – privilege description database

SYNOPSIS | </usr/lib/tsol/locale/*locale*/priv_name>

DESCRIPTION | The priv_name database specifies localized privilege names and descriptions defined on this system. This database is used along with the <sys/tsol/priv_names.h> file by priv_to_str(3TSOL), str_to_priv(3TSOL), and get_priv_text(3TSOL) to translate between privilege ID, privilege name string, and description.

Each entry in the priv_name database consists of one line with fields separated by colons (:). A line ending with a backslash (\) indicates continuation of the entry on the next line. Lines beginning with a # character are treated as comments. Each entry has the form:

constant:*name*:*description*

The entry fields are:

constant   The *constant* field must match the numeric constant defined for the privilege in the <sys/tsol/priv_names.h> file.

name   The external name of the privilege. It is returned by priv_to_str() and is used by str_to_priv(). It is also used by commands like ppriv and pprivtest. The external name can be customized and localized.

description   The description of the privilege. It is returned by get_priv_text(). The description can be customized and localized.

ATTRIBUTES | See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu

EXAMPLES | **EXAMPLE 1** A priv_name entry

```
# Example entry in /usr/lib/tsol/locale/C/priv_name
#
53:proc_setid: Allows a process to set its user or group ID to \
        one different from its current effective, real, or saved IDs. \
        Allows a process to set its supplemental group IDs. \
        Allows a process to set the process group of a controlling terminal \
        to one not in the process' process group. \
        Allows a process to set the window size on a terminal not in its \
        session.
```

| priv_to_str(3TSOL), priv_desc(4)

attributes(5)

proc(4)

NAME	proc – /proc, the process file system
DESCRIPTION	/proc is a file system that provides access to the state of each process and light-weight process (lwp) in the system. The name of each entry in the /proc directory is a decimal number corresponding to a process-ID. These entries are themselves subdirectories. Access to process state is provided by additional files contained within each subdirectory; the hierarchy is described more completely below. In this document, ''/proc file'' refers to a non-directory file within the hierarchy rooted at /proc. The owner of each /proc file and subdirectory is determined by the user-ID of the process.

/proc can be mounted on any mount point, in addition to the standard /proc mount point, and can be mounted several places at once. Such additional mounts are allowed in order to facilitate the confinement of processes to subtrees of the file system via chroot(1M) and yet allow such processes access to commands like ps(1).

Standard system calls are used to access /proc files: open(2), close(2), read(2), and write(2) (including readv(2), writev(2), pread(2), and pwrite(2)). Most files describe process state and can only be opened for reading. ctl and lwpctl (control) files permit manipulation of process state and can only be opened for writing. as (address space) files contain the image of the running process and can be opened for both reading and writing. An open for writing allows process control; a read-only open allows inspection but not control. In this document, we refer to the process as open for reading or writing if any of its associated /proc files are open for reading or writing.

In general, more than one process can open the same /proc file at the same time. *Exclusive open* is an advisory mechanism provided to allow controlling processes to avoid collisions with each other. A process can obtain exclusive control of a target process, with respect to other cooperating processes, if it successfully opens any /proc file in the target process for writing (the as or ctl files, or the lwpctl file of any lwp) while specifying O_EXCL in the open(2). Such an open will fail if the target process is already open for writing (that is, if an as, ctl, or lwpctl file is already open for writing). There can be any number of concurrent read-only opens; O_EXCL is ignored on opens for reading. It is recommended that the first open for writing by a controlling process use the O_EXCL flag; multiple controlling processes usually result in chaos.

If a process opens one of its own /proc files for writing, the open succeeds regardless of O_EXCL and regardless of whether some other process has the process open for writing. Self-opens do not count when another process attempts an exclusive open. (A process cannot exclude a debugger by opening itself for writing and the application of a debugger cannot prevent a process from opening itself.) All self-opens for writing are forced to be close-on-exec (see the F_SETFD operation of fcntl(2)).

Data may be transferred from or to any locations in the address space of the traced process by applying lseek(2) to position the as file at the virtual address of interest followed by read(2) or write(2) for the combined operation). The address-map file

/proc/*pid*/map can be read to determine the accessible areas (mappings) of the address space. I/O transfers may span contiguous mappings. An I/O request extending into an unmapped area is truncated at the boundary. A write request beginning at an unmapped virtual address fails with EIO; a read request beginning at an unmapped virtual address returns zero (an end-of-file indication).

Information and control operations are provided through additional files. <procfs.h> contains definitions of data structures and message formats used with these files. Some of these definitions involve the use of sets of flags. The set types sigset_t, fltset_t, and sysset_t correspond, respectively, to signal, fault, and system call enumerations defined in <sys/signal.h>, <sys/fault.h>, and <sys/syscall.h>. Each set type is large enough to hold flags for its own enumeration. Although they are of different sizes, they have a common structure and can be manipulated by these macros:

```
prfillset(&set);             /* turn on all flags in set */
premptyset(&set);            /* turn off all flags in set */
praddset(&set, flag);        /* turn on the specified flag */
prdelset(&set, flag);        /* turn off the specified flag */
r = prismember(&set, flag);  /* != 0 iff flag is turned on */
```

One of prfillset() or premptyset() must be used to initialize set before it is used in any other operation. flag must be a member of the enumeration corresponding to set.

The following IOCTLs provided in the Trusted Solaris environment are used to get information about the security attributes of a process: PIOCLABEL, PIOCCLEAR, PIOCEPRIV, PIOCIPRIV, PIOCPPRIV, PIOCSPRIV, PIOCTRED, and PIOCATTR. See the DESCRIPTION and NOTES sections for information about privileges and MAC policies that apply to the use of the /proc file system in the Trusted Solaris environment.

Every process contains at least one *light-weight process*, or *lwp*. Each lwp represents a flow of execution that is independently scheduled by the operating system. All lwps in a process share its address space as well as many other attributes. Through the use of lwpctl and ctl files as described below, it is possible to affect individual lwps in a process or to affect all of them at once, depending on the operation.

When the process has more than one lwp, a representative lwp is chosen by the system for certain process status files and control operations. The representative lwp is a stopped lwp only if all of the process's lwps are stopped; is stopped on an event of interest only if all of the lwps are so stopped (excluding PR_SUSPENDED lwps); is in a PR_REQUESTED stop only if there are no other events of interest to be found; or, failing everything else, is in a PR_SUSPENDED stop (implying that the process is deadlocked). See the description of the status file for definitions of stopped states. See the PCSTOP control operation for the definition of "event of interest".

The representative lwp remains fixed (it will be chosen again on the next operation) as long as all of the lwps are stopped on events of interest or are in a PR_SUSPENDED stop and the PCRUN control operation is not applied to any of them.

When applied to the process control file, every /proc control operation that must act on an lwp uses the same algorithm to choose which lwp to act upon. Together with synchronous stopping (see PCSET), this enables a debugger to control a multiple-lwp process using only the process-level status and control files if it so chooses. More fine-grained control can be achieved using the lwp-specific files.

The system supports two process data models, the traditional 32-bit data model in which ints, longs and pointers are all 32 bits wide (the ILP32 data model), and on some platforms the 64-bit data model in which longs and pointers, but not ints, are 64 bits in width (the LP64 data model). In the LP64 data model some system data types, notably size_t, off_t, time_t and dev_t, grow from 32 bits to 64 bits as well.

The /proc interfaces described here are available to both 32-bit and 64-bit controlling processes. However, many operations attempted by a 32-bit controlling process on a 64-bit target process will fail with EOVERFLOW because the address space range of a 32-bit process cannot encompass a 64-bit process or because the data in some 64-bit system data type cannot be compressed to fit into the corresponding 32-bit type without loss of information. Operations that fail in this circumstance include reading and writing the address space, reading the address-map file, and setting the target process's registers. There is no restriction on operations applied by a 64-bit process to either a 32-bit or a 64-bit target processes.

The format of the contents of any /proc file depends on the data model of the observer (the controlling process), not on the data model of the target process. A 64-bit debugger does not have to translate the information it reads from a /proc file for a 32-bit process from 32-bit format to 64-bit format. However, it usually has to be aware of the data model of the target process. The pr_dmodel field of the status files indicates the target process's data model.

To help deal with system data structures that are read from 32-bit processes, a 64-bit controlling program can be compiled with the C preprocessor symbol _SYSCALL32 defined before system header files are included. This makes explicit 32-bit fixed-width data structures (like cstruct stat32) visible to the 64-bit program. See types32(3HEAD).

DIRECTORY STRUCTURE

At the top level, the directory /proc contains entries each of which names an existing process in the system. These entries are themselves directories. Except where otherwise noted, the files described below can be opened for reading only. In addition, if a process becomes a *zombie* (one that has exited but whose parent has not yet performed a wait(2) upon it), most of its associated /proc files disappear from the hierarchy; subsequent attempts to open them, or to read or write files opened before the process exited, will elicit the error ENOENT.

Although process state and consequently the contents of /proc files can change from instant to instant, a single read(2) of a /proc file is guaranteed to return a sane representation of state; that is, the read will be atomic with respect to the state of the process. No such guarantee applies to successive reads applied to a /proc file for a running process. In addition, atomicity is not guaranteed for I/O applied to the as (address-space) file for a running process or for a process whose address space contains memory shared by another running process.

A number of structure definitions are used to describe the files. These structures may grow by the addition of elements at the end in future releases of the system and it is not legitimate for a program to assume that they will not.

STRUCTURE OF /proc/*pid* A given directory /proc/*pid* contains the following entries. A process can use the invisible alias /proc/self if it wishes to open one of its own /proc files (invisible in the sense that the name ''self'' does not appear in a directory listing of /proc obtained from ls(1), getdents(2), or readdir(3C)).

as Contains the address-space image of the process; it can be opened for both reading and writing. lseek(2) is used to position the file at the virtual address of interest and then the address space can be examined or changed through read(2) or write(2) (or by using pread(2) or pwrite(2) for the combined operation).

ctl A write-only file to which structured messages are written directing the system to change some aspect of the process's state or control its behavior in some way. The seek offset is not relevant when writing to this file. Individual lwps also have associated lwpctl files in the lwp subdirectories. A control message may be written either to the process's ctl file or to a specific lwpctl file with operation-specific effects. The effect of a control message is immediately reflected in the state of the process visible through appropriate status and information files. The types of control messages are described in detail later. See CONTROL MESSAGES.

status Contains state information about the process and the representative lwp. The file contains a pstatus structure which contains an embedded lwpstatus structure for the representative lwp, as follows:

```
typedef struct pstatus {
  int pr_flags;            /* flags (see below) */
  int pr_nlwp;             /* number of lwps in the process */
  pid_tpr_pid;            /* process id */
  pid_tpr_ppid;           /* parent process id */
  pid_tpr_pgid;           /* process group id */
  pid_tpr_sid;            /* session id */
  id_t pr_aslwpid;         /* lwp-id of the aslwp, if any */
  id_t pr_agentid;         /* lwp-id of the agent lwp, if any */
  sigset_t pr_sigpend;     /* set of process pending signals */
  uintptr_t pr_brkbase;    /* virtual address of the process heap */
  size_t pr_brksize;       /* size of the process heap, in bytes */
  uintptr_t pr_stkbase;    /* virtual address of the process stack */
  size_tpr_stksize;       /* size of the process stack, in bytes */
  timestruc_t     pr_utime;    /* process user cpu time */
  timestruc_t     pr_stime;    /* process system cpu time */
```

```
        timestruc_t    pr_cutime;    /* sum of children's user times */
        timestruc_t r_cstime;      /* sum of children's system times */
        sigset_t pr_sigtrace;      /* set of traced signals */
        fltset_t pr_flttrace;      /* set of traced faults */
        sysset_t pr_sysentry;      /* set of system calls traced on entry */
        sysset_t pr_sysexit;       /* set of system calls traced on exit */
        char pr_dmodel;            /* data model of the process */
        taskid_t pr_taskid;        /* task id */
        projid_t pr_projid;        /* project id */
        lwpstatus_t pr_lwp;        /* status of the representative lwp */
} pstatus_t;
```

`pr_flags` is a bit-mask holding the following process flags. For convenience, it also contains the lwp flags for the representative lwp, described later.

PR_ISSYS   process is a system process (see PCSTOP).

PR_VFORKP   process is the parent of a vforked child (see PCWATCH).

PR_FORK   process has its inherit-on-fork mode set (see PCSET).

PR_RLC   process has its run-on-last-close mode set (see PCSET).

PR_KLC   process has its kill-on-last-close mode set (see PCSET).

PR_ASYNC   process has its asynchronous-stop mode set (see PCSET).

PR_MSACCT   process has microstate accounting enabled (see PCSET).

PR_MSFORK   process microstate accounting is inherited on fork (see PCSET).

PR_BPTADJ   process has its breakpoint adjustment mode set (see PCSET).

PR_PTRACE   process has its ptrace-compatibility mode set (see PCSET).

`pr_nlwp` is the total number of lwps in the process.

`pr_pid`, `pr_ppid`, `pr_pgid`, and `pr_sid` are, respectively, the process ID, the ID of the process's parent, the process's process group ID, and the process's session ID.

`pr_aslwpid` is the lwp-ID for the "asynchronous signal lwp" (aslwp). It is zero if there is no aslwp in the process. The aslwp is the lwp designated to redirect asynchronous signals to other lwps in a multi-threaded process. See `signal`(3HEAD) for a description of the aslwp.

`pr_agentid` is the lwp-ID for the /proc agent lwp (see the PCAGENT control operation). It is zero if there is no agent lwp in the process.

`pr_sigpend` identifies asynchronous signals pending for the process.

`pr_brkbase` is the virtual address of the process heap and `pr_brksize` is its size in bytes. The address formed by the sum of these values is the process break (see brk(2)). `pr_stkbase` and `pr_stksize` are, respectively, the virtual address of the

process stack and its size in bytes. (Each lwp runs on a separate stack; the distinguishing characteristic of the process stack is that the operating system will grow it when necessary.)

pr_utime, pr_stime, pr_cutime, and pr_cstime are, respectively, the user CPU and system CPU time consumed by the process, and the cumulative user CPU and system CPU time consumed by the process's children, in seconds and nanoseconds.

pr_sigtrace and pr_flttrace contain, respectively, the set of signals and the set of hardware faults that are being traced (see PCSTRACE and PCSFAULT).

pr_sysentry and pr_sysexit contain, respectively, the sets of system calls being traced on entry and exit (see PCSENTRY and PCSEXIT).

pr_dmodel indicates the data model of the process. Possible values are:

PR_MODEL_ILP32   process data model is ILP32.

PR_MODEL_LP64   process data model is LP64.

PR_MODEL_NATIVE   process data model is native.

The constant PR_MODEL_NATIVE reflects the data model of the controlling process, *that is*, its value is PR_MODEL_ILP32 or PR_MODEL_LP64 according to whether the controlling process has been compiled as a 32-bit program or a 64-bit program, respectively.

pr_lwp contains the status information for the representative lwp:

```
typedef struct lwpstatus {
 int pr_flags;                /* flags (see below) */
 id_t pr_lwpid;               /* specific lwp identifier */
 short pr_why;                /* reason for lwp stop, if stopped */
 short pr_what;               /* more detailed reason */
 short pr_cursig;             /* current signal, if any */
 siginfo_t pr_info;           /* info associated with signal or fault */
 sigset_t pr_lwppend;         /* set of signals pending to the lwp */
 sigset_t pr_lwphold;         /* set of signals blocked by the lwp */
 struct sigaction pr_action;  /* signal action for current signal */
 stack_t pr_altstack;         /* alternate signal stack info */
 uintptr_t pr_oldcontext;     /* address of previous ucontext */
 short pr_syscall;            /* system call number (if in syscall) */
 short pr_nsysarg;            /* number of arguments to this syscall */
 int pr_errno;                /* errno for failed syscall */
 long pr_sysarg[PRSYSARGS];   /* arguments to this syscall */
 long pr_rval1;               /* primary syscall return value */
 long pr_rval2;               /* second syscall return value, if any */
 char pr_clname[PRCLSZ];      /* scheduling class name */
 timestruc_t pr_tstamp;       /* real-time time stamp of stop */
 ulong_t pr_instr;            /* current instruction */
 prgregset_t pr_reg;          /* general registers */
 prfpregset_t pr_fpreg;       /* floating-point registers */
} lwpstatus_t;
```

`pr_flags` is a bit-mask holding the following lwp flags. For convenience, it also contains the process flags, described previously.

PR_STOPPED	lwp is stopped.
PR_ISTOP	lwp is stopped on an event of interest (see PCSTOP).
PR_DSTOP	lwp has a stop directive in effect (see PCSTOP).
PR_STEP	lwp has a single-step directive in effect (see PCRUN).
PR_ASLEEP	lwp is in an interruptible sleep within a system call.
PR_PCINVAL	lwp's current instruction (`pr_instr`) is undefined.
PR_ASLWP	this is the asynchronous signal lwp for the process.
PR_AGENT	this is the `/proc` agent lwp for the process.

`pr_lwpid` names the specific lwp.

`pr_why` and `pr_what` together describe, for a stopped lwp, the reason for the stop. Possible values of `pr_why` and the associated `pr_what` are:

PR_REQUESTED	indicates that the stop occurred in response to a stop directive, normally because PCSTOP was applied or because another lwp stopped on an event of interest and the asynchronous-stop flag (see PCSET) was not set for the process. `pr_what` is unused in this case.
PR_SIGNALLED	indicates that the lwp stopped on receipt of a signal (see PCSTRACE); `pr_what` holds the signal number that caused the stop (for a newly-stopped lwp, the same value is in `pr_cursig`).
PR_FAULTED	indicates that the lwp stopped on incurring a hardware fault (see PCSFAULT); `pr_what` holds the fault number that caused the stop.
PR_SYSENTRY PR_SYSEXIT	indicate a stop on entry to or exit from a system call (see PCSENTRY and PCSEXIT); `pr_what` holds the system call number.
PR_JOBCONTROL	indicates that the lwp stopped due to the default action of a job control stop signal (see `sigaction(2)`); `pr_what` holds the stopping signal number.
PR_SUSPENDED	indicates that the lwp stopped due to internal synchronization of lwps within the process. `pr_what` is unused in this case.

