Least Privilege for Solaris (2002/188)

Casper Dik (Casper.Dik@Sun.COM)

February 20, 2003

Abstract

This document describes the introduction of *Process Privileges* in Solaris. *Process Privileges* serve one main purpose: restricting processes to the privileges required to perform the task at hand and no more.

1 Introduction

The traditional UNIX privilege model associates all privileges with the *effective uid 0*. The basic flaw of that model is the *all-or-nothing* approach. An application which needs a single special privilege, such as a web server binding to the reserved port 80, a program running in the *real-time* scheduling class, a server to keep the clock synchronized, the NFS server, all need to run or start as `root`.

This traditional approach has a number of shortcomings:

- It is not possible to restrict a process to a limited set of privileged operations.
- Each privileged process has complete reign of the system; all vulnerable privileged processes can be leveraged to full access to the system.
- It is not possible to extend an ordinary user's capabilities with a restricted set of privileges.
- It is often unclear exactly what privileged functionality a process requires access to.

Many operating systems, such as VMS, Trusted Solaris, Windows NT, have addressed this by introducing *Process Privileges* of some sort. *Process Privileges* allow the implementation of what is known as *the Principle of Least Privilege*, i.e., running applications with the

least privilege required to perform a certain task. This is generally considered to be a better security model.

This project defines a model and an implementation of *Least Privilege* for Solaris. The various privileged operations inside the kernel are grouped under appropriate *privileges*. The process model is extended with *privilege sets* each containing zero or more privileges. Each process has one *Effective* set which contains the privileges that are currently in effect; a *Permitted* set which contains privileges that can be made effective; a *Inheritable* set which is made effective at **exec(2)** and a *Limit* set which is an upper bound on all future effective sets for a process and its offspring. The project also strives to maintain maximum compatibility with uid 0.

New system and library calls are introduced to examine and change the privilege sets; the existing **user_attr(4)**, **prof_attr(4)** and **exec_attr(4)** and associated utilities are extended to support privileges for RBAC profiles and users. New utilities are provided to inspect and manipulate process privileges, assign privileges required to open devices and define additional privileges.

1.1 Who needs this facility?

The consumers of this facility fall in several categories; Solaris proper can use this facility to restrict the privileges of daemons now running under uid 0. This allows us to lessen the risk that comes with enabling such daemons. Trusted Solaris plans to adopt the Solaris *Process Privileges* implementation instead of its own. Kevlar[1] plans to adopt some features of this project in the implementation of *zones*.

Over the past years new functionality has been introduced in Solaris which was considered too risky to use for ordinary users; **settaskid(2)**, `RT` scheduling class, `CLOCK_HIGHRES` timers. By associating a specific privilege with each of these, local administrators can allow ordinary users access to these facilities without having to resort to set-uid wrappers which are notoriously difficult to get right.

1.2 Who gets this facility?

Process Privileges will be part of core Solaris.

[1] http://kevlar.eng/

1.3 Outline

In section 2 we discuss the privilege models as found in Solaris and Trusted Solaris; the alternatives we considered when moving *Least Privilege* to Solaris and the model we finally decided upon. Section 3 we give the complete formal definition of our model and the additional features we propose. The details of the data structure changes in the kernel, new data structures and interfaces presented to programs and users can be found in section 4. Drafts of new manual pages are included in Appendix A, a selection of drafts of modified manual pages are included in Appendix B in unified diff format. The list of initially defined privileges can be found in the new manual page **privileges(5)**. Tables of exported and extended interfaces can be found in chapter 6.

1.4 Notation used in this document

Throughout this document the following notation is used:

- $\vee$ - logical or
- $\wedge$ - logical and
- $\in$ - element of
- $\cup$ - setwise union
- $\cap$ - setwise intersection
- $\subset$ - is a subset of
- $\subseteq$ - is a subset or equal
- $\emptyset$ - the empty set
- B - the set of basic privileges
- P - the set of all privileges
- $\leftarrow$ - assignment

2 Process Privileges Models

Converting Solaris from one privilege model to the next is a major undertaking. From the outset we formulated the following requirements:

- The new model should be completely transparent to legacy applications.
- Applications expecting the old and new model should be able to coexist in the same Solaris instance.
- It should be possible to restrict a process and its offspring to a certain set of privileges.

We want complete backward compatibility *and* we want to be able to use the new functionality in the system *at all times*. If it is functionality that can be turned off, Solaris itself cannot use it and we cannot apply the principle of least privilege to Solaris system daemons. Acceptance by ISVs will necessarily suffer also. The functionality would not only be restricted to the latest release, but even then it would not be guaranteed to be configured on. Conversely, if traditional Solaris applications cannot work in the new environment, we would be required to provide a knob to turn the feature off with all the problems that brings.

The third requirement follows naturally from the traditional UNIX capability of *set-uid* applications, applications which gain elevated privileges merely by being executed. If we are able to limit the extent of privileges a process can gain even in the face of *set-uid* applications, we further our goal of improving overall system security.

E.g., one can imagine a system where the privileges needed to configure network interfaces can only be obtained when logging in on the console. If we extend privileges to cover actions which are not privileged at this time, we could disallow all outgoing network activity for sessions originating from the network, making it impossible to use a system as a stepping stone.

Compatibility, Coexistence and Containment are our three guiding principles.

We will continue with a high level description of the current Solaris Super-user model, the Trusted Solaris privilege model and our proposed Solaris privilege model followed by an in-depth discussion of the Solaris privilege implementation.

2.1 The Traditional Solaris Super-user Model

The traditional Solaris Security model has only seen minor revisions since the first UNIX days.

Each process has a credential associated with it; the credential consists of an *effective*, *saved*, and *real* user and group ids as well as a supplementary group list. Except for a few

exceptions[2], a process is considered privileged if and only if the effective uid is 0. This test is generally performed in two kernel routines, the DDI-compliant interface **drv_priv(9f)** and the traditional UNIX kernel function `suser()`, although some code checks the credential against uid 0 directly.

The three uids are usually exactly the same. When they differ, they allow a coarse-grained swapping between different privilege states.

In the context of our discussion, we define C, the process credential, C' the credential resulting from an operation and X, the properties of an executable. The items of interest for our discussion are:

- $C.e$ - the *effective* uid of a process
- $C.r$ - the *real* uid of a process
- $C.s$ - the *saved* uid of a process
- $X.u$ - the *uid* of the user owning an executable[3]

Both `suser()` and `drv_priv()` are essentially defined as $C.e = 0$.

There are several times at which uids can change.

- When a non set-uid program is executed the saved uid is made equal to the effective uid. Typically, this doesn't cause a change to the process credential.

$$C'.s \leftarrow C.e$$

- When a set-uid program is executed the saved and effective uid are set to the uid of the program.

$$C'.s \leftarrow C'.e \leftarrow X.u$$

- When a process manipulates uids using any of the **setuid(2)** family of system calls.
  - setuid(u) - change $C.e, r, s$ if we're Super-user, else change $C.e$.

$$\begin{aligned} suser(C) \Rightarrow \quad & C'.s \leftarrow u, C'.e \leftarrow u, C'.r \leftarrow u \\ C.r = u \vee C.s = u \Rightarrow \quad & C'.e \leftarrow u \end{aligned}$$

[2]The IA scheduling class and power management allow certain functions to be performed by processes with an effective *group* id of 0

[3]We use u and not o in keeping with **chmod(1)**

– seteuid(u) - change $C.e$.

$$C.r = u \vee C.s = u \vee suser(C) \Rightarrow C'.e \leftarrow u$$

– setreuid(r, e) - change $C.e$, $C.r$, update $C.s$ if $C.r$ changes and the new $C.e$ isn't equal to the new $C.r$.

$$\begin{aligned} e \neq -1 \wedge (C.s = e \vee C.r = e \vee suser(C)) &\Rightarrow C'.e \leftarrow e \\ r \neq -1 \wedge (C.e = r \vee suser(C)) &\Rightarrow C'.r \leftarrow r \\ r \neq -1 \wedge C'.e \neq C'.r &\Rightarrow C'.s \leftarrow C'.e \end{aligned}$$

In the OS-Networking kernel components of Solaris, the Super-user policy is enforced using the following checks, in decreasing frequency of use:

- Direct comparison of `cr->cr_uid` (the effective uid).
- The tradition kernel function call `suser(cred_t *)`.
- The DDI-compliant function call `drv_priv(cred_t *)`.
- Direct comparison of `cr->cr_ruid` (the real uid, used in some resource limit situations)

Additionally, files and device nodes on the system are protected by file ownership and permissions bits also known as *discretionary access control (DAC)*.

2.2 The Trusted Solaris Privilege Model

Trusted Solaris was Sun's first operating system to introduce fine grained privileges. The current Trusted Solaris implementation adds four privilege sets as well as an additional user id to the process credential:

- $C.I$ – the Inheritable set
- $C.P$ – the Permitted set
- $C.E$ – the Effective set
- $C.S$ – the Saved set
- $C.x$ – the start user id which is associated with the saved set.