`pr_cursig` names the current signal, that is, the next signal to be delivered to the lwp, if any. `pr_info`, when the lwp is in a PR_SIGNALLED or PR_FAULTED stop, contains additional information pertinent to the particular signal or fault (see `<sys/siginfo.h>`).

`pr_lwppend` identifies any synchronous or directed signals pending for the lwp.
`pr_lwphold` identifies those signals whose delivery is being blocked by the lwp (the
signal mask).

`pr_action` contains the signal action information pertaining to the current signal (see
`sigaction(2)`); it is undefined if `pr_cursig` is zero. `pr_altstack` contains the
alternate signal stack information for the lwp (see `sigaltstack(2)`).

`pr_oldcontext`, if not zero, contains the address on the lwp stack of a `ucontext`
structure describing the previous user-level context (see `ucontext(3HEAD)`). It is
non-zero only if the lwp is executing in the context of a signal handler.

`pr_syscall` is the number of the system call, if any, being executed by the lwp; it is
non-zero if and only if the lwp is stopped on `PR_SYSENTRY` or `PR_SYSEXIT`, or is
asleep within a system call (`PR_ASLEEP` is set). If `pr_syscall` is non-zero,
`pr_nsysarg` is the number of arguments to the system call and `pr_sysarg` contains
the actual arguments.

`pr_rval1`, `pr_rval2`, and `pr_errno` are defined only if the lwp is stopped on
`PR_SYSEXIT` or if the `PR_VFORKP` flag is set. If `pr_errno` is zero, `pr_rval1` and
`pr_rval2` contain the return values from the system call. Otherwise, `pr_errno`
contains the error number for the failing system call (see `<sys/errno.h>`).

`pr_clname` contains the name of the lwp's scheduling class.

`pr_tstamp`, if the lwp is stopped, contains a time stamp marking when the lwp
stopped, in real time seconds and nanoseconds since an arbitrary time in the past.

`pr_instr` contains the machine instruction to which the lwp's program counter
refers. The amount of data retrieved from the process is machine-dependent. On
SPARC based machines, it is a 32-bit word. On IA based machines, it is a single byte.
In general, the size is that of the machine's smallest instruction. If `PR_PCINVAL` is set,
`pr_instr` is undefined; this occurs whenever the lwp is not stopped or when the
program counter refers to an invalid virtual address.

`pr_reg` is an array holding the contents of a stopped lwp's general registers.

SPARC	On SPARC-based machines, the predefined constants `R_G0` ... `R_G7`, `R_O0` ... `R_O7`, `R_L0` ... `R_L7`, `R_I0` ... `R_I7`, `R_PC`, `R_nPC`, and `R_Y` can be used as indices to refer to the corresponding registers; previous register windows can be read from their overflow locations on the stack (however, see the `gwindows` file in the `/proc/`*pid*`/lwp/`*lwpid* subdirectory).
SPARC V8 (32-bit)	For SPARC V8 (32-bit) controlling processes, the predefined constants `R_PSR`, `R_WIM`, and `R_TBR` can be used as indices to refer to the corresponding special registers. For SPARC V9 (64-bit) controlling processes,

the predefined constants R_CCR, R_ASI, and R_FPRS
can be used as indices to refer to the corresponding
special registers.

IA   On IA based machines, the predefined constants SS,
                       UESP, EFL, CS, EIP, ERR, TRAPNO, EAX, ECX, EDX, EBX,
                       ESP, EBP, ESI, EDI, DS, ES, FS, and GS can be used as
                       indices to refer to the corresponding registers.

pr_fpreg is a structure holding the contents of the floating-point registers.

SPARC registers, both general and floating-point, as seen by a 64-bit controlling
process are the V9 versions of the registers, even if the target process is a 32-bit (V8)
process. V8 registers are a subset of the V9 registers.

If the lwp is not stopped, all register values are undefined.

psinfo Contains miscellaneous information about the process and the representative lwp
needed by the ps(1) command.

Mandatory read access to the file descriptor is required. To override this restriction,
the calling process may assert the PRIV_FILE_MAC_READ privilege.

psinfo is accessible after a process becomes a *zombie*. The file contains a psinfo
structure which contains an embedded lwpsinfo structure for the representative
lwp, as follows:

```
typedef struct psinfo {
    int pr_flag;            /* process flags */
    int pr_nlwp;            /* number of lwps in the process */
    pid_t pr_pid;           /* process id */
    pid_t pr_ppid;          /* process id of parent */
    pid_t pr_pgid;          /* process id of process group leader */
    pid_t pr_sid;           /* session id */
    uid_t pr_uid;           /* real user id */
    uid_t pr_euid;          /* effective user id */
    gid_t pr_gid;           /* real group id */
    gid_t pr_egid;          /* effective group id */
    uintptr_t pr_addr;      /* address of process */
    size_t pr_size;         /* size of process image in Kbytes */
    size_t pr_rssize;       /* resident set size in Kbytes */
    dev_t pr_ttydev;        /* controlling tty device (or PRNODEV) */
    ushort_t pr_pctcpu;     /* % of recent cpu time used by all lwps */
    ushort_t pr_pctmem;     /* % of system memory used by process */
    timestruc_t pr_start;   /* process start time, from the epoch */
    timestruc_t pr_time;    /* cpu time for this process */
    timestruc_t pr_ctime;   /* cpu time for reaped children */
    taskid_t pr_taskid;     /* task id */
    projid_t pr_projid;     /* project id */
    char pr_fname[PRFNSZ];  /* name of exec'ed file */
    char pr_psargs[PRARGSZ]; /* initial characters of arg list */
    int pr_wstat;           /* if zombie, the wait() status */
    int pr_argc;            /* initial argument count */
    uintptr_t pr_argv;      /* address of initial argument vector */
```

```
    uintptr_t pr_envp;        /* address of initial environment vector */
    char pr_dmodel;           /* data model of the process */
    lwpsinfo_t pr_lwp;        /* information for representative lwp */
} psinfo_t;
```

Some of the entries in `psinfo`, such as `pr_flag` and `pr_addr`, refer to internal kernel data structures and should not be expected to retain their meanings across different versions of the operating system.

`pr_pctcpu` and `pr_pctmem` are 16-bit binary fractions in the range 0.0 to 1.0 with the binary point to the right of the high-order bit (1.0 == 0x8000). `pr_pctcpu` is the summation over all lwps in the process.

`pr_lwp` contains the `ps`(1) information for the representative lwp. If the process is a *zombie*, `pr_nlwp` and `pr_lwp.pr_lwpid` are zero and the other fields of `pr_lwp` are undefined:

```
typedef struct lwpsinfo {
    int pr_flag;              /* lwp flags */
    id_t pr_lwpid;            /* lwp id */
    uintptr_t pr_addr;        /* internal address of lwp */
    uintptr_t pr_wchan;       /* wait addr for sleeping lwp */
    char pr_stype;            /* synchronization event type */
    char pr_state;            /* numeric lwp state */
    char pr_sname;            /* printable character for pr_state */
    char pr_nice;             /* nice for cpu usage */
    short pr_syscall;         /* system call number (if in syscall) */
    char pr_oldpri;           /* pre-SVR4, low value is high priority */
    char pr_cpu;              /* pre-SVR4, cpu usage for scheduling */
    int pr_pri;               /* priority, high value = high priority */
    ushort_t pr_pctcpu;       /* % of recent cpu time used by this lwp */
    timestruc_t pr_start;     /* lwp start time, from the epoch */
    timestruc_t pr_time;      /* cpu time for this lwp */
    char pr_clname[PRCLSZ];   /* scheduling class name */
    char pr_name[PRFNSZ];     /* name of system lwp */
    processorid_t pr_onpro;   /* processor which last ran this lwp */
    processorid_t pr_bindpro; /* processor to which lwp is bound */
    psetid_t pr_bindpset;     /* processor set to which lwp is bound */
} lwpsinfo_t;
```

Some of the entries in `lwpsinfo`, such as `pr_flag`, `pr_addr`, `pr_wchan`, `pr_stype`, `pr_state`, and `pr_name`, refer to internal kernel data structures and should not be expected to retain their meanings across different versions of the operating system.

`pr_pctcpu` is a 16-bit binary fraction, as described above. It represents the CPU time used by the specific lwp. On a multi-processor machine, the maximum value is 1/N, where N is the number of CPUs.

cred   Contains a description of the credentials associated with the process:

```
typedef struct prcred {
    uid_t pr_euid;       /* effective user id */
    uid_t pr_ruid;       /* real user id */
    uid_t pr_suid;       /* saved user id (from exec) */
    gid_t pr_egid;       /* effective group id */
```

```
    gid_t pr_rgid;      /* real group id */
    gid_t pr_sgid;      /* saved group id (from exec) */
    int pr_ngroups;     /* number of supplementary groups */
    gid_t pr_groups[1]; /* array of supplementary groups */
} prcred_t;
```

Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

The array of associated supplementary groups in pr_groups is of variable length; the cred file contains all of the supplementary groups. pr_ngroups indicates the number of supplementary groups. (See also the PCSCRED control operation.)

sigact | Contains an array of sigaction structures describing the current dispositions of all signals associated with the traced process (see sigaction(2)). Signal numbers are displaced by 1 from array indices, so that the action for signal number *n* appears in position *n*-1 of the array.

Mandatory read access to the file system object is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

auxv | Contains the initial values of the process's aux vector in an array of auxv_t structures (see <sys/auxv.h>). The values are those that were passed by the operating system as startup information to the dynamic linker.

Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

ldt | This file exists only on IA based machines. It is non-empty only if the process has established a local descriptor table (LDT). If non-empty, the file contains the array of currently active LDT entries in an array of elements of type struct ssd, defined in <sys/sysi86.h>, one element for each active LDT entry.

Mandatory read access to the file system object is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

map | Contains information about the virtual address map of the process.

Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

The file contains an array of prmap structures, each of which describes a contiguous virtual address region in the address space of the traced process:

```
typedef struct prmap {
    uintptr_tpr_vaddr;          /* virtual address of mapping */
    size_t pr_size;             /* size of mapping in bytes */
    char pr_mapname[PRMAPSZ];   /* name in /proc/pid/object */
    offset_t pr_offset;         /* offset into mapped object, if any */
    int pr_mflags;              /* protection and attribute flags */
    int pr_pagesize;            /* pagesize for this mapping in bytes */
```

```
    int pr_shmid;                /* SysV shared memory identifier */
} prmap_t;
```

`pr_vaddr` is the virtual address of the mapping within the traced process and `pr_size` is its size in bytes. `pr_mapname`, if it does not contain a null string, contains the name of a file in the `object` directory (see below) that can be opened read-only to obtain a file descriptor for the mapped file associated with the mapping. This enables a debugger to find object file symbol tables without having to know the real path names of the executable file and shared libraries of the process. `pr_offset` is the 64-bit offset within the mapped file (if any) to which the virtual address is mapped.

`pr_mflags` is a bit-mask of protection and attribute flags:

MA_READ   mapping is readable by the traced process.

MA_WRITE   mapping is writable by the traced process.

MA_EXEC   mapping is executable by the traced process.

MA_SHARED   mapping changes are shared by the mapped object.

MA_ISM   mapping is intimate shared memory (shared MMU resources).

A contiguous area of the address space having the same underlying mapped object may appear as multiple mappings due to varying read, write, and execute attributes. The underlying mapped object does not change over the range of a single mapping. An I/O operation to a mapping marked MA_SHARED fails if applied at a virtual address not corresponding to a valid page in the underlying mapped object. A write to a MA_SHARED mapping that is not marked MA_WRITE fails. Reads and writes to private mappings always succeed. Reads and writes to unmapped addresses fail.

`pr_pagesize` is the page size for the mapping, currently always the system pagesize.

`pr_shmid` is the shared memory identifier, if any, for the mapping. Its value is −1 if the mapping is not System V shared memory. See shmget(2).

rmap Contains information about the reserved address ranges of the process. The file contains an array of prmap structures, as defined above for the map file. Each structure describes a contiguous virtual address region in the address space of the traced process that is reserved by the system in the sense that an mmap(2) system call that does not specify MAP_FIXED will not use any part of it for the new mapping. Examples of such reservations include the address ranges reserved for the process stack and the individual thread stacks of a multi-threaded process.

cwd A symbolic link to the process's current working directory (see chdir(2)). A readlink(2) of /proc/*pid*/cwd yields a null string. However, it can be opened, listed, and searched as a directory and can be the target of chdir(2).

root A symbolic link to the process's root directory. /proc/*pid*/root can differ from the system root directory if the process or one of its ancestors executed chroot(2) as a privileged process. It has the same semantics as /proc/*pid*/cwd.

fd | A directory containing references to the open files of the process. Each entry is a decimal number corresponding to an open file descriptor in the process.

If an entry refers to a regular file, it can be opened with normal file system semantics but, to ensure that the controlling process cannot gain greater access than the controlled process, with no file access modes other than its read/write open modes in the controlled process. If an entry refers to a directory, it appears as a symbolic link and can be accessed with the same semantics as /proc/*pid*/cwd. An attempt to open any other type of entry fails with EACCES.

object | A directory containing read-only files with names corresponding to the pr_mapname entries in the map and pagedata files. Opening such a file yields a file descriptor for the underlying mapped file associated with an address-space mapping in the process. The file name a.out appears in the directory as an alias for the process's executable file.

The object directory makes it possible for a controlling process to gain access to the object file and any shared libraries (and consequently the symbol tables) without having to know the actual path names of the executable files.

pagedata | Opening the page data file enables tracking of address space references and modifications on a per-page basis.

Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

A read(2) of the page data file descriptor returns structured page data and atomically clears the page data maintained for the file by the system. That is to say, each read returns data collected since the last read; the first read returns data collected since the file was opened. When the call completes, the read buffer contains the following structure as its header and thereafter contains a number of section header structures and associated byte arrays that must be accessed by walking linearly through the buffer.

```
typedef struct prpageheader {
    timestruc_t pr_tstamp;  /* real time stamp, time of read() */
    ulong_t pr_nmap;        /* number of address space mappings */
    ulong_t pr_npage;       /* total number of pages */
} prpageheader_t;
```

The header is followed by pr_nmap prasmap structures and associated data arrays. The prasmap structure contains at least the following elements:

```
typedef struct prasmap {
    uintptr_t pr_vaddr;         /* virtual address of mapping */
    ulong_t pr_npage;           /* number of pages in mapping */
    char pr_mapname[PRMAPSZ];   /* name in /proc/pid/object */
    offset_t pr_offset;         /* offset into mapped object, if any */
    int pr_mflags;              /* protection and attribute flags */
    int pr_pagesize;            /* pagesize for this mapping in bytes */
    int pr_shmid;               /* SysV shared memory identifier */
} prasmap_t;
```

Each section header is followed by pr_npage bytes, one byte for each page in the mapping, plus 0-7 null bytes at the end so that the next prasmap structure begins on an eight-byte aligned boundary. Each data byte may contain these flags:

PG_REFERENCED   page has been referenced.

PG_MODIFIED   page has been modified.

If the read buffer is not large enough to contain all of the page data, the read fails with E2BIG and the page data is not cleared. The required size of the read buffer can be determined through fstat(2). Application of lseek(2) to the page data file descriptor is ineffective; every read starts from the beginning of the file. Closing the page data file descriptor terminates the system overhead associated with collecting the data.

More than one page data file descriptor for the same process can be opened, up to a system-imposed limit per traced process. A read of one does not affect the data being collected by the system for the others. An open of the page data file will fail with ENOMEM if the system-imposed limit would be exceeded.

watch Contains an array of prwatch structures, one for each watched area established by the PCWATCH control operation. See PCWATCH for details.

usage Contains process usage information described by a prusage structure which contains at least the following fields:

```
typedef struct prusage {
    id_tpr_lwpid;         /* lwp id.  0: process or defunct */
    int pr_count;         /* number of contributing lwps */
    timestruc_t pr_tstamp;   /* real time stamp, time of read() */
    timestruc_t pr_create;   /* process/lwp creation time stamp */
    timestruc_t pr_term;     /* process/lwp termination time stamp */
    timestruc_t pr_rtime;    /* total lwp real (elapsed) time */
    timestruc_t pr_utime;    /* user level CPU time */
    timestruc_t pr_stime;    /* system call CPU time */
    timestruc_t pr_ttime;    /* other system trap CPU time */
    timestruc_t pr_tftime;   /* text page fault sleep time */
    timestruc_t pr_dftime;   /* data page fault sleep time */
    timestruc_t pr_kftime;   /* kernel page fault sleep time */
    timestruc_t pr_ltime;    /* user lock wait sleep time */
    timestruc_t pr_slptime;  /* all other sleep time */
    timestruc_t pr_wtime;    /* wait-cpu (latency) time */
    timestruc_t pr_stoptime; /* stopped time */
    ulong_t pr_minf;         /* minor page faults */
    ulong_t pr_majf;         /* major page faults */
    ulong_t pr_nswap;        /* swaps */
    ulong_t pr_inblk;        /* input blocks */
    ulong_t pr_oublk;        /* output blocks */
    ulong_t pr_msnd;         /* messages sent */
    ulong_t pr_mrcv;         /* messages received */
    ulong_t pr_sigs;         /* signals received */
    ulong_t pr_vctx;         /* voluntary context switches */
    ulong_t pr_ictx;         /* involuntary context switches */
    ulong_t pr_sysc;         /* system calls */
```

```
        ulong_t pr_ioch;          /* chars read and written */
} prusage_t;
```

Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.