The traditional components of C behave as in Solaris.

The *Effective Set* contains the privileges that are currently in effect; the *Permitted Set* contains privileges that the process can put in the effective set at will. The *Inheritable Set* contains those privileges which are inherited across **exec(2)**. The *Saved Set* and *start user id* are used to implement a form of compatibility for traditional set-uid applications.

While the notion of *set-uid* still exists, a similar mechanism was introduced for privileges. The filesystem is extended to allow two privilege sets to be associated with each executable:

- $X.A$ – the Allowed set
- $X.F$ – the Forced set

The *Forced Set* is most like the *set-uid* feature; when a forced set is added to an executable, the privileges in the Forced Set come into effect when it is executed. The *Allowed Set* puts a bound on the privileges the program can wield and allows you to contain an untrusted executable in some fashion.

The *Saved Set* can provide some compatibility for old set-uid root applications by adding forced privileges and making the application set-uid.

A process can manipulate the privilege sets E, P and I in a restricted fashion using the **setppriv(2)** system call, such that the following conditions are always met:

$$C'.I \subseteq C.I \cup C.P$$
$$C'.P \subseteq C.P$$
$$C'.E \subseteq C'.P$$
$$C'.S \subseteq C'.P$$

A process can add privileges from P to I and E; a process can remove privileges from I, P and E, noting that when privileges are removed from P they are automatically removed from E and S as well but they are not removed from I. A process cannot modify S directly.

When a program X is executed, the following transformations take place:

$$C'.I \leftarrow C.I$$
$$C'.P \leftarrow (C.I \cup X.F) \cap X.A$$
$$C'.E \leftarrow C'.P$$
$$C'.S \leftarrow C.I \cap X.A$$
$$C'.x \leftarrow C'.e$$

The uids follow the same rules as in Solaris.

The *Saved Set* and *start-uid* are used in the following manner when the effective user id changes:

$$C.e = C.x \wedge C'.e \neq C.x \;\Rightarrow\; C'.S \leftarrow C.E, C'.E \leftarrow \emptyset$$
$$C.e \neq C.x \wedge C'.e = C.x \;\Rightarrow\; C'.E \leftarrow C.S$$

In the Trusted Solaris kernel, the kernel policy is enforced by checking whether the required privilege p is a member of the effective set, $C.E$:

$$p \in C.E$$

In addition to privileges, the Trusted Solaris kernel supports a second mechanism, known as *Mandatory Access Control*. While beyond the scope of this project, it is important to mention *MAC* briefly here as it is an additional mechanism in Trusted Solaris that prevents processes with an effective uid 0 from tampering with certain files and device nodes owned by user id 0.

The effective user id of a Trusted Solaris processes is only used for discretionary access control; user id 0 no longer has any special meaning. The saved set allows us to have some form of automatic privilege set juggling on uid swaps, but not one that would allow full compatibility for ordinary uid 0 processes as it only caters for some form of set-uid applications.

The implementation of privileges moves all the responsibility for privilege checks to a single module; the `suser()` calls are all replaced with `secpolicy_policy_name()` to make the appropriate privilege decision; one such function exists for each type of decision, abstracting the actual privileges needed from the interface used inside the kernel.

2.3 Extending Solaris with Process Privileges

One of the major selling points of Solaris is continued backward binary compatibility, allowing for applications to live on in binary form without change, hopefully forever.

We believe that moving from the *suser* model to an exclusively fine grained privilege based model would break quite a few promises we have made our customers. Applications assume they have powers with uid 0 and set-uid root binaries would suddenly break if we changed that.

We would like to evolve to a model where ordinary user ids can have additional privileges and where the user with uid 0 is no longer all powerful. This has led us to make the choice that we check against the privilege sets only and not against uid 0.

We started out with a simplified Trusted Solaris model; we drop S and x because we are going to guarantee full compatibility rather than the partial compatibility offered by S and

x. Current lack of security attributes for files in Solaris makes it impossible to implement *Forced* and *Allowed* sets so we need to define different mechanisms to gain privilege and deny privilege. We feel that this project has enough to offer without supporting security attributes. When such attributes appear in future, they can easily be incorporated into our framework.

We add a new set, the *Limit Set* (L), the upper bound of privileges that a process and its offspring can inherit. The limit set is only enforced at exec time, allowing a process to drop privileges on exec, while still using them until that point in time. L can be seen as carrying A with a process; all executables will appear to have L as allowed set for that particular process.