If microstate accounting has not been enabled for the process (see the `PR_MSACCT` flag for the `PCSET` operation, below), the `usage` file contains only an estimate of times spent in the various states. The `usage` file is accessible after a process becomes a *zombie*.

lstatus | Contains a `prheader` structure followed by an array of `lwpstatus` structures, one for each lwp in the process (see also /proc/*pid*/lwp/*lwpid*/lwpstatus, below). The `prheader` structure describes the number and size of the array entries that follow.

```
typedef struct prheader {
    long pr_nent;          /* number of entries */
    size_t pr_entsize;     /* size of each entry, in bytes */
} prheader_t;
```

The `lwpstatus` structure may grow by the addition of elements at the end in future releases of the system. Programs must use `pr_entsize` in the file header to index through the array. These comments apply to all /proc files that include a `prheader` structure (`lpsinfo` and `lusage`, below).

lpsinfo | Contains a `prheader` structure followed by an array of `lwpsinfo` structures, one for each lwp in the process. (See also /proc/*pid*/lwp/*lwpid*/lwpsinfo, below.)

lusage | Contains a `prheader` structure followed by an array of `prusage` structures, one for each lwp in the process plus an additional element at the beginning that contains the summation over all defunct lwps (lwps that once existed but no longer exist in the process).

Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.

Excluding the `pr_lwpid`, `pr_tstamp`, `pr_create`, and `pr_term` entries, the entry-by-entry summation over all these structures is the definition of the process usage information obtained from the `usage` file. (See also /proc/*pid*/lwp/*lwpid*/lwpusage, below.)

lwp | A directory containing entries each of which names an lwp within the process. These entries are themselves directories containing additional files as described below.

STRUCTURE OF | A given directory /proc/*pid*/lwp/*lwpid* contains the following entries:
/proc/*pid*/lwp/*lwpid*

lwpctl | Write-only control file. The messages written to this file affect the specific lwp rather than the representative lwp, as is the case for the process's `ctl` file.

lwpstatus | lwp-specific state information. This file contains the `lwpstatus` structure for the specific lwp as described above for the representative lwp in the process's `status` file.

lwpsinfo	lwp-specific ps(1) information. This file contains the lwpsinfo structure for the specific lwp as described above for the representative lwp in the process's psinfo file.
lwpusage	This file contains the prusage structure for the specific lwp as described above for the process's usage file.
gwindows	This file exists only on SPARC based machines. If it is non-empty, it contains a gwindows_t structure, defined in <sys/regset.h>, with the values of those SPARC register windows that could not be stored on the stack when the lwp stopped. Conditions under which register windows are not stored on the stack are: the stack pointer refers to nonexistent process memory or the stack pointer is improperly aligned. If the lwp is not stopped or if there are no register windows that could not be stored on the stack, the file is empty (the usual case).
xregs	Extra state registers. The extra state register set is architecture dependent; this file is empty if the system does not support extra state registers. If the file is non-empty, it contains an architecture dependent structure of type prxregset_t, defined in <procfs.h>, with the values of the lwp's extra state registers. If the lwp is not stopped, all register values are undefined. See also the PCSXREG control operation, below.
asrs	This file exists only for 64-bit SPARC V9 processes. It contains an asrset_t structure, defined in <sys/regset.h>, containing the values of the lwp's platform-dependent ancillary state registers. If the lwp is not stopped, all register values are undefined. See also the PCSASRS control operation, below.
CONTROL MESSAGES	Process state changes are effected through messages written to a process's ctl file or to an individual lwp's lwpctl file. All control messages consist of a long that names the specific operation followed by additional data containing the operand, if any.
	Multiple control messages may be combined in a single write(2) (or writev(2)) to a control file, but no partial writes are permitted. That is, each control message, operation code plus operand, if any, must be presented in its entirety to the write(2) and not in pieces over several system calls. If a control operation fails, no subsequent operations contained in the same write(2) are attempted.
	Descriptions of the allowable control messages follow. In all cases, writing a message to a control file for a process or lwp that has terminated elicits the error ENOENT.
PCSTOP PCDSTOP PCWSTOP PCTWSTOP	When applied to the process control file, PCSTOP directs all lwps to stop and waits for them to stop, PCDSTOP directs all lwps to stop without waiting for them to stop, and PCWSTOP simply waits for all lwps to stop. When applied to an lwp control file, PCSTOP directs the specific lwp to stop and waits until it has stopped, PCDSTOP directs the specific lwp to stop without waiting for it to stop, and PCWSTOP simply waits for the specific lwp to stop. When applied to an lwp control file, PCSTOP and PCWSTOP complete when the lwp stops on an event of interest, immediately if already so stopped; when applied to the process control file, they complete when every lwp has stopped either on an event of interest or on a PR_SUSPENDED stop.

PCTWSTOP is identical to PCWSTOP except that it enables the operation to time out, to avoid waiting forever for a process or lwp that may never stop on an event of interest. PCTWSTOP takes a `long` operand specifying a number of milliseconds; the wait will terminate successfully after the specified number of milliseconds even if the process or lwp has not stopped; a timeout value of zero makes the operation identical to PCWSTOP.

An ''event of interest'' is either a PR_REQUESTED stop or a stop that has been specified in the process's tracing flags (set by PCSTRACE, PCSFAULT, PCSENTRY, and PCSEXIT). PR_JOBCONTROL and PR_SUSPENDED stops are specifically not events of interest. (An lwp may stop twice due to a stop signal, first showing PR_SIGNALLED if the signal is traced and again showing PR_JOBCONTROL if the lwp is set running without clearing the signal.) If PCSTOP or PCDSTOP is applied to an lwp that is stopped, but not on an event of interest, the stop directive takes effect when the lwp is restarted by the competing mechanism. At that time, the lwp enters a PR_REQUESTED stop before executing any user-level code.

A write of a control message that blocks is interruptible by a signal so that, for example, an alarm(2) can be set to avoid waiting forever for a process or lwp that may never stop on an event of interest. If PCSTOP is interrupted, the lwp stop directives remain in effect even though the write(2) returns an error. (Use of PCTWSTOP with a non-zero timeout is recommended over PCWSTOP with an alarm(2).)

A system process (indicated by the PR_ISSYS flag) never executes at user level, has no user-level address space visible through /proc, and cannot be stopped. Applying one of these operations to a system process or any of its lwps elicits the error EBUSY.

PCRUN Make an lwp runnable again after a stop. Mandatory write access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege. This operation takes a `long` operand containing zero or more of the following flags:

PRCSIG   clears the current signal, if any (see PCCSIG).

PRCFAULT   clears the current fault, if any (see PCCFAULT).

PRSTEP   directs the lwp to execute a single machine instruction. On completion of the instruction, a trace trap occurs. If FLTTRACE is being traced, the lwp stops; otherwise, it is sent SIGTRAP. If SIGTRAP is being traced and is not blocked, the lwp stops. When the lwp stops on an event of interest, the single-step directive is cancelled, even if the stop occurs before the instruction is executed. This operation requires hardware and operating system support and may not be implemented on all processors. It is implemented on SPARC and IA based machines.

PRSABORT   is meaningful only if the lwp is in a PR_SYSENTRY stop or is marked PR_ASLEEP; it instructs the lwp to abort execution of the system call (see PCSENTRY and PCSEXIT).

PRSTOP   directs the lwp to stop again as soon as possible after resuming
                  execution (see PCDSTOP). In particular, if the lwp is stopped on
                  PR_SIGNALLED or PR_FAULTED, the next stop will show
                  PR_REQUESTED, no other stop will have intervened, and the lwp
                  will not have executed any user-level code.

When applied to an lwp control file, PCRUN clears any outstanding directed-stop
request and makes the specific lwp runnable. The operation fails with EBUSY if the
specific lwp is not stopped on an event of interest or has not been directed to stop or if
the agent lwp exists and this is not the agent lwp (see PCAGENT).

When applied to the process control file, a representative lwp is chosen for the
operation as described for /proc/*pid*/status. The operation fails with EBUSY if the
representative lwp is not stopped on an event of interest or has not been directed to
stop or if the agent lwp exists. If PRSTEP or PRSTOP was requested, the representative
lwp is made runnable and its outstanding directed-stop request is cleared; otherwise
all outstanding directed-stop requests are cleared and, if it was stopped on an event of
interest, the representative lwp is marked PR_REQUESTED. If, as a consequence, all
lwps are in the PR_REQUESTED or PR_SUSPENDED stop state, all lwps showing
PR_REQUESTED are made runnable.

PCSTRACE   Define a set of signals to be traced in the process. The receipt of one of these signals by
                an lwp causes the lwp to stop. The set of signals is defined using an operand
                sigset_t contained in the control message. Receipt of SIGKILL cannot be traced; if
                specified, it is silently ignored.

                Mandatory write access is required to the file system object. To override this
                restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.

                If a signal that is included in an lwp's held signal set (the signal mask) is sent to the
                lwp, the signal is not received and does not cause a stop until it is removed from the
                held signal set, either by the lwp itself or by setting the held signal set with PCSHOLD.

PCCSIG   The current signal, if any, is cleared from the specific or representative lwp.

PCSSIG   The current signal and its associated signal information for the specific or
                representative lwp are set according to the contents of the operand siginfo structure
                (see <sys/siginfo.h>)

                Mandatory write access is required to the file system object. To override this
                restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.

                If the specified signal number is zero, the current signal is cleared. The semantics of
                this operation are different from those of kill(2) in that the signal is delivered to the
                lwp immediately after execution is resumed (even if it is being blocked) and an
                additional PR_SIGNALLED stop does not intervene even if the signal is traced. Setting
                the current signal to SIGKILL terminates the process immediately.

PCKILL If applied to the process control file, a signal is sent to the process with semantics identical to those of kill(2). If applied to an lwp control file, a directed signal is sent to the specific lwp. The signal is named in a `long` operand contained in the message. Sending SIGKILL terminates the process immediately.

Mandatory write access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.

PCUNKILL A signal is deleted, that is, it is removed from the set of pending signals. If applied to the process control file, the signal is deleted from the process's pending signals. If applied to an lwp control file, the signal is deleted from the lwp's pending signals. The current signal (if any) is unaffected. The signal is named in a `long` operand in the control message. It is an error (EINVAL) to attempt to delete SIGKILL.

Mandatory write access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.

PCSHOLD Set the set of held signals for the specific or representative lwp (signals whose delivery will be blocked if sent to the lwp). The set of signals is specified with a `sigset_t` operand. SIGKILL and SIGSTOP cannot be held; if specified, they are silently ignored.

Mandatory write access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.

PCSFAULT Define a set of hardware faults to be traced in the process. On incurring one of these faults, an lwp stops.

Mandatory write access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.

The set is defined via the operand `fltset_t` structure. Fault names are defined in `<sys/fault.h>` and include the following. Some of these may not occur on all processors; there may be processor-specific faults in addition to these.

FLTILL	illegal instruction
FLTPRIV	privileged instruction
FLTBPT	breakpoint trap
FLTTRACE	trace trap (single-step)
FLTWATCH	watchpoint trap
FLTACCESS	memory access fault (bus error)
FLTBOUNDS	memory bounds violation
FLTIOVF	integer overflow
FLTIZDIV	integer zero divide
FLTFPE	floating-point exception

FLTSTACK   unrecoverable stack fault

FLTPAGE   recoverable page fault

When not traced, a fault normally results in the posting of a signal to the lwp that incurred the fault. If an lwp stops on a fault, the signal is posted to the lwp when execution is resumed unless the fault is cleared by PCCFAULT or by the PRCFAULT option of PCRUN. FLTPAGE is an exception; no signal is posted. The pr_info field in the lwpstatus structure identifies the signal to be sent and contains machine-specific information about the fault.

PCCFAULT   The current fault, if any, is cleared; the associated signal will not be sent to the specific or representative lwp.

Mandatory write access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.

PIOCLABEL   Returns the sensitivity label of the process associated with the file descriptor. Mandatory read access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PIOCCLEAR   Returns the clearance of the process associated with the file descriptor. Mandatory read access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PIOCEPRIV   Returns the effective privilege set of the process associated with the file descriptor. Mandatory read access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PIOCIPRIV   Returns the inheritable privilege set of the process associated with the file descriptor. Mandatory read access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PIOCPPRIV   Returns the permitted privilege set of the process associated with the file descriptor. Read access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PIOCSPRIV   Returns the saved privilege set of the process associated with the file descriptor. Mandatory read access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PIOCATTR   Returns the Trusted Solaris process attributes of the process associated with the file descriptor. Mandatory read access is required to the file system object. To overrdie this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege. For the calling process to receive the file system object's PAF_LABEL_XLAT attribute flags, the PAF_TRUSTED_PATH attribute flag of the calling process must be set.

PIOCAPSA   Returns the audit attributes of the calling process. Mandatory read access to the file system object is required. to override this restriction, the calling process may assert the

PRIV_FILE_MAC_READ privilege. The calling process may assert one of the following:
PRIV_PROC_AUDIT_APPL or PRIV_PROC_AUDIT_TCB or PRIV_SYS_AUDIT
privilege.

PIOCTCRED   Returns the Trusted Solaris process credentials of the process associated with the file
descriptor. Mandatory read access to the file system object is required. To override this
restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PCSENTRY   These control operations instruct the process's lwps to stop on entry to or exit from
PCSEXIT   specified system calls.

Mandatory write access is required to the file system object. To override this
restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.

The set of system calls to be traced is defined via an operand sysset_t structure.

When entry to a system call is being traced, an lwp stops after having begun the call to
the system but before the system call arguments have been fetched from the lwp.
When exit from a system call is being traced, an lwp stops on completion of the
system call just prior to checking for signals and returning to user level. At this point,
all return values have been stored into the lwp's registers.

If an lwp is stopped on entry to a system call (PR_SYSENTRY) or when sleeping in an
interruptible system call (PR_ASLEEP is set), it may be instructed to go directly to
system call exit by specifying the PRSABORT flag in a PCRUN control message. Unless
exit from the system call is being traced, the lwp returns to user level showing EINTR.

PCWATCH   Set or clear a watched area in the controlled process from a prwatch structure
operand:

```
typedef struct prwatch {
    uintptr_t pr_vaddr;  /* virtual address of watched area */
    size_t pr_size;      /* size of watched area in bytes */
    int pr_wflags;       /* watch type flags */
} prwatch_t;
```

pr_vaddr specifies the virtual address of an area of memory to be watched in the
controlled process. pr_size specifies the size of the area, in bytes. pr_wflags
specifies the type of memory access to be monitored as a bit-mask of the following
flags:

WA_READ	read access
WA_WRITE	write access
WA_EXEC	execution access
WA_TRAPAFTER	trap after the instruction completes

If `pr_wflags` is non-empty, a watched area is established for the virtual address range specified by `pr_vaddr` and `pr_size`. If `pr_wflags` is empty, any previously-established watched area starting at the specified virtual address is cleared; `pr_size` is ignored.

A watchpoint is triggered when an lwp in the traced process makes a memory reference that covers at least one byte of a watched area and the memory reference is as specified in `pr_wflags`. When an lwp triggers a watchpoint, it incurs a watchpoint trap. If `FLTWATCH` is being traced, the lwp stops; otherwise, it is sent a `SIGTRAP` signal; if `SIGTRAP` is being traced and is not blocked, the lwp stops.

The watchpoint trap occurs before the instruction completes unless `WA_TRAPAFTER` was specified, in which case it occurs after the instruction completes. If it occurs before completion, the memory is not modified. If it occurs after completion, the memory is modified (if the access is a write access).

`pr_info` in the `lwpstatus` structure contains information pertinent to the watchpoint trap. In particular, the `si_addr` field contains the virtual address of the memory reference that triggered the watchpoint, and the `si_code` field contains one of `TRAP_RWATCH`, `TRAP_WWATCH`, or `TRAP_XWATCH`, indicating read, write, or execute access, respectively. The `si_trapafter` field is zero unless `WA_TRAPAFTER` is in effect for this watched area; non-zero indicates that the current instruction is not the instruction that incurred the watchpoint trap. The `si_pc` field contains the virtual address of the instruction that incurred the trap.

A watchpoint trap may be triggered while executing a system call that makes reference to the traced process's memory. The lwp that is executing the system call incurs the watchpoint trap while still in the system call. If it stops as a result, the `lwpstatus` structure contains the system call number and its arguments. If the lwp does not stop, or if it is set running again without clearing the signal or fault, the system call fails with `EFAULT`. If `WA_TRAPAFTER` was specified, the memory reference will have completed and the memory will have been modified (if the access was a write access) when the watchpoint trap occurs.

If more than one of `WA_READ`, `WA_WRITE`, and `WA_EXEC` is specified for a watched area, and a single instruction incurs more than one of the specified types, only one is reported when the watchpoint trap occurs. The precedence is `WA_EXEC`, `WA_READ`, `WA_WRITE` (`WA_EXEC` and `WA_READ` take precedence over `WA_WRITE`), unless `WA_TRAPAFTER` was specified, in which case it is `WA_WRITE`, `WA_READ`, `WA_EXEC` (`WA_WRITE` takes precedence).

`PCWATCH` fails with `EINVAL` if an attempt is made to specify overlapping watched areas or if `pr_wflags` contains flags other than those specified above. It fails with `ENOMEM` if an attempt is made to establish more watched areas than the system can support (the system can support thousands).