This gets us our initial list of per-process privilege sets:

- $C.I$ – the Inheritable set
- $C.P$ – the Permitted set
- $C.E$ – the Effective set
- $C.L$ – the Limit set

The **exec(2)** privilege set transformation rules are:

$$\begin{aligned} C'.I &\leftarrow C.I \cap C.L \\ C'.P &\leftarrow (C'.I \cup X.F) \cap X.A \\ C'.E &\leftarrow C'.P \\ C'.L &\leftarrow C.L \end{aligned}$$

But since we are not able to add *Forced* and *Allowed* privileges in the initial implementation, this simplifies under $F = \emptyset$ and $A = P$ (where P is the set of all privileges) to:

$$\begin{aligned} C'.I &\leftarrow C.I \cap C.L \\ C'.P &\leftarrow C'.I \\ C'.E &\leftarrow C'.P \\ C'.L &\leftarrow C.L \end{aligned}$$

As in Trusted Solaris, all kernel security policy checks are performed using privileges only:

$$p \in C.E$$

As we have chosen to implement security policy enforcement solely based on the effective privileges of a process, *Super-user* compatibility would appear to be a daunting task. Most of the discussions in the design stage have centered on the various proposed methods of achieving this compatibility without sacrificing security.

Without going into too much detail about methods we are not proposing, we feel that it is useful to mention some of the aborted attempts at superuser compatibility models, why we abandoned them and why we selected a particular model.

We used several criteria to judge the proposed models:

- Simplicity - complexity is the enemy of security and usability; we want fewer privilege sets, and few operations with unexpected or unintended side effects.
- Compatibility - we want full compatibility for existing software, including programs customers are known to replace such as daemons like **telnetd(1m)**, **sshd(1m)**, and programs such as **login(1)** and **sendmail(1m)**. Installing an older version of Solaris user code on a privilege aware kernel should work as it did before; during development this allows us to change only ON components while keeping the rest of the WOS unchanged.
- Least surprise - compatibility features should not get in the way of programs using Least Privilege. It should be possible to completely separate user id and privilege manipulation.
- Permitted set bounding - the permitted set should not gain privileges except when set-uid root.
- Forward compatibility

2.3.1 Trusted Solaris

Trusted Solaris was not developed with compatibility as an absolute goal; we have already established that it assigns no special meaning to uid 0, and it is therefore incompatible with the Solaris Super-user model.

2.3.2 Root Set

The *Root Set* proposal added a system-wide set *R*. This set would be added into *P* and *E* when executing a set-uid program. If a program performed the system call `setuid(0)`, *R* could be added to *I*; once a process made all of its uids non-zero, *R* was subtracted from all privilege sets except *L*, or, in another variant, the other privilege sets were recovered from *I*.

R suffers from several drawbacks; precarious privilege set manipulations were needed in order to retain privileges while getting rid of uid 0; adding R was easy enough, but removing R would be destructive. A single privilege awarded to a process could not survive any uid 0 transitions.

Privilege manipulations on transition from or to an effective uid of 0 were needed but made it impossible to run a process in a mode where it only cared about privileges and not about uids.

Full compatibility wasn't achieved and while it might have gotten there that would certainly have pushed the model's complexity to an unacceptable level.

The *Root Set* model also included the future possibility of removing privileges from R to end up with an empty R in the far future.

Variants that used R as E when $euid = 0$ were also discussed; such models disallow running a uid 0 process without privileges except with the help of L; that was considered too limiting for certain applications.

2.3.3 As many sets as it takes

We then tried to extend the model with as many sets as necessary to achieve the desired result.

- E_R - the effective set if $C.e = 0$
- P_R - the permitted set if $C.e = 0 \vee C.s = 0 \vee C.r = 0$. (Any of the process' uids is 0)

With $E_R = P_R = P$, this does give full compatibility but E_R and P_R are very visible to applications. When an application needs to have its privileges independent from uid manipulations, it must make sure that $E_R = E$ and $P_R = P$. More complicated is the behaviour on exec, especially in the light of set-uid root applications; should E_R and P_R be inherited or are they reset to either L or a system wide R on exec?

The main problem with this model is the fact that the application needs to be aware that uid manipulations affect E and P and that the application needs to manipulate both in order to achieve uid independence. Another is the counter intuitive behaviour that a process can have fewer privileges when the effective uid is 0 than when it is non-zero.

The different privilege sets might also need to be manipulated directly, which would require changes to applications if we where to adopt a privilege only model in future. Or there would need to be a *mode bit* to indicate that the sets for uid 0 and non-0 are now connected.

2.3.4 Privilege Awareness

The final compatibility model is the *Privilege Awareness* model; the privilege state of the program is extended with a *Privilege Aware State* (*pas*) which can take the values PA (privilege aware) and NPA (not privilege aware). We also introduce the notion of *observed* Effective and Permitted sets, E^O and P^O, respectively. These are the E and P as observed by the code that enforces the kernel security policy, the process itself and other processes.

A process running in NPA mode, behaves almost exactly like a traditional process. It may have privileges in E and P and the kernel will honor those, but if the process set its effective uid to 0 the upper bound of that process's privileges, L, is used as the effective set instead. Similarly, if any of the process' uids become 0, L is used as the observed P. The observer will see E follow the transition of the effective uid and he will see P revert when all of the process' uids are no longer 0. In contrast, the *implementation* sets P^I and E^I remain unchanged during uid manipulations.

An observer will not see E and P change when a process in PA mode changes its uids around.

Summarizing, the observer sees E and P behave as follows:

$$\begin{aligned} C.E^O &= C.e \neq 0 \vee C.pas = PA \; ? \; C.E^I : C.L \\ C.P^O &= (C.e \neq 0 \wedge C.s \neq 0 \wedge C.r \neq 0) \vee C.pas = PA \; ? \; C.P^I : C.L \end{aligned}$$

Changing *pas* is mostly automatic and restricted; a process that manipulates E or P becomes privilege aware automatically as E^O and P^O must be decoupled from L; similarly, if L is changed, E^O and P^O must remain unchanged which can only be achieved by a change to PA. A change to I only has effect after the subsequent **exec(2)** and does not affect *pas* until that time. A process can only become NPA if E, P can be adjusted in a manner that keeps E^O and P^O unchanged and when such a change can not lead to the process leveraging *uid 0* into full privileges.

While we admit that a state bit such as *pas* doesn't deserve any credit towards the *software engineering of the month award*, it has some benefits over the other proposals:

- It is immediately obvious that a process is fully backward compatible with the Solaris Super-user model when it is NPA; i.e., unless an application chooses to be incompatible it will not notice the existence of privileges.
- It is immediately obvious that when a process is PA, no manipulation of uids will ever affect the privilege sets.
- There is no need to keep and administer alternate copies of E or other sets with all the possible resynchronization trouble that brings.

Some or most of shortcoming of the other models could also be addressed with a *pure privilege* state bit; we believe, however, that this model offers best compatibility, the ability to switch to a pure privilege model on a per process basis, it is easily extended with the ability to force all processes to be privilege aware which allows for, e.g, a Trusted Solaris implementation and it a relatively easy to understand.

2.4 Other Unix Implementations

Solaris does not exist in a void; we pause here to see what other UNIX vendors have done. A long time ago, a POSIX standard was in development, 1003.1e. The standard was abandoned for various reasons; some blame lack of interest by the industry at the time, others blame the size; POSIX 1003.1e tried to standardize *Mandatory Access Controls*, *Access Control Lists*, *Information Labels*, *Security Auditing* and *Capabilities*. It was abandoned in January, 1998.

Even though the defunct draft standard uses the term *privileges* throughout the document, the term *capabilities* was substituted for most uses of privileges in the standard shortly before the standard floundered. The rationale given for this was that "It has been pointed out that the term *privilege* has been commonly used for a mechanism that achieves the above stated goals. However, the term *privilege* is also commonly used in the international community to mean something else entirely. It is felt that the confusion that would result from using the term *privilege* would not serve this standard well." [4]

We don't understand what this confusion is about; moreover, the term *capability* is already well established in computer science and has a completely different meaning. The choice of terminology has made the documentation more confusing as capabilities and privileges are used almost interchangeably but not quite.

Various different drafts are being used in various forms in both IRIX and Linux. The IRIX implementation appears to allow hard-coding fixed size bit sets in executables. Both implementations make numerical capability values visible as manifest constants.

Differences exist in the precise rules for process inheritance. Linux appears to have started with draft 17, IRIX capabilities have their roots in draft 16.

Each process and each executable file has capabilities associated with it; each capability has three flags associated with it, named *Inheritable*, *Permitted* and *Effective*. We can think of this in terms of three sets of capabilities associated with each executable and process.