The child of a `vfork`(2) borrows the parent's address space. When a `vfork`(2) is executed by a traced process, all watched areas established for the parent are

suspended until the child terminates or performs an exec(2). Any watched areas established independently in the child are cancelled when the parent resumes after the child's termination or exec(2). PCWATCH fails with EBUSY if applied to the parent of a vfork(2) before the child has terminated or performed an exec(2). The PR_VFORKP flag is set in the pstatus structure for such a parent process.

Certain accesses of the traced process's address space by the operating system are immune to watchpoints. The initial construction of a signal stack frame when a signal is delivered to an lwp will not trigger a watchpoint trap even if the new frame covers watched areas of the stack. Once the signal handler is entered, watchpoint traps occur normally. On SPARC based machines, register window overflow and underflow will not trigger watchpoint traps, even if the register window save areas cover watched areas of the stack.

Watched areas are not inherited by child processes, even if the traced process's inherit-on-fork mode, PR_FORK, is set (see PCSET, below). All watched areas are cancelled when the traced process performs a successful exec(2).

PCSET PCUNSET   PCSET sets one or more modes of operation for the traced process.

Mandatory write access is required to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.

PCUNSET unsets these modes. The modes to be set or unset are specified by flags in an operand long in the control message:

PR_FORK   (inherit-on-fork): When set, the process's tracing flags and its inherit-on-fork mode are inherited by the child of a fork(2) or vfork(2). When unset, child processes start with all tracing flags cleared.

                 Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PR_RLC   (run-on-last-close): When set and the last writable /proc file descriptor referring to the traced process or any of its lwps is closed, all of the process's tracing flags and watched areas are cleared, any outstanding stop directives are canceled, and if any lwps are stopped on events of interest, they are set running as though PCRUN had been applied to them. When unset, the process's tracing flags and watched areas are retained and lwps are not set running on last close.

                 Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PR_KLC   (kill-on-last-close): When set and the last writable /proc file descriptor referring to the traced process or any of its lwps is closed, the process is terminated with SIGKILL.

PR_ASYNC   (asynchronous-stop): When set, a stop on an event of interest by one lwp does not directly affect any other lwp in the process. When unset and an lwp stops on an event of interest other than PR_REQUESTED, all other lwps in the process are directed to stop.

PR_MSACCT   (microstate accounting): When set, microstate accounting is enabled for the process. This allows the usage file to contain accurate values for the times the lwps spent in their various processing states. When unset (the default), the overhead of microstate accounting is avoided and the usage file can only contain an estimate of times spent in the various states.

PR_MSFORK   (inherit microstate accounting): When set, and microstate accounting is enabled for the process, microstate accounting will be enabled for future child processes. When unset, child processes start with microstate accounting disabled.

PR_BPTADJ   (breakpoint trap pc adjustment): On IA based machines, a breakpoint trap leaves the program counter (the EIP) referring to the breakpointed instruction plus one byte. When PR_BPTADJ is set, the system will adjust the program counter back to the location of the breakpointed instruction when the lwp stops on a breakpoint. This flag has no effect on SPARC based machines, where breakpoint traps leave the program counter referring to the breakpointed instruction.

PR_PTRACE   (ptrace-compatibility): When set, a stop on an event of interest by the traced process is reported to the parent of the traced process via wait(2), SIGTRAP is sent to the traced process when it executes a successful exec(2), setuid/setgid flags are not honored for execs performed by the traced process, any exec of an object file that the traced process cannot read fails, and the process dies when its parent dies. This mode is deprecated; it is provided only to allow ptrace(2) to be implemented as a library function using /proc.

It is an error (EINVAL) to specify flags other than those described above or to apply these operations to a system process. The current modes are reported in the pr_flags field of /proc/*pid*/status and /proc/*pid*/lwp/*lwp*/lwpstatus.

PCSREG   Set the general registers for the specific or representative lwp according to the operand prgregset_t structure.

On SPARC based systems, only the condition-code bits of the processor-status register (R_PSR) of SPARC V8 (32-bit) processes can be modified by PCSREG. Other privileged registers cannot be modified at all.

On IA based systems, only certain bits of the flags register (EFL) can be modified by PCSREG: these include the condition codes, direction-bit, and overflow-bit.

Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PCSREG fails with EBUSY if the lwp is not stopped on an event of interest.

PCSVADDR Set the address at which execution will resume for the specific or representative lwp from the operand long. On SPARC based systems, both %pc and %npc are set, with %npc set to the instruction following the virtual address. On IA based systems, only %eip is set. PCSVADDR fails with EBUSY if the lwp is not stopped on an event of interest.

PCSFPREG Set the floating-point registers for the specific or representative lwp according to the operand prfpregset_t structure. An error (EINVAL) is returned if the system does not support floating-point operations (no floating-point hardware and the system does not emulate floating-point machine instructions). PCSFPREG fails with EBUSY if the lwp is not stopped on an event of interest.

Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PCSXREG Set the extra state registers for the specific or representative lwp according to the architecture-dependent operand prxregset_t structure. An error (EINVAL) is returned if the system does not support extra state registers. PCSXREG fails with EBUSY if the lwp is not stopped on an event of interest.

Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PCSASRS Set the ancillary state registers for the specific or representative lwp according to the SPARC V9 platform-dependent operand asrset_t structure. An error (EINVAL) is returned if either the target process or the controlling process is not a 64-bit SPARC V9 process. Most of the ancillary state registers are privileged registers that cannot be modified. Only those that can be modified are set; all others are silently ignored. PCSASRS fails with EBUSY if the lwp is not stopped on an event of interest.

PCAGENT Create an agent lwp in the controlled process with register values from the operand prgregset_t structure (see PCSREG, above). The agent lwp is created in the stopped state showing PR_REQUESTED and with its held signal set (the signal mask) having all signals except SIGKILL and SIGSTOP blocked.

The PCAGENT operation fails with EBUSY unless the process is fully stopped via /proc, that is, unless all of the lwps in the process are stopped either on events of interest or on PR_SUSPENDED, or are stopped on PR_JOBCONTROL and have been directed to stop via PCDSTOP. It fails with EBUSY if an agent lwp already exists. It fails with ENOMEM if system resources for creating new lwps have been exhausted.

Any PCRUN operation applied to the process control file or to the control file of an lwp other than the agent lwp fails with EBUSY as long as the agent lwp exists. The agent lwp must be caused to terminate by executing the _lwp_exit(2) system call before the process can be restarted.

Once the agent lwp is created, its lwp-ID can be found by reading the process status file. To facilitate opening the agent lwp's control and status files, the directory name /propc/*pid*/lwp/agent is accepted for lookup operations as an invisible alias for /proc/*pid*/lwp/*lwpid*, *lwpid* being the lwp-ID of the agent lwp (invisible in the sense that the name "agent" does not appear in a directory listing of /proc/*pid*/lwpobtained from ls(1), getdents(2), or readdir(3C)).

The purpose of the agent lwp is to perform operations in the controlled process on behalf of the controlling process: to gather information not directly available via /proc files, or in general to make the process change state in ways not directly available via /proc control operations. To make use of an agent lwp, the controlling process must be capable of making it execute system calls (specifically, the _lwp_exit(2) system call). The register values given to the agent lwp on creation are typically the registers of the representative lwp, so that the agent lwp can use its stack.

The agent lwp is not allowed to execute any variation of the fork(2), exec(2), or _lwp_create(2) system calls. Attempts to do so yield ENOTSUP to the agent lwp.

PCREAD
PCWRITE

Read or write the target process's address space via a priovec structure operand:

```
typedef struct priovec {
    void *pio_base;       /* buffer in controlling process */
    size_t pio_len;       /* size of read/write request in bytes */
    off_t pio_offset;     /* virtual address in target process */
} priovec_t;
```

These operations have the same effect as pread(2) and pwrite(2), respectively, of the target process's address space file. The difference is that more than one PCREAD or PCWRITE control operation can be written to the control file at once, and they can be interspersed with other control operations in a single write to the control file. This is useful, for example, when planting many breakpoint instructions in the process's address space, or when stepping over a breakpointed instruction. Unlike pread(2) and pwrite(2), no provision is made for partial reads or writes; if the operation cannot be performed completely, it fails with EIO.

PCNICE

The traced process's nice(2) value is incremented by the amount in the operand long. Only a privileged process may better a process's priority in this way, but any user may lower the priority. This operation is not meaningful for all scheduling classes.

Mandatory read access to the file descriptor is required. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

PCSCRED

Set the target process credentials to the values contained in the prcred_t structure operand (see /proc/*pid*/cred). The effective, real, and saved user-IDs and group-IDs

of the target process are set. The target process's supplementary groups are not changed; the pr_ngroups and pr_groups members of the structure operand are ignored. Only a privileged process may perform this operation; for all others it fails with EPERM.

PROGRAMMING NOTES

For security reasons, except for the psinfo, usage, lpsinfo, lusage, lwpsinfo, and lwpusage files, which are world-readable, and except for a privileged process, an open of a /proc file fails unless both the user-ID and group-ID of the caller match those of the traced process and the process's object file is readable by the caller. Except for the world-readable files just mentioned, files corresponding to setuid and setgid processes can be opened only by a privileged process.

Even if held by a privileged process, an open process or lwp file descriptor (other than file descriptors for the world-readable files) becomes invalid if the traced process performs an exec(2) of a setuid/setgid object file or an object file that the traced process cannot read. Any operation performed on an invalid file descriptor, except close(2), fails with EAGAIN. In this situation, if any tracing flags are set and the process or any lwp file descriptor is open for writing, the process will have been directed to stop and its run-on-last-close flag will have been set (see PCSET). This enables a controlling process (if it has permission) to reopen the /proc files to get new valid file descriptors, close the invalid file descriptors, unset the run-on-last-close flag (if desired), and proceed. Just closing the invalid file descriptors causes the traced process to resume execution with all tracing flags cleared. Any process not currently open for writing via /proc, but that has left-over tracing flags from a previous open, and that executes a setuid/setgid or unreadable object file, will not be stopped but will have all its tracing flags cleared.

To wait for one or more of a set of processes or lwps to stop or terminate, /proc file descriptors (other than those obtained by opening the cwd or root directories or by opening files in the fd or object directories) can be used in a poll(2) system call. When requested and returned, either of the polling events POLLPRI or POLLWRNORM indicates that the process or lwp stopped on an event of interest. Although they cannot be requested, the polling events POLLHUP, POLLERR, and POLLNVAL may be returned. POLLHUP indicates that the process or lwp has terminated. POLLERR indicates that the file descriptor has become invalid. POLLNVAL is returned immediately if POLLPRI or POLLWRNORM is requested on a file descriptor referring to a system process (see PCSTOP). The requested events may be empty to wait simply for termination.

FILES

/proc
    directory (list of processes)

/proc/*pid*
    specific process directory

/proc/self
    alias for a process's own directory

/proc/*pid*/as
   address space file

/proc/*pid*/ctl
   process control file

/proc/*pid*/status
   process status

/proc/*pid*/lstatus
   array of lwp status structs

/proc/*pid*/psinfo
   process ps(1) info

/proc/*pid*/lpsinfo
   array of lwp ps(1) info structs

/proc/*pid*/map
   address space map

/proc/*pid*/rmap
   reserved address map

/proc/*pid*/cred
   process credentials

/proc/*pid*/sigact
   process signal actions

/proc/*pid*/auxv
   process aux vector

/proc/*pid*/ldt
   process LDT (IA only)

/proc/*pid*/usage
   process usage

/proc/*pid*/lusage
   array of lwp usage structs

/proc/*pid*/pagedata
   process page data

/proc/*pid*/watch
   active watchpoints

/proc/*pid*/cwd
   symlink to the current working directory

/proc/*pid*/root
   symlink to the root directory

/proc/*pid*/fd
   directory (list of open files)

/proc/*pid*/fd/*
    aliases for process's open files

/proc/*pid*/object
    directory (list of mapped files)

/proc/*pid*/object/a.out
    alias for process's executable file

/proc/*pid*/object/*
    aliases for other mapped files

/proc/*pid*/lwp
    directory (list of lwps)

/proc/*pid*/lwp/*lwpid*
    specific lwp directory

/proc/*pid*/lwp/agent
    alias for the agent lwp directory

/proc/*pid*/lwp/*lwpid*/lwpctl
    lwp control file

/proc/*pid*/lwp/*lwpid*/lwpstatus
    lwp status

/proc/*pid*/lwp/*lwpid*/lwpsinfo
    lwp ps(1) info

/proc/*pid*/lwp/*lwpid*/lwpusage
    lwp usage

/proc/*pid*/lwp/*lwpid*/gwindows
    register windows (SPARC only)

/proc/*pid*/lwp/*lwpid*/xregs
    extra state registers

/proc/*pid*/lwp/*lwpid*/asrs
    ancillary state registers (SPARC V9 only)

SUMMARY OF TRUSTED SOLARIS CHANGES

Appropriate privilege is required to override mandatory access checks. Discretionary access checks have already been performed when the object was opened.

An open(2) by a process with the PRIV_SYS_DEVICES privilege that does not specify O_EXCL succeeds even if an exclusive write open is in effect on the file.

A traced process's nice(2) priority is incremented by the amount contained in the *int* addressed by *p* when the process asserts the PRIV_SYS_CONFIG privilege. This operation is meaningful only when applied to a process in the time-sharing scheduling class.

The following IOCTLs added in the Trusted Solaris environment are used to get information about the security attributes of a process: PIOCLABEL, PIOCCLEAR, PIOCEPRIV, PIOCIPRIV, PIOCPPRIV, PIOCSPRIV, PIOCATTR, PIOCAPSA, and PIOCTCRED.

Trusted Solaris 8 4/01 Reference Manual

chroot(1M), chdir(2), chroot(2), creat(2), exec(2), fork(2), fork1(2), fstat(2), getaudit(2), getdents(2), getpattr(2), kill(2), lseek(2), nice(2), open(2), pread(2), pwrite(2), read(2), readlink(2), readv(2), shmget(2), vfork(2), write(2), writev(2)

SunOS 5.8 Reference Manual

ps(1), _lwp_create(2), _lwp_exit(2), alarm(2), brk(2), close(2), dup(2), fcntl(2), ioctl(2), poll(2), ptrace(2), sigaction(2), sigaltstack(2), wait(2), readdir(3C), siginfo(3HEAD), signal(3HEAD), types32(3HEAD), ucontext(3HEAD)

DIAGNOSTICS

Errors that can occur in addition to the errors normally associated with file system access:

EACCES   The calling process does not have mandatory read access to the file system object. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

ENOENT   The traced process or lwp has terminated after being opened.

EIO   A write(2) was attempted at an illegal address in the traced process.

EBUSY   PCSTOP, PCDSTOP, PCWSTOP, or PCTWSTOP was applied to a system process; an exclusive open(2) was attempted on a /proc file for a process already open for writing; PCRUN, PCSREG, PCSVADDR, PCSFPREG, or PCSXREG was applied to a process or lwp not stopped on an event of interest; an attempt was made to mount /proc when it was already mounted; PCAGENT was applied to a process that was not fully stopped or that already had an agent lwp.

EPERM   Someone other than the process asserting the PRIV_SYS_CONFIG privilege attempted to better a process's priority by issuing PIOCNICE.

ENOSYS   An attempt was made to perform an unsupported operation (such as creat(2), link(2), or unlink(2)).

EINVAL   In general, this means that some invalid argument was supplied to a system call. A non-exhaustive list of conditions eliciting this error includes: a control message operation code is undefined; an out-of-range signal number was specified with PCSSIG, PCKILL, or PCUNKILL; SIGKILL was specified with PCUNKILL; PCSFPREG was applied on a system that does not support floating-point operations; PCSXREG was applied on a system that does not support extra state registers.

ENOMEM	The system-imposed limit on the number of page data file descriptors was reached on an open of /proc/*pid*/pagedata; an attempt was made with PCWATCH to establish more watched areas than the system can support; the PCAGENT operation was issued when the system was out of resources for creating lwps.
E2BIG	Data to be returned in a read(2) of the page data file exceeds the size of the read buffer provided by the caller.
EINTR	A signal was received by the controlling process while waiting for the traced process or lwp to stop via PCSTOP, PCWSTOP, or PCTWSTOP.
EAGAIN	The traced process has performed an exec(2) of a setuid/setgid object file or of an object file that it cannot read; all further operations on the process or lwp file descriptor (except close(2)) elicit this error.
EOVERFLOW	A 32-bit controlling process attempted to read or write the as file or attempted to read the map, rmap, or pagedata file of a 64-bit target process. A 32-bit controlling process attempted to apply one of the control operations PCSREG, PCSXREG, PCSVADDR, PCWATCH, PCAGENT, PCREAD, PCWRITE to a 64-bit target process.

NOTES   For security reasons, a process must have both discretionary and mandatory read and write access to a traced process as well as discretionary and mandatory read access to the process's executable file. Files corresponding to setuid, setguid, and privileged processes (those with permitted privileges) can only be opened by a process which in addition to having discretionary and mandatory read access has asserted the PRIV_PROC_OWNER privilege. If a traced process performs an exec(2), the open process or lwp file descriptor will become invalid if the new object file cannot be read. If a traced process performs an exec(2), the open process or lwp file descriptor will become invalid if the new object file is either a setuid/setgid object file, or will have the use or privileges upon exection. The tracing process may assert the PRIV_PROC_OWNER privilege to override this restriction.

Descriptions of structures in this document include only interesting structure elements, not filler and padding fields, and may show elements out of order for descriptive clarity. The actual structure definitions are contained in <procfs.h>.