The **exec(2)** rules are defined as follows in Linux, with the restriction that $X.E$ is either $\emptyset$ or P:

[4] POSIX 1003.1e, draft 17, p. 307

$$
\begin{aligned}
C'.I &\leftarrow C.I \\
C'.P &\leftarrow X.P \cup (X.I \cap C.I) \\
C'.E &\leftarrow C'.P \cap X.E
\end{aligned}
$$

and as follows in SGI IRIX:

$$
\begin{aligned}
C'.I &\leftarrow C.I \cap X.I \\
C'.P &\leftarrow X.P \cup (X.I \cap C.P) \\
C'.E &\leftarrow C'.P \cap X.E
\end{aligned}
$$

and different again in POSIX 1003.1e, draft 17:

$$
\begin{aligned}
C'.I &\leftarrow C.I \\
C'.P &\leftarrow (X.P \cap XX) \cup (X.I \cap C.I) \\
C'.E &\leftarrow C'.P \cap X.E
\end{aligned}
$$

where XX is an additional implementation specific restriction.

The last definition is very much like our privilege sets; $C.I$ remains constant, $X.I$ is virtual identical to the *Allowed* set and $X.P$ behaves like our *Forced* set. $X.E$ should contain all privileges for applications not aware of privileges and none for those applications that know how to turn privileges on. XX fits our *Limit* set.

IRIX 6.5 can be configured to either use capabilities exclusively or also allow uid 0 to be honored. IRIX has support for capabilities in the filesystem, as long as you use XFS.

Linux 2.4.16 has various settings, the default assigns all capabilities to uid 0 and removes them on **set*uid(2)** calls that do away with the effective uid. The system supports a per-process flags that allows you to keep privileges when getting rid of uid 0; Linux supports a global flag that causes uid juggling not to effect privileges at all.

But was we found in our own design process, such solutions are problematic. It was therefore no surprise that the direction experimental Linux patches take is making sets and uids orthogonal, introducing a per-process flag that determines whether uid 0 is special or not. Unprivileged processes, however, cannot affect this state and it must be changed explicitly. A further *set-uid 0 not allowed* flag exists. Both flags are inherited.

Additionally, patches exist which add *Basic Privileges* to Linux; in the first Linux model these need to be special cased on all points where uid 0 is causes capability sets to be changed.

Analogous to our proposed per-process *Limit Set*, Linux has a system-wide *Capability Bound*; later patches have made the bound per-process, though strangely not part of the

capability state proper. It is unclear to us why the flags and per-process capability bound are not part of the capability state.

Our main reason not to go with a POSIX 1003.1e draft based implementation is the fact that it is not an approved standard, it is even withdrawn; what exists in implementation space is very much a moving target. Some of the features now found in Linux and in our proposal did not exist when we started our project. The standard took more than 10 years to develop and then failed. The terminology chosen is often confusing; all known implementations use some form of *sets* but the standard tries hard not to speak of three capability sets. Using the same three sets for executables and processes doesn't do wonders for clarity either. The Linux capability FAQ[5] explains this as follows: "Now to make sense of the equations think of $X.P$ as the Forced set of the executable, and $X.I$ as the Allowed set of the executable."

The current Linux implementation is very much in flux, it has gone through several iterations trying to achieve sufficient uid 0 compatibility.

By choosing a completely separate name space, PRIV_ over CAP_, we keep the option open of implementing 1003.1e compatible interfaces on top of ours in future, once those are sufficiently established.

Both the IRIX and Linux implementation address few of our forward binary compatibility constraints by externalizing numerical constants and binary representation in filesystems. IRIX appears to make the size of a capability set a first-class citizen for much of the code whereas Linux appears to limit the visibility of the data structure to the kernel. As binary device drivers are a necessity for Solaris, we cannot use numeric constants or bit test macros outside of the OS-Networking consolidation.

Linux device access is arranged on an ad-hoc basis inside the kernel. Support for capabilities in the filesystem is available as an experimental patch only, at the time of this writing.

[5] http://www.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/capfaq-0.2.txt

3 Privilege Awareness: the Details

In this section we present the precise semantics and describe the various features of our privilege model.