BUGS   Because the old ioctl(2)-based version of /proc is currently supported for binary compatibility with old applications, the top-level directory for a process, /proc/*pid*, is not world-readable, but it is world-searchable. Thus, anyone can open /proc/*pid*/psinfo even though ls(1) applied to /proc*pid* will fail for anyone but the owner or a privileged process. Support for the old ioctl(2)-based version of /proc will be dropped in a future release, at which time the top-level directory for a process will be made world-readable.

On SPARC based machines, the types `gregset_t` and `fpregset_t` defined in `<sys/regset.h>` are similar to but not the same as the types `prgregset_t` and `prfpregset_t` defined in `<procfs.h>`.

NAME | prof_attr – profile description database

SYNOPSIS | `/etc/security/prof_attr`

DESCRIPTION | `/etc/security/prof_attr` is a local source for rights profile names, descriptions, and other attributes of profiles. The `prof_attr` file can be used with other profile sources, including the `prof_attr` NIS map and NIS+ table. Programs use the `getprofattr`(3SECDB) routines to gain access to this information.

The search order for multiple `prof_attr` sources is specified in the `/etc/nsswitch.conf` file, as described in the `nsswitch.conf`(4) man page.

A rights profile is a mechanism used to bundle together the commands, CDE actions, and authorizations needed to perform a specific function. A profile can also contain other profiles. Each entry in the `prof_attr` database consists of one line of text containing five fields separated by colons (`:`). Line continuations using the backslash (`\`) character are permitted. The format of each entry is:

profname:*res1*:*res2*:*desc*:*attr*

profname	The name of the profile. Profile names are case-sensitive.
res1	Reserved for future use.
res2	Reserved for future use.
desc	A long description. This field should explain the purpose of the profile, including what type of user would be interested in using it. The long description should be suitable for displaying in the help text of an application.
attr	An optional list of semicolon-separated (`;`) key-value pairs that describe the security attributes to apply to the object upon execution. Zero or more keys may be specified. There are three valid keys: `help`, `profiles`, and `auths`.
	`help` is assigned the name of a file ending in `.htm` or `.html`.
	`auths` specifies a comma-separated list of authorization names chosen from those names defined in the `auth_attr`(4) database. Authorization names may be specified using the asterisk (`*`) character as a wildcard. For example, `solaris.printer.*` would mean all of Sun's authorizations for printing.
	`profiles` specifies a comma-separated list of profile names chosen from those names defined in the `prof_attr` database.

EXAMPLES

EXAMPLE 1 Allowing execution of all commands

The following entry allows the user to execute all commands:

```
All:::Execute any command as the user or role:help=RtAll.html
```

EXAMPLE 2 Consulting the local prof_attr file first

With the following nsswitch.conf entry, the local prof_attr file is consulted before the NIS+ table:

```
prof_attr: files nisplus
```

FILES

/etc/nsswitch.conf

/etc/security/prof_attr

NOTES

When deciding which authorization source to use (see DESCRIPTION), keep in mind that NIS+ provides stronger authentication than NIS.

The root user is usually defined in local databases because root needs to be able to log in and do system maintenance in single-user mode and at other times when the network name service databases are not available. So that the profile definitions for root can be located at such times, root's profiles should be defined in the local prof_attr file, and the order shown in the example nsswitch.conf(4) file entry under EXAMPLES is highly recommended.

Because the list of legal keys is likely to expand, any code that parses this database must be written to ignore unknown key-value pairs without error. When any new keywords are created, the names should be prefixed with a unique string, such as the company's stock symbol, to avoid potential naming conflicts.

Each application has its own requirements for whether the help value must be a relative pathname ending with a filename or the name of a file. The only known requirement is for the name of a file.

The following characters are used in describing the database format and must be escaped with a backslash if used as data: colon (:), semicolon (;), equals (=), and backslash (\).

SUMMARY OF TRUSTED SOLARIS CHANGES

Rights profiles can include CDE actions and other profiles.

auths(1), profiles(1), exec_attr(4), user_attr(4)

getuserattr(3SECDB), auth_attr(4)

NAME | resolv.conf – resolver configuration file

SYNOPSIS | /etc/resolv.conf

DESCRIPTION | The resolver is a set of routines that provide access to the Internet Domain Name System. See resolver(3RESOLV). resolv.conf is a configuration file that contains the information that is read by the resolver routines the first time they are invoked by a process. The file is designed to be human readable and contains a list of keywords with values that provide various types of resolver information.

The resolv.conf file contains the following configuration directives:

nameserver
Specifies the Internet address in dot-notation format of a name server that the resolver is to query. Up to *MAXNS* name servers may be listed, one per keyword. See <resolv.h>. If there are multiple servers, the resolver library queries them in the order listed. If no name server entries are present, the resolver library queries the name server on the local machine. The resolver library follows the algorithm to try a name server until the query times out. It then tries the the name servers that follow, until each query times out. It repeats all the name servers until a maximum number of retries are made.

domain
Specifies the local domain name. Most queries for names within this domain can use short names relative to the local domain. If no domain entry is present, the domain is determined from sysinfo(2) or from gethostname(3C). (Everything after the first '.' is presumed to be the domain name.) If the host name does not contain a domain part, the root domain is assumed. You can use the LOCALDOMAIN environment variable to override the domain name.

search
The search list for host name lookup. The search list is normally determined from the local domain name. By default, it contains only the local domain name. You can change the default behavior by listing the desired domain search path following the search keyword, with spaces or tabs separating the names. Most resolver queries will be attempted using each component of the search path in turn until a match is found. This process may be slow and will generate a lot of network traffic if the servers for the listed domains are not local. Queries will time out if no server is available for one of the domains.

The search list is currently limited to six domains and a total of 256 characters.

sortlist*addresslist*

Allows addresses returned by the libresolv-internal gethostbyname() to be sorted. A sortlist is specified by IP address netmask pairs. The netmask is optional and defaults to the natural netmask of the net. The IP address and optional network pairs are separated by slashes. Up to 10 pairs may be specified. For example:

sortlist 130.155.160.0/255.255.240.0 130.155.0.0

options

Allows certain internal resolver variables to be modified. The syntax is

options option ... where option is one of the following:

debug
   Sets RES_DEBUG in the _res.options field.

ndots:*n*
   Sets a threshold floor for the number of dots which must appear in a name given to res_query() before an initial absolute (as-is) query is performed. See resolver(3RESOLV). The default value for *n* is 1, which means that if there are any dots in a name, the name is tried first as an absolute name before any search list elements are appended to it.

timeout:*n*
retrans:*n*
   Sets the amount of time the resolver will wait for a response from a remote name server before retrying the query by means of a different name server. Measured in seconds, the default is RES_TIMEOUT. See <resolv.h>. The timeout and retrans values are the starting point for an exponential back off procedure where the timeout is doubled for every retransmit attempt.

attempts:*n*
retry:*n*
   Sets the number of times the resolver will send a query to its name servers before giving up and returning an error to the calling application. The default is RES_DFLRETRY. See <resolv.h>.

rotate

  Sets RES_ROTATE in _res.options. The name
servers are queried round-robin from among those
listed. The query load is spread among all listed
servers, rather than having all clients try the first
listed server first every time.

no-check-names

  Sets RES_NOCHECKNAME in _res.options. This
disables the modern BIND checking of incoming
host names and mail names for invalid characters
such as underscore (_), non-ASCII, or control
characters.

inet6

  Sets RES_USE_INET6 in _res.options. In the
Solaris BIND port, this has no effect on
gethostbyname(3NSL). To retrieve IPv6 addresses
or IPv4 addresses in mapped form, use
getipnodebyname(3SOCKET) instead of setting
inet6.

The domain and search keywords are mutually exclusive. If more than one instance
of these keywords is present, the last instance takes precedence

You can override the search keyword of the system resolv.conf file on a
per-process basis by setting the environment variable LOCALDOMAIN to a
space-separated list of search domains.

You can amend the options keyword of the system resolv.conf file on a
per-process basis by setting the environment variable RES_OPTIONS to a
space-separated list of resolver options.

The keyword and value must appear on a single line. Start the line with the keyword,
for example, nameserver, followed by the value, separated by white space.

To protect /etc/resolv.conf from unauthorized modification, it must have a
sensitivity label of ADMIN_LOW. The DNS name servers specified in these files can
reside on either Trusted Solaris hosts or non-trusted hosts. Administrators are advised
to configure only DNS name servers on Trusted Solaris hosts in the
/etc/resolv.conf file.

SUMMARY OF TRUSTED SOLARIS CHANGES

The /etc/resolv.conf file must have a sensitivity label of ADMIN_LOW.

FILES

/etc/resolv.conf   resolver configuration file

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Standard BIND 8.2.2

`in.named`(1M), `sysinfo`(2), `resolver`(3RESOLV)

`domainname`(1M), `gethostbyname`(3NSL), `getipnodebyname`(3SOCKET),
`gethostname`(3C)

Vixie, Paul;Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for
BIND*(public domain), Internet Software Consortium, 1996.

NAME | rmtab – Remote mounted file system table

SYNOPSIS | `/etc/rmtab`

DESCRIPTION | rmtab contains a table of file systems that are remotely mounted by NFS clients. This file is maintained by mountd(1M), the mount daemon. The data in this file should be obtained only from mountd(1M) using the MOUNTPROC_DUMP remote procedure call.

The file contains a line of information for each remotely mounted file system. There are a number of lines of the form:

> *hostname* : *fsname*

The mount daemon adds an entry for any client that successfully executes a mount request and deletes the appropriate entries for an unmount request.

Lines beginning with a hash ('#') are commented out. These lines are removed from the file by mountd(1M) when it first starts up. Stale entries may accumulate for clients that crash without sending an unmount request.

SUMMARY OF TRUSTED SOLARIS CHANGES | The `/etc/rmtab` file must have a sensitivity label of ADMIN_LOW and be owned by UID 0.

`/etc/rmtab`   Remote mounted file system table.

Trusted Solaris 8 4/01 Reference Manual | mountd(1M), showmount(1M)

NAME | sel_config – selection rules for copy, cut, paste, drag and drop operations

SYNOPSIS | /usr/dt/config/sel_config

DESCRIPTION | The sel_config file specifies how the system behaves when a user performs cut-and-paste, copy-and-paste, and drag-and-drop operations on data between windows that have different sensitivity label. There are two types of entries in this file: automatic confirmation and automatic reply.

Automatic Confirmation | This type of entry specifies whether a confirmation window (the selection confirmer) displays. Each entry has the form:

relationship: *confirmation*

relationship identifies the result of comparing the selected data's source and destination windows' SLs. There are 3 allowed values:

upgradesl	The source window's sensitivity label is less than the destination window's label.
downgradesl	The source window's sensitivity label is higher than the destination window's label.
disjointsl	The source and destination windows' sensitivity labels are disjoint (neither dominates the other).

confirmation specifies whether to perform automatic confirmation. Allowed values are:

y	Use automatic confirmation (that is, do not display the selection confirmer window).
n	Use manual confirmation (that is, display the selection confirmer window). This is the default.

Automatic Reply | This set of entries provides a means to reduce the number of confirmations that are required of the user, since a single user operation may involve several flows of information between the source and destination windows.

There must be one entry of this form:

autoreply: *value*

If *value* is y (for yes), then the remaining entries of the set are used as attributes for the selection data (rather than the actual contents) to complete the operation without confirmation. If *value* is n (for no), then the remaining entries are ignored.

Defaults can be specified for any *type* field that appears in the Confirmer window. Below are some examples entries for defaults.

replytype: TARGETS
replytype: Pixel Sets
replytype: LENGTH

replytype: Type Of Monitor

The TARGETS entry, when used, returns the list of target atoms that are supported by the source window. The Pixel Sets and Type Of Monitor entries, are used for animation during a drag-and-drop operation. The LENGTH entry specifies the number of bytes in the selection.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu

Trusted Solaris 8 4/01 Reference Manual *Trusted Solaris administrator's document set*

SunOS 5.8 attributes(5)

NAME | shadow – shadow password file

DESCRIPTION | /etc/shadow is an access-restricted ASCII system file that stores users' encrypted passwords and related information. The shadow file can be used in conjunction with other shadow sources, including the NIS maps passwd.byname and passwd.byuid and the NIS+ table passwd. Programs use the getspnam(3C) routines to access this information.

The fields for each user entry are separated by colons. Each user is separated from the next by a newline. Unlike the /etc/passwd file, /etc/shadow does not have general read permission.

Each entry in the shadow file has the form:

username:*password*:*lastchg*: *min*:*max*:*warn*: *inactive*:*expire*:*flag*The fields are defined as follows:

username	The user's login name (UID).
password	A 13-character encrypted password for the user, a *lock* string to indicate that the login is not accessible, or no string, which shows that there is no password for the login.
lastchg	The number of days between January 1, 1970, and the date that the password was last modified.
min	The minimum number of days required between password changes.
max	The maximum number of days the password is valid.
warn	The number of days before password expires that the user is warned.
inactive	The number of days of inactivity allowed for that user.
expire	An absolute date specifying when the login may no longer be used.
flag	Used to keep a count of the bad passwords entered by the account. If the correct password is entered, or if a new password is assigned to the account, the count is reset to 0. If the count exceeds the maximum number of bad passwords allowed at the site, the account is locked with the string *LK* entered in the status field of the account's passwd(4) entry. An administartor can open a locked account by assigning a new password to the account to reset the count to zero (0). The *flag* field only works for files and NIS+.

The encrypted password consists of 13 characters chosen from a 64-character alphabet (., /, 0–9, A–Z, a–z). To update this file, use the passwd(1), and smuser(1M) or smrole(1M) commands.

In order to make system administration manageable, /etc/shadow entries should appear in exactly the same order as /etc/passwd entries; this includes "+" and "-" entries if the compat source is being used (see nsswitch.conf(4)).

SUMMARY OF TRUSTED SOLARIS CHANGES

In Trusted Solaris 8 and later releases, the *flag* field is used for files and NIS+.

FILES

/etc/shadow	shadow password file
/etc/passwd	password file
/etc/nsswitch.conf	name-service switch configuration file
/etc/user_attr	extended user attributes database

Trusted Solaris 8 4/01 Reference Manual

SEE ALSO

login(1), passwd(1), smrole(1M), smuser(1M), nsswitch.conf(4), user_attr(4)

getspnam(3C), putspent(3C), passwd(4)

NOTES

If password aging is turned on in any name service the *passwd:* line in the /etc/nsswitch.conf file must have a format specified in the nsswitch.conf(4) man page.

If the /etc/nsswitch.conf passwd policy is not in one of the supported formats, logins will not be allowed upon password expiration because the software does not know how to handle password updates under these conditions. See nsswitch.conf(4) for additional information.

NAME | sharetab – Shared file system table

DESCRIPTION | `sharetab` resides in directory `/etc/dfs` and contains a table of local resources shared by the `share` command.

Each line of the file consists of the following fields:

pathname resource fstype specific_options description

where

pathname	Indicate the path name of the shared resource.
resource	Indicate the symbolic name by which remote systems can access the resource.
fstype	Indicate the file system type of the shared resource.
specific_options	Indicate filesystem-type-specific options that were given to the `share` command when the resource was shared.
description	Describe the shared resource provided by the system administrator when the resource was shared.

SUMMARY OF TRUSTED SOLARIS CHANGES | The `/etc/dfs/sharetab` file must have a sensitivity label of `ADMIN_LOW` and be owned by UID 0.

FILES | `/etc/dfs/sharetab`   Shared file system table.

Trusted Solaris 8 4/01 Reference Manual | `share(1M)`

NAME | tndlog – log of tnd debugging information

SYNOPSIS | /var/tsol/tndlog

DESCRIPTION | /var/tsol/tndlog is the default log file for debugging tnd(1M). This file contains one record for each debugging message. Each record contains the debugging message and time.

tndlog is a text file. Each field within each entry is separated from the next by a colon. Each entry is separated from the next by a new line.

By default, tndlog does not exist, so no logging is done. To enable logging, tnd must be started with a debug level, or tnctl(1M) must be used to turn on debugging.

FILES | /var/tsol/tndlog   Log of tnd debugging information

Trusted Solaris 8 4/01 Reference Manual | tnctl(1M), tnd(1M)

NAME | tnidb – trusted network interface-control database

SYNOPSIS | `/etc/security/tsol/tnidb`

DESCRIPTION | The `tnidb` database specifies the accreditation range and default security attributes for each network interface. The following set of default attributes applies to any network interface that does not have an entry in this file:

```
min_sl=ADMIN_LOW;max_SL=ADMIN_HIGH;def_label=ADMIN_LOW;
def_cl=ADMIN_HIGH;forced_privs=empty;
```

Each entry in the interface database consists of one long line, with fields of the entry separated by semicolons (;):

interface_name:*field1*;*field2*;*field3*;*fieldn*;

A pound sign (#) as the first character of a line indicates a comment line, which is ignored. Each entry consists of a line of this form:

interface_name:`min_sl=`*value*;`max_sl=`*value*;`def_label=`*value*;
`def_cl=`*value*;`forced_privs=`*value*;

Note – The width of this man page prevents showing the foregoing entry on a single line. However, each entry in the database *must* be a single line.

The first field for each entry is the interface name. Each entry must contain valid specifications for the accreditation range of the interface for all enforceable security attributes. All fields are mandatory; each entry contains these fields:

`min_sl`, `max_sl` | Specify the accreditation range of the interface. Only packets with a sensitivity label within the specified accreditation range are allowed into or out of the interface. For a configuration that allows for traffic at all labels, the range should be `admin_low` (in hex) to `admin_high` (in hex).

`def_label` | Apply this default label to a packet received from an approved remote host that does not support mandatory access control. Under these conditions, all packets imported from the interface that are not labeled with a sensitivity label are assigned this default label.

`def_cl` | Apply this default clearance to a packet received from an approved remote host that does not support mandatory access control.

`forced_privs` | Define the effective privileges to be applied to the incoming packet received from a host that does not support privileges. The format of the privilege set is:

forced_privs=priv[,priv][...] | none | empty | all where

priv | The text string (such as `net_mac_read`) for privilege. (`forced_privs=net_mac_read`)

none	Apply no privileges. (forced_privs=none)
empty	Apply no privileges. (forced_privs=empty)
all	Apply all privileges. (forced_privs=all)

Any default label, clearance, and the forced privilege values specified in trusted network databases apply only on incoming packets that do not have the attributes.

Any values for a remote host specified through tnrhdb(4) or tnrhtp(4) entries take precedence over values specified in this database for the network interface through which the remote host is accessed.

All labels are specified in their hex format.

If this database is modified while the network is up, the changes do not take effect until tnctl(1M) updates the interface entries.

Errors in the format of this file can be detected by tnchkdb(1M), which should be run on each database once it has been created or modified. (Refer to the tnchkdb man page for more information.)

The /etc/security/tsol/tnidb file is protected at label admin_low with permission bits 444, owner root, and group sys.

ATTRIBUTES   See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsr

EXAMPLES   **EXAMPLE 1** Sample interface entries

For the sake of clarity on this man page, examples are shown using a continuation character (\). In the database file, however, the backslash is not permitted because each entry is made on a single line.

```
#
# Sample interface entries.
#
lo0:min_sl=0x00000000000000000000000000000000000000000000000
00000000000000000000000;\
max_sl=0x7fffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffff;\
def_label=0x00040c000000000000000000000000000000000000000000
00000000ffffffffffffffff;\
def_cl=0x00060000000000000000000000000000000000000000000000
0000000ffffffffffffffff;\
forced_privs=none;
 # Note that default values are not necessary for lookback interfaces
 # because ALL attributes are to accompany the data, and default values
 # are only for unlabeled hosts.
```

EXAMPLE 1 Sample interface entries   *(Continued)*

```
 #
 #
le0:min_sl=0x00000000000000000000000000000000000000000000000
00000000000000000000000;\
max_sl=0x0006000000000000000000000000000000000000000000000
0000000ffffffffffffffff;\
def_label=0x00040c0000000000000000000000000000000000000000
00000000ffffffffffffffff;\
def_cl=0x000600000000000000000000000000000000000000000000000
0000000ffffffffffffffff;\
forced_privs=none;
le1:min_sl=0x00000000000000000000000000000000000000000000000
00000000000000000000000;\
max_sl=0x0006000000000000000000000000000000000000000000000
0000000ffffffffffffffff;\
def_label=[0x00040c0000000000000000000000000000000000000000
00000000ffffffffffffffff];\
def_cl=0x000600000000000000000000000000000000000000000000000
0000000ffffffffffffffff;\
forced_privs=none;
```

This sample accreditation range for interfaces `le0` and `le1` specifies that only packets with a sensitivity label that dominates `admin_low` and is dominated by TS NATIONALITY: CNTRY1/CNTRY2 are allowed into or out of the interface through those interfaces.

Note that interpretations vary by definitions in the `label_encodings`(4) file.

FILES   `/etc/security/tsol/tnidb`   Trusted network interface-control database

`tnd`(1M), `tnctl`(1M), `tnchkdb`(1M), `tnrhdb`(4)

`attributes`(5)

NOTES   A physical network interface, for example `hme0`, can be associated with multiple logical interfaces, for example `hme0:1`, `hme0:2`, each of which can have a database entry.

Since the colon (:) character is a database separation character, the logical interface names such as `hme0:1` must be escaped with a backslash (\) , as in `hme0\:1`.

A Trusted Solaris system acting as an intermediate router always uses the default label of the physical interface when applying a default label to a packet. This is true even if the physical interface, for example `hme0`, is associated with multiple logical interfaces, for example `hme0:1` and `hme0:2`, each of which may have a different default label. In all other cases, the Trusted Solaris system uses the default label of the correct logical interface.

tnidb(4)

WARNINGS For proper functioning, the loopback and primary interface need the `min_sl` to be `admin_low` (in hex) and the `max_sl` to be `admin_high` (in hex).

NAME | tnrhdb – trusted network remote-host database

SYNOPSIS | `/etc/security/tsol/tnrhdb`

DESCRIPTION | The `tnrhdb` database specifies which remote-host template to use for each host, including the local host, in the distributed system. `tnrhdb` works together with the `tnrhtp`(4) database in allowing the administrator to establish the security and network accreditation attributes for each host. The trusted-network software uses a network "longest prefix of matching bits" mechanism in looking for a `tnrhdb` entry for a host. The software looks first for an entry specific to the host; if it does not find one, the software falls back to searching for an entry with the longest prefix of a matching bit pattern, and so on.

Using this mechanism, an IPv4 wildcard entry (IPv4 address `0.0.0.0`) has a prefix length of 0 and hence can match any IPv4 address. If a host's IP address cannot be matched to some entry in the `tnrhdb` database, communication with the host is not permitted.

Each entry consists of a line of this form:

IP_address:template_name

IP_address   This field is the IP address of the host or network that has the security properties specified by the *template_name* defined in the `tnrhtp` database.

An entry can either be an IPv4 or IPv6 address of a host (for example, `10.100.100.201` or `fec0\:\:9\:20ff\:fea0\:21f7`), or a wildcard IPv4 or IPv6 address of a subnet. An IPv4 wildcard address can be either in the form of a class A, B, or C address (`10.100.0.0`) or a subnet_address with a prefix length (`10.100.128.0/17`). An IPv6 wildcard entry is a subnet address with a prefix length (`fec0\:\:/10`).

Any colon (:) character in an IPv6 address must be escaped with a backslash (\), as in `fec0\:\:a00\:20ff\:fea0\:21f7`.

template_name   This value must be a valid template name in the `tnrhtp` database. See man pages for `tnrhtp`(4) for information on the security attributes.

More than one IP address can use the same template. If this database is modified while the network is up, the changes do not take effect until after `tnctl`(1M) is used to update the remote-host entries. Administrators are allowed to add new entries and modify existing entries while network is up. The *template_name* field cannot contain any white spaces.

Errors in the format of this file can be detected by running tnchkdb, which should be run every time the database is modified or created. Refer to the tnchkdb(1M) man page for more information.

The /etc/security/tsol/tnrhdb file is protected at label ADMIN_LOW with permission bits 444, owner root, and group sys.

ATTRIBUTES   See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsr

EXAMPLES   **EXAMPLE 1** A Sample tnrhdb

The templates in the following example are first defined in the tnrhtp, then used in the tnrhdb file. The example shows a host that uses template ripso_secure_route, a host that uses template tsol, a subnet that uses template tsol, a subnet that uses template secret; and every other host uses the default_template template specified in the wildcard entry.

```
#
# Assume that templates default_template, tsol, secret, and
# ripso_secure_route are defined in the tnrhtp database.
#
# the first two entries are addresses of the IPv4 and
# IPv6 loopback interfaces
  127.0.0.1:tsol
  \:\:1:tsol
  10.0.0.1:tsol
  192.168.120.6:tsol
  192.168.120.0:tsol
  192.168.120.7:ripso_secure_route
  192.168.121.0:secret
  0.0.0.0:default_template
  fec0\:\:a00\:20ff\:fea0\:21f7:tsol
```

FILES   /etc/security/tsol/tnrhdb   Trusted network remote-host database

Trusted Solaris 8   tnd(1M), tnchkdb(1M), tnctl(1M), tnidb(4), tnrhtp(4)
4/01 Reference
Manual   hosts(4), ipnodes(4), attributes(5)
SunOS 5.8
Reference Manual
WARNINGS   For proper functioning, the primary host name must point to a template that has min_sl=ADMIN_LOW (in hex) and max_sl=ADMIN_HIGH (in hex).

Changing a template while the network is up can change the security view of an undetermined number of hosts.

NOTES The colon (:) character is a database separation character, so it must be escaped with a backslash (\) if used as part of a data field, as in `fec0\:\:a00\:20ff\:fea0\:21f7`.

The administrator may wish to make one `tnrhdb` entry for each host running the Trusted Solaris release, and make one subnet entry that applies to all unlabeled hosts that have the same security attributes. Then, the administrator may make a separate entry for each host that must be assigned a different set of security attributes.

The template assigned to any of a Trusted Solaris host's interface addresses must be a sun_tsol type. Trusted Solaris is by default shipped with the `tsol`, `tsol_cipso`, and `tsol_ripso` sun_tsol template types.

NAME	tnrhtp – trusted network remote-host templates
SYNOPSIS	`/etc/security/tsol/tnrhtp`
DESCRIPTION	The `tnrhtp` database of templates is specified by the administrator for convenience when assigning accreditation and security attributes for each host in the distributed system, including the local host and network. `tnrhtp` works together with `tnrhdb`(4); IP addresses in `tnrhdb` can be assigned only to templates defined in the `tnrhtp` database. The administrator should run `tnchkdb`(1M) to check the syntax after each modification to the `tnrhtp` database.

Each entry in the template database is formed as one long line, with fields of the entry separated by semicolons (;):

template_name: *field_name=value*;[*field_name=value*; ...]

A pound sign (#) as the first character of a line indicates a comment line, which is ignored.

The following host types are currently supported: `unlabeled`, `sun_tsol`, `ripso`, `cipso`, and `tsix`.

All fields of a particular *host_type* are mandatory unless otherwise indicated even if no value is set other than `none`. If this database is modified while the network is up, the changes do not take effect immediately unless `tnctl`(1M) is used to update the template entries; otherwise, the changes take effect when next polled by the trusted network daemon, `tnd`(1M). Administrators are allowed to add new templates and modify attributes of existing templates while the network is up.

The `/etc/security/tsol/tnrhtp` file is protected at label `ADMIN_LOW` with permission bits `444`, owner `root`, and group `sys`.

When specifying a name for a template, note that only the first 31 characters of the template name are read and interpreted. You can use any printable character in a template name except for field delimiters, newline, or the comment character.

Trusted Solaris 8 and later releases extend the use of the domain of interpretation notion to all template types. The domain of interpretation defines the set of rules for translating between the external or internal representation of the security attributes and their network representation. Trusted Solaris systems that have the same domain of interpretation share that set of rules. They also share the same interpretation for the default attributes assigned to the unlabeled templates that have that same domain of interpretation.

Template for unlabeled Hosts	The template for the `unlabeled` host type has these fields:
	template_name Specify a name for the template.
	`host_type` `unlabeled`

doi   This is the domain of interpretation for `def_label` and `def_cl`
                       fields.

`def_label, def_cl`
Define the default attributes to be applied to incoming data from the remote hosts
that do not support these attributes. These defaults override the defaults specified
for an interface in the `tnidb(4)` database.

`min_sl, max_sl`
Specify the accreditation range for unlabeled gateways of this template. The format
is the same as that in the `tnidb(4)` database. All labels are specified in their hex
format.

`forced_privs`
Define the effective privileges to be applied to the incoming packet received from a
host that does not support privileges. The format of the privilege set is:

`forced_privs=`*priv*`[,`*priv*`] [...]|none|empty|all` where

priv   The text string (such as `net_mac_read`) for privilege.
                (`forced_privs=net_mac_read`)

none   Apply no privileges. (`forced_privs=none`) .

empty   Take the default from `tnidb(4)`. (`forced_privs=empty`)

all   Apply all privileges. (`forced_privs=all`)

`ip_label`
(Optional) Provide for IP labeling. When present, packets coming from hosts of this
template are labeled using the IP option specified by `ip_label`. The format of the
label is:

`[ip_label=cipso|ripso|none|empty]`

**Template for
sun_tsol Hosts**
Host type `sun_tsol` has these fields:

template_name   Specify a name for the template.

`host_type`   `sun_tsol`

`doi`   This number is the domain of interpretation.

`min_sl, max_sl`
Specify the accreditation range for the remote hosts using this template. The format
is the same as that in the `tnidb(4)` database. All labels are specified in their hex
format.

`allowed_privs`
Limit the effective privilege set for an incoming packet. If a source host associated
with this template sends a packet to a destination host, the destination will limit the
privilege set of the incoming packet to that specified in this field. The format of the
privilege set is:

```
allowed_privs=priv[,priv][...]|none|empty|all
```
where

priv The text string (such as `net_mac_read`) for privilege. (`allowed_privs=net_mac_read`)

none Apply no privileges. (`allowed_privs=none`)

empty Take the default from tnidb(4). (`allowed_privs=empty`)

all Apply all privileges. (`allowed_privs=all`)

`ip_label`
Provide for IP labeling. These are valid types for `ip_label`:

none `ripso` and `cipso` options are not used to label data sent to the host. However, `ripso` and `cipso` security options may be sent to the host if the host is acting as a gateway.

ripso For hosts that label their packets with the Revised IP Security Option per RFC 1108. If `ripso` is selected for a host, the `ripso_label` and `ripso_error` fields are required.

cipso For hosts that label their packets according to the Common IP Security Options (Tag Type 1 only) as detailed by the Trusted Systems Interoperability Group (TSIG). If `ip_label` is set to `cipso`, then packets for which the host is the final destination will be labeled with a `CIPSO` label containing the specified `doi`. If the host is configured as a gateway, then the host will be able to route `CIPSO`-labeled packets containing the specified `doi`.

`ripso_label`
If `ip_label` is set to `ripso`, then packets for which the host is the final destination will be labeled with the specified RIPSO label. If the host is configured as a gateway, then the host will be able to route packets with the specified RIPSO label.

If `ip_label` is set to `none` and `ripso_label` is set, then the host will be able to forward packets labeled with the specified RIPSO label even though packets addressed to the host will not contain a RIPSO label.

Set this field explicitly to `empty` if no value is to be assigned.

A `ripso_label` is made up of a classification level followed by a protection authority flag. The supported classification levels are: `TOP_SECRET`, `SECRET`, `CONFIDENTIAL`, `UNCLASSIFIED` or a hexadecimal representation, The supported protection authority flags are: `GENSER`, `SIOP-ESI`, `SCI`, `NSA`, `DOE`, or a hexadecimal representation.

`ripso_error`
These are the protection authority flags that are used to label ICMP messages generated in response to incoming `RIPSO`-labeled packets: `GENSER`, `SIOP-ESI`, `SCI`, `NSA`, `DOE`, or a hexadecimal representation. The classification level is taken

from the `ripso_label` field. The sender's template is always used when labeling ICMP error messages with RIPSO labels.

This field can take multiple values; these must be separated by commas.

Set this field explicitly to `empty` if no value is to be assigned.

Template for ripso Hosts

The template for `ripso` host type is for non-sun_tsol hosts that label packets with the RIPSO basic security option. This template has these fields:

template_name   Specify a name for the template.

`host_type`   `ripso`

`doi`   (Optional) This number is the domain of interpretation. It applies to the `def_label` and `def_cl` fields.

`ripso_label`
  A `ripso_label` is made up of a classification level followed by a protection authority flag. The supported classification levels are: `TOP_SECRET`, `SECRET`, `CONFIDENTIAL`, `UNCLASSIFIED` or a hexadecimal representation, The supported protection authority flags are: `GENSER`, `SIOP-ESI`, `SCI`, `NSA`, `DOE`, or a hexadecimal representation.

`ripso_error`
  These are the protection authority flags that are used to label ICMP messages generated in response to incoming RIPSO-labeled packets.

  This field can take multiple values; these must be separated by commas.

`def_label`, `def_cl`
  Define the default attributes to be applied to incoming data from the remote hosts that do not support these attributes. These defaults override the defaults specified for an interface in the `tnidb`(4) database.

  Set this field explicitly to `empty` if no value is to be assigned.

  Default labels are not required for the remote-host entry if there are interface defaults that would be the same for the remote host.

`min_sl`, `max_sl`
  Specify the accreditation range for the remote host gateway using this template. The format is the same as that in the `tnidb`(4) database. All labels are specified in their hex format.

`forced_privs`
  Define the effective privileges to be applied to the incoming packet received from a host that does not support privileges. Having no privileges specified is *not* the same as specifying the word `none`. The format of the privilege set is:

  `forced_privs=`*priv*`[,`*priv*`][...]|none|empty|all`
  where

priv	The text string (such as net_mac_read) for privilege. (forced_privs=net_mac_read)
none	Apply no privileges. (forced_privs=none)
empty	Take the default from tnidb(4). (forced_privs=empty)
all	Apply all privileges. (forced_privs=all)

Template for cipso Hosts

The template for cipso host type is for hosts that use CIPSO (Common IP Security Options — Tag Type 1 only) to label packets. This template has these fields:

template_name	Specify a name for the template.
host_type	cipso
doi	This number is the domain of interpretation. It is used in the CIPSO label.

min_sl, max_sl
  Specify the accreditation range for the remote hosts using this template. The format is the same as that in the tnidb(4) database. All labels are specified in their hex format.

def_label, def_cl
  Define the default attributes to be applied to incoming data from the remote hosts that do not support these attributes. These defaults override the defaults specified for an interface in the tnidb(4) database.

forced_privs
  Defines the effective privileges to be applied to the incoming packet received from a host that does not support privileges. Having no privileges specified is *not* the same as specifying the word none. The format of the privilege set is:

  forced_privs=*priv*[,*priv*][...]|none|empty|all
  where

priv	The text string (such as net_mac_read) for privilege. (forced_privs=net_mac_read)
none	Apply no privileges. (forced_privs=none)
empty	Take the default from tnidb(4). (forced_privs=empty)
all	Apply all privileges. (forced_privs=all)

Template for tsix Hosts

The template for tsix host type is for hosts that use TSIX(RE) 1.1 protocols with token mapping to label packets. This template has these fields:

template_name	Specify a name for the template.
host_type	tsix
doi	This number is the domain of interpretation.

min_sl, max_sl
    Specify the accreditation range for the remote hosts using this template.