3.1 Per Process State

We extend the per-process state from 2.3 with the *Privilege Awareness* property, *pas*, and the semantically void[6] privilege debugging flag, *db*, and get:

- $C.E$ – the Effective set
- $C.P$ – the Permitted set
- $C.I$ – the Inheritable set
- $C.L$ – the Limit set
- $C.pas$ – the privilege awareness property
- $C.db$ – the privilege debugging flag

The notion of *observed* Effective and Permitted set is used only in those cases where we need to make a distinction between the *implementation* sets E^I and P^I and the *observed* sets E^O and P^O. When talking about the semantics and the user visible behaviour, we will use E and P for E^O and P^O. These are also the values returned by **getppriv(2)** and as shown by **ppriv(1)**. For reference, these are the observation rules:

$$\begin{aligned} C.E^O &= C.e \neq 0 \vee C.pas = PA \; ? \; C.E^I : C.L \\ C.P^O &= (C.e \neq 0 \wedge C.s \neq 0 \wedge C.r \neq 0) \vee C.pas = PA \; ? \; C.P^I : C.L \end{aligned}$$

The initial process `init` starts off with $E^I = P^I = I = I^0$ and $L = P$. I^0 is the set of basic privileges which can be empty. This default process credential behaves exactly like the *Superuser* model: when the effective uid is non-zero, the process has the default privileges; when the effective uid is 0, the process has all privileges.

[6] In Trusted Solaris, privilege debugging is a system wide property that requires a reboot to switch on or off; it then allows applications in the Trusted Path to manipulate the privilege debugging flag for certain applications; those applications will then have all privilege checks succeed; this privilege debugging property is not semantically void.

3.2 Privilege Awareness state transitions

In this section we describe the transitions between being privilege aware and not being privilege aware. This can only happen if the *observed set invariance* condition is met. I.e., when a process transitions between PA and NPA, no changes to E and P are observed.

3.2.1 Becoming Privilege Aware

There are no restrictions on becoming privilege aware. Assuming $pas = NPA$, the following changes to the credential take place when transitioning to PA:

$$\begin{aligned} C'.pas &\leftarrow PA \\ C'.E^I &\leftarrow C.e \neq 0 \; ? \; C.E^I : C.L \\ C'.P^I &\leftarrow C.e \neq 0 \wedge C.s \neq 0 \wedge C.r \neq 0 \; ? \; C.P^I : C.L \end{aligned}$$

E^O and P^O are constant under this operation. This mechanism converts a uid 0 process that appears to follow the Super-user model to a fully privileged process for which uid manipulations no longer have an effect on the privilege status.

3.2.2 Losing Privilege Awareness

There are several times when a process may want to lose the privilege awareness, e.g., on **exec(2)**. Several restrictions apply as not all combinations of privilege sets can successfully and safely be converted to an equivalent NPA process. We derived the following rules from the observed set invariance property:

- If any of the uids is 0 P^I must be equal to L;
- If the effective uid is 0 E^I must be equal to L;
- If none of the uids is 0 there are no restrictions.

The last rule tells us that in a process without any 0 uids, *pas* can be changed at will.

When the conditions are met and a process transitions to NPA, $C.pas$ is set to NPA and the implementation sets are conditionally changed as follows:

$$\begin{aligned} C.e = 0 \vee C.r = 0 \vee C.s = 0 &\Rightarrow C'.P^I \leftarrow C.I \cap C.L \\ C.e = 0 &\Rightarrow C'.E^I \leftarrow C.I \cap C.L \end{aligned}$$

Again note that E^O and P^O are unchanged under these operations.

3.3 Privilege State Manipulation

There are various times at which the per-process state changes; either as a result of explicit actions or as a side-effect of other actions. In some cases the changes are merely observed and not reflected in the underlying data structures.

3.3.1 What happens when user ids change?

In all circumstances, the actual implementation sets remain unchanged, but we can distinguish several cases:

- In privilege aware processes, no changes are observed.
- In processes not having any uids set to 0, no changes are observed, even if privileges are used to obtain uid 0.
- In non-privilege aware processes, E^O will alternate between L and E^I depending on the value of the effective uid; P^O will appear as L until the last uid is set to non-zero; it will then revert to P^I.

3.3.2 What happens on process creation?

When creating a process using any of **fork(2)**, **vfork(2)** or **fork1(2)**, the full privilege state is inherited by the child process.

3.3.3 What happens when execing a different program?

When calling any of members of the **exec(2)** family, the following rules apply:

$$\begin{aligned} C'.I &\leftarrow C.I \cap C.L \\ C'.P &\leftarrow (C'.I \cup X.F) \cap X.A \\ C'.E &\leftarrow C'.P \\ C'.L &\leftarrow C.L \end{aligned}$$

We include $X.A$ and $X.F$ in these equations both for use in future implementations and to keep open the option of per-mount point allowed sets and some form of forced privilege emulation[7].

[7] At one stage the prototype emulated a single forced privilege using the set-gid bit; a single privilege appears to be sufficient for many solaris applications: commands using **rcmd(3socket)**, such as **rsh(1)** and **rlogin(1)** as well as applications using raw sockets, e.g., **ping(1m)** and **traceroute(1m)**.