    All labels are specified in their hex format.

allowed_privs
    Limit the effective privilege set for an incoming packet. If a source host associated
    with this template sends a packet to a destination host, the destination will limit the
    privilege set of the incoming packet to that specified in this field. The format of the
    privilege set is:

    `allowed_privs=priv[,priv][...]|none|empty|all`
    where

    priv   The text string (such as `net_mac_read`) for privilege.
               (`allowed_privs=net_mac_read`)

    none   Apply no privileges. (`allowed_privs=none`)

    empty   Take the default from `tnidb(4)`. (`allowed_privs=empty`)

    all   Apply all privileges. (`allowed_privs=all`)

forced_privs
    Define the effective privileges to be applied to the incoming packet received from a
    host that is not supplying privileges. Having no privileges specified is *not* the same
    as specifying the word none. The format of the privilege set is:

    `forced_privs=priv[,priv][...]|none|empty|all`
    where

    priv   The text string (such as `net_mac_read`) for privilege.
               (`forced_privs=net_mac_read`)

    none   Apply no privileges. (`forced_privs=none`)

    empty   Take the default from `tnidb(4)`. (`forced_privs=empty`)

    all   Apply all privileges. (`forced_privs=all`)

def_label, def_cl
    Define the default attributes to be applied to incoming data from the remote hosts
    that are not supplying these attributes. These defaults override the defaults
    specified for an interface in the `tnidb(4)` database.

    Default labels are not required for the remote-host entry if there are interface
    defaults that would be the same for the remote host.

ip_label
    Provide for IP labeling. These are valid types for `ip_label`:

    none   ripso and cipso options are not used to label data sent to the host.
              However, ripso and cipso security options may be sent to the host if the
              host is acting as a gateway.

ripso For hosts that label their packets with the Revised IP Security Option per RFC 1108. If RIPSO is selected for a host, the ripso_label field is required.

cipso For hosts that label their packets according to the Common IP Security Options (Tag Type 1 only) as detailed by the Trusted Systems Interoperability Group (TSIG).

ripso_label

If ip_label is set to ripso, then packets for which the host is the final destination will be labeled with the specified RIPSO label. If the host is configured as a gateway, then the host will be able to route packets with the specified RIPSO label.

If set to none and ripso_label is set, then the host will be able to forward packets labeled with the specified RIPSO label even though packets addressed to the host will not contain a RIPSO label.

A ripso_label is made up of a classification level followed by a protection authority flag. The supported classification levels are: TOP_SECRET, SECRET, CONFIDENTIAL, UNCLASSIFIED or a hexadecimal representation, The supported protection authority flags are: GENSER, SIOP-ESI, SCI, NSA, DOE, or a hexadecimal representation.

ripso_error

These are the protection authority flags that are used to label ICMP messages generated in response to incoming RIPSO-labeled packets. These are supported protection authority flags: GENSER, SIOP-ESI, SCI, NSA, DOE. The classification level is taken from the ripso_label field. The sender's template is always used when labeling ICMP error messages with RIPSO labels.

This field can take multiple values; these must be separated by commas.

If you do not want to assign a value, you must set this field equal to empty.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsr

EXAMPLES **EXAMPLE 1** Unlabeled Hosts

For the sake of clarity on this man page, examples are shown using a continuation character (\). In the database file, however, the backslash is not permitted because each entry is made on a single line.

```
# Sample ADMIN_LOW template entry for machines or networks.
# Note that the doi field is required.
#
admin_low:host_type=unlabeled;\
```

EXAMPLE 1 Unlabeled Hosts   *(Continued)*

```
def_label=[0x0000000000000000000000000000000000000000000000
00000000000000000000000];\
def_cl=0x000000000000000000000000000000000000000000000000000
00000000000000000;\
forced_privs=empty;\
min_sl=0x00000000000000000000000000000000000000000000000000
0000000000000000;\
max_sl=0x7fffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffff;\
doi=0;\
ip_label=none;\
ripso_label=empty;\
ripso_error=empty;
```

Unless the label at which you want to communicate with an unlabeled host is
ADMIN_LOW, you should not use the above template. A template matching an entry in
your label encodings file, similar to the following example that matches an entry in the
sample label_encodings file, should be used.

```
# Sample UNCLASSIFIED template entry
# based on the sample label_encodings file.
#
unclassified:host_type=unlabeled;\
def_label=[0x0001000000000000000000000000000000000000000000
00000000000000000000000];\
def_cl=0x00040c0000000000000000000000000000000000000000000000
003fffffffffff0000;\
forced_privs=empty;\
min_sl=0x00000000000000000000000000000000000000000000000000
0000000000000000;\
max_sl=0x7fffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffff;\
doi=0;\
ip_label=none;\
ripso_label=empty;\
ripso_error=empty
```

EXAMPLE 2 Sun TSOL Hosts

```
# A sample tnrhtp template entry for sun_tsol hosts or networks.
# Note that the doi field is required.
#
tsol:host_type=sun_tsol;\
min_sl=0x00000000000000000000000000000000000000000000000000
0000000000000000;\
max_sl=0x7fffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffff;\
allowed_privs=all;\
ip_label=none;\
ripso_label=empty;\
ripso_error=empty;\
doi=0;
```

EXAMPLE 3 Sun TSOL and RIPSO

```
# A sample tnrhtp template entry for sun_tsol hosts
# or networks that label packets with the RIPSO security option.
#
tsol_ripso:host_type=sun_tsol;\
min_sl=0x000000000000000000000000000000000000000000000000
00000000000000000;\
max_sl=0x7ffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffff;\
allowed_privs=all;\
ip_label=ripso;\
ripso_label=0x3d 0x20000000;\
ripso_error=0x80000000;\
doi=0;
```

EXAMPLE 4 Sun TSOL and CIPSO

```
# A sample tnrhtp template entry for sun_tsol hosts
# or networks that label packets with the CIPSO security option.
#
tsol_cipso:host_type=sun_tsol;\
min_sl=0x000000000000000000000000000000000000000000000000
00000000000000000;\
max_sl=0x7ffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffff;\
allowed_privs=all;\
ip_label=cipso;\
ripso_label=empty;\
ripso_error=empty;\
doi=1;
```

EXAMPLE 5 RIPSO Security Option

```
# A sample tnrhtp template entry for ripso hosts
# or networks that label packets with the RIPSO security option.
#
ripso_top_secret:host_type=ripso;\
ripso_label=0x3d 0x20000000;\
ripso_error=0x80000000;\
def_label=[0x00060c0000000000000000000000000000000000000000
00003fffffffffffff0000];\
def_cl=[0x00060c000000000000000000000000000000000000000000
00003fffffffffffff0000];\
forced_privs=empty;\
min_sl=0x000000000000000000000000000000000000000000000000
00000000000000000;\
max_sl=0x7ffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffff;\
doi=0;
```

EXAMPLE 6 CIPSO Security Option

```
# A sample tnrhtp template entry for cipso hosts
# or networks that label packets with the CIPSO security option.
```

EXAMPLE 6 CIPSO Security Option   *(Continued)*

```
#
cipso:host_type=cipso;\
doi=1;\
min_sl=0x0000000000000000000000000000000000000000000000000
00000000000000000;\
max_sl=0x7fffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffff;\
def_cl=0x7fffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffff;\
forced_privs=empty;
```

EXAMPLE 7 TSIX Host

```
# A sample tnrhtp template entry for tsix hosts
# or networks that label packets with the RIPSO security option.
#
tsix:host_type=tsix;\
min_sl=0x0000000000000000000000000000000000000000000000000
00000000000000000;\
max_sl=0x7fffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffff;\
allowed_privs=all;\
forced_privs=empty;\
def_label=[0x000000000000000000000000000000000000000000000
000000000000000000000];\
def_cl=0x7fffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffff;\
ip_label=none;\
ripso_label=empty;\
ripso_error=empty; \
doi=0;
```

EXAMPLE 8 Routing Unlabeled Packets through a Trusted Domain

```
# A sample tnrhtp template entry for unlabeled hosts
# or networks that are being securely routed through
# a trusted domain with RIPSO labels inserted.
#
ripso_secure_route:host_type=unlabeled;\
def_label=[0x000000000000000000000000000000000000000000000
000000000000000000000];\
def_cl=0x000000000000000000000000000000000000000000000000000
00000000000000;\
forced_privs=empty;\
min_sl=0x0000000000000000000000000000000000000000000000000
00000000000000000;\
max_sl=0x7fffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffff;\
doi=0;\
ip_label=ripso;\
ripso_label=0x3d 0x20000000;\
ripso_error=0x80000000;
```

```
# A sample tnrhtp template entry for unlabeled hosts
# or networks that are being securely routed through
# a trusted domain with CIPSO labels inserted.
#
cipso_secure_route:host_type=unlabeled;\
def_label=[0x00000000000000000000000000000000000000000000000
00000000000000000000];\
def_cl=0x0000000000000000000000000000000000000000000000000000
0000000000000000;\
forced_privs=empty;\
min_sl=0x000000000000000000000000000000000000000000000000000
0000000000000000;\
max_sl=0x7ffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffff;\
ip_label=cipso;\
doi=0;\
ripso_label=empty;\
ripso_error=empty;
```

FILES   /etc/security/tsol/tnrhtp   Trusted network remote-host templates

NOTES   The doi entry is expected for all templates.

The cipso_doi entry is allowed for backward compatibility.

The doi entry is allowed to be empty for backward compatibility. The absence of the doi entry causes the default doi=0 to be used.

Trusted Solaris 8   smnettmpl(1M), tnchkdb(1M), tnd(1M), tnctl(1M), tnidb(4)
4/01 Reference
Manual
SunOS 5.8   attributes(5)
Reference Manual
WARNINGS   Changing a template while the network is up can change the security view of an undetermined number of hosts.

Allowing unlabeled hosts onto a Trusted Solaris network is a security risk. In order to avoid compromising the rest of your network, such hosts must be *trusted* in the sense that the administrator is certain that they will not be used to compromise the environment. These hosts should also be physically protected to restrict access to authorized individuals. If you cannot guarantee that an unlabeled host is physically secure from tampering, it and similar hosts should be isolated on a separate branch of the network.

Unlabeled hosts can be isolated using the Trusted Solaris labeling feature, which ensures that unlabeled packets originating from outside a trusted domain are routed according to their level of trust inside the domain (see Example 8). The gateway to the untrusted hosts must be a sun_tsol host type, and the gateway's database entries for these untrusted hosts and the interface connected to them must be set to reflect the accreditation of these hosts.

NAME | tsolgateways – static routing configuration file

SYNOPSIS | /etc/tsolgateways

DESCRIPTION | The /etc/tsolgateways file is used to configure static routes for a host. At system start up, if /etc/tsolgateways exists, its contents are used to set up static routes. If /etc/tsolgateways does not exist, /etc/defaultrouter is checked. If /etc/defaultrouter exists, its contents are used to set up static routes. If neither /etc/tsolgateways nor /etc/defaultrouter exists, then the host uses dynamic routing. For dynamic routing, if in.rdisc(1M) exists, it is used. If the program file /usr/sbin/in.rdisc does not exist, in.routed(1M) is used.

The tsolgateways file differs from the defaultrouter file in several ways. The latter can be used only to specify default gateways along with simple metrics that indicate the hop count to the destination. tsolgateways can be used not only to specify default gateways but also to specify gateways for specific hosts and networks. Host and network routing entries in tsolgateways can be specified with an optional *emetric* that includes security attributes associated with the route. The emetric is used for trusted routing through the shortest route to a destination through gateways whose security level matches the sensitivity of the data being sent out. The emetric is made up of the simple metric plus additional security routing information (SRI). The SRI includes a sensitivity label range and other optional keywords described below.

The format of /etc/tsolgateways is shown below:

```
default [ gateway [ args ]] [ extended_metric ]
or
[ net | host ] destination [ gateway [ args ]] [ -m emetric ]
or
[ net | host ] destination [ gateway [ args ]] [ metric ]
```

where:

destination	Is the IP address of the network.
gateway	Is the IP address or hostname of the gateway. If a hostname is used, it must be in the /etc/hosts file. Any destination host(s), network(s), and gateway(s) must be specified with an appropriate host type and template in the local or NIS+ versions of the tnrhdb/tnrhtp databases.
metric	Is an integer representing the number of hops to the destination network. This option is supported for backward compatibility.
emetric	Combines the metric and the SRI of a route, as described below.

The first form uses the default keyword to specify a default gateway through which packets are routed if the destination does not match another route specified in the file. If no default is specified and no match can be found among the host or network entries, the packet is dropped.

The third form uses either the `net` or `host` keywords to set up a route to a specific network or host using a simple metric. This form is obsolete.

The second form is like the third form but it uses the `-m` option to specify the *emetric*. The emetric is specified in the following form (with the single line shown as two for readability):

metric= *val*,min_sl=*val*,max_sl=*val*,doi= *val*
ripso_label= *val*,ripso_error=*val*,ripso_only,cipso_only

If *val* contains a space, the space must be protected by double quotes around the value.

The keywords to be used for the emetric are described below:

`metric=`	Specify an integer from 0 to 15 for the number of hops to the destination. Mandatory.
`min_sl, max_sl`	Specify a sensitivity label in either hexadecimal or string form. Mandatory.
`doi=`	Specify a nonzero integer corresponding to a CIPSO domain of interpretation. If this keyword is specified, do not specify `ripso_label`, `ripso_error`, or `ripso_only`.
`ripso_label=`	Specify the classification, followed by a space, followed by a list of protection authority flags (PAF) separated by semicolons (;). The classification and the PAF flags can be specified either in hexadecimal or string form. The supported classifications are TOP SECRET, SECRET, CONFIDENTIAL, and UNCLASSIFIED. The PAF flags (also referred to as the Send PAF) are GENSER, SIOP-ESI, SCI, NSA, and DOE. If this keyword is specified, ripso_error is required. If this keyword is specified, do not specify `doi` or `cipso_only`.
`ripso_error=`	Specify a list of protection flags separated by semicolons (;) in either hexadecimal or string form. The supported PAF flags (also referred to as the Return PAF) are GENSER, SIOP-ESI, SCI, NSA, and DOE. If this keyword is specified, `ripso_label` is required. If this keyword is specified, do not specify `doi` or `cipso_only`.
`ripso_only`	Specify without a value. If a SUN_RIPSO gateway is involved in a route, use this keyword to indicate that a route can only forward packets having RIPSO labels. If this keyword is specified, `ripso_error` and `ripso_label` are required. If this keyword is specified, do not specify `doi` or `cipso_only`.

<table>
<tr><td>cipso_only</td><td>Specify without a value. If a SUN_CIPSO gateway is involved in a route, use this keyword to indicate that a route can only forward packets having CIPSO labels. If this keyword is specified, a doi is required. If this keyword is specified, do not specify ripso_label, ripso_error or ripso_only.</td></tr>
</table>

EXAMPLES   The first two lines in the following example show a default and a network entry, each with a simple metric. The third line shows an entry for a network that specifies the gateway name as chastain-118, and the metric as 2, and that assigns an SRI that specifies a label range from UNCLASSIFIED to CONFIDENTIAL, a ripso label of CONFIDENTIAL GENSER, and a ripso error of GENSER. The fourth line is an entry for a host, with an IP address 192.168.101.3. The host entry specifies a gateway called trusted, with a label range of TOP SECRET to TOP SECRET, a cipso doi of 1, and the optional keyword cipso_only. (The long lines are broken because they do not fit on a single line.)

EXAMPLE 1 Sample tsolgateways file

```
default 192.168.117.1 1
net 192.168.113.0 chastain 1
net 192.168.116.0 chastain-118 -m metric=2,min_sl="UNCLASSIFIED",
max_sl="CONFIDENTIAL",ripso_label="CONFIDENTIAL GENSER",
ripso_error="GENSER"
host 192.168.101.3 trusted -m metric=3,min_sl="TOP SECRET",max_sl="TOP SECRET",
doi=1,cipso_only
```

Trusted Solaris 8 4/01 Reference Manual   in.rdisc(1M), in.routed(1M), route(1M), *Trusted Solaris Administrator's Procedures*

NAME | tsolinfo – Package security-attribute description file

DESCRIPTION | `tsolinfo` describes security attributes used as overrides for file attributes of files contained in a package. This text file is created by the developer of a software package and is included in the package. If the file is not included in the package, a set of default filesystem security attributes will be used.

Each entry in the `tsolinfo` file describes a single file security attribute for a specific file. The entry consists of several fields of information, each field separated by a space. Lines that begin with # are comment lines and are ignored. Empty lines are not allowed. The fields are described below and must appear in the order shown.

attribute　A character field that defines the attribute type. Valid attribute types are:

`label`　A CMW label in text. The exact label name must be used. See EXAMPLES below.

`acl`　A comma-separated list of acl entries terminated with a comma.

`allowed_privs`　A list of comma-separated allowed privileges.

`forced_privs`　A list of comma-separated forced privileges.

`mld`　Specifies a multilevel directory. Do not set an attribute value for this type.

`public`　Specifies that read operations on this file should not be audited. Do not set an attribute value for this type.

pathname　A character file that defines the name of the file for which the attribute is being defined.

attribute-value　A character string that defines the value of the attribute. This field is not valid for the `mld` or `public` attributes.

The `tsolinfo` file also provides a special set of entries to define a set of default security attributes associated with all of the files within a package. The `default` attribute is used to denote a default attribute entry. The pathname component of the entry is replaced with the name of the attribute for which the default is being set. Package defaults can be set for any of the attributes described above. The package defaults override the filesystem default security attributes.

The `tsolinfo` file should be created at the same time as the package prototype file is created, and should be located in the same directory. The `tsolinfo` file must be included in the package prototype file by using the package prototype `include` command.

When the `pkgmk(1)` command is used to create a package, the `tsolinfo` file is relocated to the `install/` subdirectory of the newly created package directory.

EXAMPLES

EXAMPLE 1 A sample tsolinfo file

```
default label [ADMIN_LOW]
default allowed_privs all
default forced_privs all
label usr/sbin/myfile [ADMIN_HIGH]
forced_privs usr/sbin/myfile file_mac_read
allowed_privs usr/sbin/myfile file_mac_read,file_mac_write
```

EXAMPLE 2 A tsolinfo file with an exact CMW label

If an initial compartment is specified for the classification NEED TO KNOW and assigned to default word SSE in the SENSITIVITY LABELS: WORDS: section of the `label_encodings` file, as in:

```
-------------------------------------------------------------------------
CLASSIFICATIONS:

name= NEED TO KNOW;        sname=NTK;  value= 5; initial compartments= 14;
. . .
SENSITIVITY LABELS:
WORDS:

name= SSE;                       compartments= 14;
-------------------------------------------------------------------------
```

it is not enough to enter NEED TO KNOW in the `tsolinfo` file. The label must include all label components, NEED TO KNOW SSE.

```
default label [ADMIN_LOW]
default allowed_privs file_mac_read,file_mac_write
default forced_privs file_mac_read
label usr/sbin/myfile [NEED TO KNOW SSE]
forced_privs usr/sbin/newfile file_mac_read
allowed_privs usr/sbin/newfile file_mac_read,file_mac_write
```

Trusted Solaris 8
4/01 Reference
Manual

SEE ALSO

`setfsattr`(1M)

pkginfo(4), pkgmap(4), pkgmk(1), `prototype`(4)

NOTES

The `tsolinfo` file should only contain entries for pathnames that require special file security attributes, other than the default ones supplied by the UFS filesystem. If the package does not contain any files that require special file security attributes, the `tsolinfo` file should not be created.

If the `tsolinfo` file is not present during package installation, the files contained within a package are assigned default file security attributes provided by the UFS filesystem.

If the `tsolinfo` file contains only the default entries, all of the files within a package are installed with security attributes specified by the `tsolinfo` file entries, along with any non-conflicting default UFS attributes.

NAME	tsolprof – User profiles database
SYNOPSIS	`/etc/security/tsol/tsolprof (obsolete)`
DESCRIPTION	The `tsolprof` database is replaced in Trusted Solaris 8 and later releases with the `exec_attr`(4) and `prof_attr`(4) databases. For library functions that search `exec_attr` entries, see getexecattr(3SECDB). For library functions that search `prof_attr` entries, see the getprofattr(3SECDB) man page.

NAME | tsoluser – User security attributes database

SYNOPSIS | `/etc/security/tsol/tsoluser (obsolete)`

DESCRIPTION | The `tsoluser` database is replaced in Trusted Solaris 8 and later releases with the `user_attr(4)` database. For library functions that search `user_attr` entries, see the `getuserattr(3SECDB)` man page.

NAME	user_attr – extended user attributes database
SYNOPSIS	/etc/user_attr
DESCRIPTION	/etc/user_attr is a local source of extended attributes associated with users and roles. user_attr can be used with other user attribute sources, including the user_attr NIS map and NIS+ table. Programs use the getuserattr(3SECDB) routines to gain access to this information.

The search order for user_attr sources follows the order specified for passwd(4) in the nsswitch.conf(4) file. No entry should be made for user_attr.

Each entry in the user_attr databases consists of a single line with five fields separated by colons (:). Line continuations using the backslash (\) character are permitted. Each entry has the form:

user:qualifier:res1:res2:attr

user	The name of the user as specified in the passwd(4) database.
qualifier	Reserved for future use.
res1	Reserved for future use.
res2	Reserved for future use.
attr	An optional list of semicolon-separated (;) key-value pairs that describe the security attributes to apply to the user. Zero or more keys may be specified. These are the keys currently interpreted bythe system:

auths	Specifies a comma-separated list of authorization names chosen from those names defined in the auth_attr(4) database. Authorization names may be specified using the asterisk (*) character as a wildcard. For example, solaris.printer.* means all of Sun's printer authorizations.
profiles	Contains an ordered, comma-separated list of profile names chosen from prof_attr(4). Profiles are used by the profile shells, pfcsh, pfksh, and pfsh. (See pfexec(1).)
roles	Can be assigned a comma-separated list of role names from the set of user accounts in this database whose type field indicates the account is a role. If the roles key value is not specified, the user is not permitted to assume any role.

type
Can be assigned one of these strings: `normal`, indicating that this account is for a normal user, one who logs in; or `role`, indicating that this account is for a role. Roles can only be assumed by a normal user after the user has logged in.

lock_after_retries
Specifies whether or not an account is locked after the count of failed logins for a user equals or exceeds the allowed number of retries as defined by `RETRIES` in `/etc/default/login`. Possible values are `yes` or `no`.

gen
Contains either of the strings: `automatic` or `manual`. `automatic` specifies that a user must choose a machine-generated password to change a password. `manual` specifies that a user may devise a password of his or her choice.

idletime
Contains a number representing the number of seconds a workstation may remain idle before the window manager attempts the task specified in `idlecmd`. A zero in this field specifies that the `idlecmd` command is never executed.

idlecmd
Contains one of two keywords that the window manager interprets when a workstation is idle for too long. The keyword `lock` specifies that the workstation is to be locked (and thus requires the user to provide a password to resume the session). The keyword `logout` specifies that session is to be terminated (thus killing the user's processes launched in the current session).

labelview
Contains comma-separated keywords. Supported keyword pairs are `internal|external` and `showsl|hidesl`. `internal` specifies that the user may see the `ADMIN_LOW` and `ADMIN_HIGH` labels displayed by various commands and applications, and `external` specifies that the user may not see the labels. `showsl` indicates that labels are displayed, and `hidesl` indicates that sensitivity labels are not displayed.

labeltrans	Contains a hexadecimal number representing the process attribute flags that control label translation.
clearance	Contains the maximum sensitivity label at which the user may operate. This label is given as hexadecimal string. See atohexlabel(1M).
min_label	Contains the minimum sensitivity label at which the user may log in. This label is given as hexadecimal string. See atohexlabel(1M).

EXAMPLES **EXAMPLE 1** Assigning a profile to root

The following example entry assigns to root the `All` profile, which allows root to use all commands in the system, and also assigns two authorizations:

`root:::::auths=solaris.*,solaris.grant;profiles=All;type=role`The `solaris.*` wildcard authorization shown above gives root all the `solaris` authorizations; and the `solaris.grant` authorization gives root the right to grant to others any `solaris` authorizations that root has. The combination of authorizations enables root to grant to others all the `solaris` authorizations. See `auth_attr`(4) for more about authorizations.

FILES

/etc/nsswitch.conf	Configuration file for the name service switch.
/etc/user_attr	Defines extended user attributes.

NOTES When deciding which authorization source to use (see DESCRIPTION), keep in mind that NIS+ provides stronger authentication than NIS.

The root user is usually defined in local databases for a number of reasons, including the fact that root needs to be able to log in and do system maintenance in single-user mode, before the network name service databases are available. An entry should exist for root in the local user_attr file.

Because the list of legal keys is likely to expand, any code that parses this database must be written to ignore unknown key-value pairs without error. When any new keywords are created, the names should be prefixed with a unique string, such as the company's stock symbol, to avoid potential naming conflicts.

In the `attr` field, escape the following symbols with a backslash (\) if you use them in any value: colon (:), semicolon (;), carriage return (\n), equals (=), or backslash (\).

SUMMARY OF TRUSTED SOLARIS CHANGES In addition to `auths`, `profiles`, `roles`, and `types`, the following keywords are used in the Trusted Solaris environment: `lock`, `gen`, `idletime`, `idlecmd`, `labelview`, `labeltrans`, `clearance`, and `min_label`.

lock_after_retries specifies whether or not an account is locked after the count of failed logins for a user equals or exceeds the allowed number of retries as defined by RETRIES in /etc/default/login. Possible values are yes or no.

Trusted Solaris 8 4/01 Reference Manual
auths(1), profiles(1), roles(1), exec_attr(4), nsswitch.conf(4), prof_attr(4)

SunOS 5.8 Reference Manual
pfexec(1), getuserattr(3SECDB), auth_attr(4), passwd(4)

NAME	vfstab – Table of file system defaults
DESCRIPTION	The file /etc/vfstab describes defaults for each file system. The information is stored in a table with the following column headings:

```
device      device      mount      FS      fsck      mount      mount
to mount    to fsck     point      type    pass      at boot    options
```

The fields in the table are space-separated and show the resource name (device to mount), the raw device to fsck (device to fsck), the default mount directory (mount point), the name of the file system type (FS type), the number used by fsck to decide whether to check the file system automatically (fsck pass), whether the file system should be mounted automatically by mountall (mount at boot), and the file system mount options (mount options). (See respective mount file system man page below in SEE ALSO for mount options.) A - is used to indicate no entry in a field. This may be used when a field does not apply to the resource being mounted.

The getvfsent(3C) family of routines is used to read and write to /etc/vfstab.

/etc/vfstab may be used to specify swap areas. An entry so specified, (which can be a file or a device), will automatically be added as a swap area by the /sbin/swapadd script when the system boots. To specify a swap area, the *device-to-mount* field contains the name of the swap file or device, the *FS-type* is "swap", *mount-at-boot* is "no" and all other fields have no entry.

Mount-time security attributes for a file system specified in the vfstab file can be specified with the -o or -S option on the mount(1M) command line or in an entry created for the file system in the vfstab_adjunct(4) file. See the DESCRIPTION sections in the mount and the vfstab_adjunct man pages for more about specifying security attributes. The vfstab file should not be edited directly; instead, it should be edited using the Set Mount Points action, which maintains the proper user, group, sensitivity label, and file permissions for the file and audits all changes. The Set Mount Points action resides in the System_Admin folder available in the Application Manager folder in the Front Panel. By default, the administrator (admin) role has the Set Mount Points action in the File System Management execution profile.

SUMMARY OF TRUSTED SOLARIS CHANGES	Two new pairs of security-relevant mount options devices|nodevices, and priv|nopriv can be specified in the vfstab file for filesystems that support them as filesystem-specific options: mount_hsfs(1M), mount_nfs(1M), and mount_ufs(1M). Mount-time security attributes can be specified for file systems whose objects do not have any attributes (such as user and group IDs) and for file systems that do not have the Trusted Solaris extended security attributes (such as sensitivity labels). Trusted Solaris security policy applies when mounting. The vfstab file should be edited by using the Set Mount Points action.
Trusted Solaris 8 4/01 Reference Manual	mount(1M), mount_hsfs(1M), mount_nfs(1M), mount_tmpfs(1M), mount_ufs(1M), vfstab_adjunct(4)

SunOS 5.8
Reference Manual

fsck(1M), mount_cachefs(1M), swap(1M), getvfsent(3C)

System Administration Guide, Volume 1

NAME | vfstab_adjunct – Attribute data file for mounting a file system

SYNOPSIS | `/etc/security/tsol/vfstab_adjunct`

DESCRIPTION | The `vfstab_adjunct` file can be used to assign any or all of the following mount-time security attributes to the named file system when appropriate: a sensitivity label, forced privilege(s), allowed privilege(s), a filesystem label range, or an MLD prefix. If the mount(1M) command is called with the `-o` or `-S` option to specify security attributes, the `vfstab_adjunct` file is not consulted.

When access control decisions are made, any security attributes on a file or directory always take precedence over security attributes specified either at the filesystem level or mount time.

The `vfstab_adjunct` file is protected at the label `admin_high` and is not edited directly. It should be edited in an `admin_high` workspace by an administrator using the Set Mount Attributes action in the System_Admin folder in the Application Manager. The action maintains the proper user, group, sensitivity label, and file permissions for the file and audits all changes. By default, the security administrator (secadmin) role has the Set Mount Attributes action in its rights profiles.

Mount-time security attributes can be specified for all file systems. When an appropriate attribute is not specified at mount time for a fixed attribute file system, a default value is applied. The default values are described later in this section.

File system types `UFS`, `TMPFS`, and `NFS` (from a Trusted Solaris server) have a full set of Trusted Solaris extended security attributes already defined. (See the getfsattr(1M) man page for how to get attributes on mounted file systems). Because the attributes can be changed on these file systems *after* they are mounted, they are called *variable* file systems. For example, the sensitivity label on a file in a variable file system can be changed by an authorized user. Security attributes on variable file systems can be overridden at mount-time, but objects in the file system that have assigned security attributes retain those attributes.

File systems that do not support the Trusted Solaris extended security attributes are called *fixed* because any attributes assigned to them (either at mount time or by default) cannot be changed. For example, the sensitivity label specified for a mounted fixed-attribute file system cannot be changed on any of the objects in that file system. An object that is moved or copied from the fixed file system to a variable file system can be changed after the move.

Mount-time security attributes override existing security attributes on a file system. However, mount-time attributes never override security attributes on the files and directories within the file system.

Each record in the `vfstab_adjunct` file represents a single file system. An entry consists of the file system's full pathname followed by a semicolon, followed by keyword=value assignments in semicolon-separated fields.

The pathname of the file system is the only portion of the entry that is required and therefore has no keyword associated with it. All keyword fields are optional and follow the format: `keyword=value` where *keyword* is one of the following:

slabel   Sets the sensitivity label for all objects in the file system. Specify the sensitivity label in string (text) or hexadecimal format.

forced   Specify one or more forced privileges for all executable files in the file system. Specify symbolic privilege name(s) in a comma-separated list (such as: `forced=file_audit,` `file_chown;`) or use `all` to indicate all privileges. Using `none` or omitting the keyword results in no forced privileges being applied. For example, the assignment of `forced=;` results in the default of `none` being applied. Any forced privileges must be a subset of the allowed privileges. See `priv_desc`(4) for names of privileges.

allowed   Specify one or more allowed privilege(s) for all executable files in the file system. Specify symbolic privilege names in a comma-separated list (such as: `allowed=file_audit,` `file_chown;`) or use `all` to indicate all privileges. Using `none` or omitting the keyword results in no allowed privileges being applied. See `priv_desc`(4) for names of privileges. Any allowed privilege(s) must be a superset of the forced privileges.

low_range   Specify the lower bound of the file system label range as a sensitivity label in string (text) or hexadecimal format.

hi_range   Specify the upper bound of the file system label range as a sensitivity label in string (text) or hexadecimal format.

mld_prefix   Set a prefix to be used in the adorned names of multilevel directories. (See `multilevel directories` in the `DEFINITIONS` in `Intro`(2) for more about the MLD prefix.) Specify the value in text format (such as: `.MLD.` or `.hidden.`). On unlabeled (fixed attribute) file systems, the prefix generally has no useful effect—with the exception that an `mld_prefix` should be supplied if a variable filesystem is being mounted on the unlabeled filesystem and the root of the variable filesystem is an MLD.

A comment line or entry is terminated by an unescaped newline character. Lines ending with a (\) (backslash) continue the current entry to the next line. Leading and trailing white space characters (blank, tab) surrounding a keyword or an attribute value are ignored. When a keyword value is quoted, spaces can be included within the value. Comments are indicated by a pound sign (#) at the beginning of a line and cause the rest of the line to be ignored.

When a keyword appears without an attribute value or when a keyword is missing, a
default value is assigned to that attribute. The default values for fixed attribute file
systems are:

slabel   The default sensitivity label of a fixed file system being mounted
                 from a local device (such as a hard disk, floppy, or CD-ROM) is the
                 sensitivity label of the device. For an allocated device, the file
                 system is assigned the sensitivity label at which the device was
                 allocated.

forced   None

allowed   None

low_range   ADMIN_LOW

hi_range   ADMIN_HIGH

mld_prefix   None

EXAMPLES   **EXAMPLE 1** PUBLIC Filesystem

The following example sets a sensitivity label of PUBLIC on a file system
(/workspaces) being mounted from an unlabeled host running the Solaris operating
environment. For this to work, PUBLIC must be a valid sensitivity label on the local
host, the file system must either be automounted or an entry must exist for the file
system in the vfstab(4) file. Also, entries for the unlabeled host in the
tnrhdb/tnrhtp files must assign a template to the unlabeled host that specifies a
matching default sensitivity label of PUBLIC.

/workspaces; \
slabel=PUBLIC;

EXAMPLE 2 DOS Filesystem

The following example is for a DOS file system named /no_attributes, being
mounted from a floppy disk. The file system contains an executable that needs the
file_chown privilege in order to work. The entry sets the low_range for the file
system to ADMIN_LOW and lowers the hi_range from the default of ADMIN_HIGH to
ADMIN_LOW.

/no_attributes; \
slabel=admin_low; \
low_range=admin_low; \
hi_range=admin_low;

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsr

Trusted Solaris 8 4/01 Reference Manual

getfattrflag(1), getfsattr(1M), setfsattr(1M), getmldadorn(1), mount(1M), mount_hsfs(1M), mount_nfs(1M), mount_tmpfs(1M), mount_ufs(1M), newsecfs(1M), priv_desc(4)

Trusted Solaris Administrator's Procedures

SunOS 5.8 Reference Manual

attributes(5)

vfstab_adjunct(4)

Index