The *db* property is inherited across **exec(2)**, *pas* is adjusted using the following rules:

- If a process is NPA, it becomes PA only in the presence of forced privileges, taking the following rule into consideration.
- If a process is PA, the privileges are examined before exec and a process becomes NPA if possible using the rules outlined in 3.2.2. If the processes remains PA a second attempt at reverting to NPA is attempted directly after the exec.

The changes to the *implementation* sets E^I and P^I are defined in such a way that the *observed* sets E^O and P^O remain the same.

As a consequence of these rules, it is not always possible for PA processes to gain privileges by executing a *set-uid root* executable. It *is* possible to gain privileges in that way for NPA processes.

In addition to the current practice of marking a process `NOCD|SUGID`[8] if the real and effective ids do not match, we add the marker if the permitted set of the process calling exec is not a superset of the inheritable set. I.e., when the child process runs with more privileges it is awarded additional protection.

An implementation of Trusted Solaris might force most or all processes to PA.

3.3.4 Manipulating Privileges directly

This project adds a system call family **setppriv(2)** which allows a process to manipulate its privileges directly.

The function `setppriv()` takes three arguments: the operation to perform, which is one of `PRIV_OFF`, `PRIV_ON` or `PRIV_SET`; the privilege set name and, as third argument, a privilege set with privileges to switch off, on or to replace the current set completely with.

The *effective set* is most useful to perform *privilege bracketing*. In the Super-user world, privilege bracketing looks like this:

```
/* program start */
uid = getuid();
seteuid(uid);
```

[8]The NOCD flags was introduced as a consequence of the realization that the current credentials of a process do not properly reflect the capability status of a process; if a process has opened restricted file, device or has acquired some other capability and afterwards dropped its extra privileges, the process still has some capabilities or data that should be restricted. The name has its origin in *NO Core Dump* but was later extended to also deny process inspection through `/proc` and in libraries to restrict the use of certain environment variables.. The `SUGID` flag is set when a process was the result the exec of a set-uid or set-gid executable.

```
/* privilege bracketing */
seteuid(0);
/* code requiring super-user privileges */
....
seteuid(uid);

/* ordinary code */

....

/* Permanently giving up root */

setreuid(uid, uid);
```

With *Process Privileges*, the code (simplified) becomes:

```
/* program start */
getppriv(PRIV_PERMITTED, pset);
setppriv(PRIV_SET, PRIV_EFFECTIVE, emptyset);

/* privilege bracketing */
setppriv(PRIV_SET, PRIV_EFFECTIVE, pset);
/* code requiring privileges */
....
setppriv(PRIV_SET, PRIV_EFFECTIVE, emptyset);

/* ordinary code */

....

/* Permanently giving up privileges */

setppriv(PRIV_SET, PRIV_PERMITTED, emptyset);
```

The added advantage is that this manipulation can be done at the level of single privileges; for this purpose convenience a single convenience function, **priv_set(3c)**, is provided.

Removing privileges from *E* is not restricted; only privileges in *P* can be added to *E*. A process manipulating *E* needs to be PA; it will transition to PA following the rules in section 3.2 if necessary.

The *permitted set* can only be shrunk; a process should shrink its permitted set if it does not have any future need for the privileges. Privileges removed from *P* are automatically

removed from E, enforcing $E \subseteq P$. Processes manipulating P will transition to PA automatically.

The *inheritable set* can be added to and removed from; privileges from P can be added freely to I. Privileges can be removed from I without restriction. As changing I does not influence E^O or P^O, a process does not need to transition to PA when manipulating I but might transition to PA on subsequent **exec(2)**. I is allowed to be a superset of P, i.e., removing privileges from P does not affect I.

The *limit set* can not be added to by any mechanism. Privileges can be removed from the limit set but not without peril. Applications running under uid 0 usually assume full privileges on the OS; this often leads to no error checking for privileged operations or, worse, different behaviour of certain operations whether you are privileged or not. E.g., `setuid(non-0)` will reset *all* uids to non-0 if the effective uid is 0 but will only change the *effective* uid if a process is not privileged but can swap uids because of the value of the *saved* uid.

It was discovered[9] that removing selected privileges from a further privileged process could render it insecure. To this end, we define the set of unsafe privileges, the privileges to risky to allow privilege elevation without. A process that doesn't have all privileges from the unsafe set, currently `PRIV_PROC_SETID`, `PRIV_SYS_RESOURCE` and `PRIV_PROC_AUDIT`, in L will fail to execute set-uid 0 processes.

Changing the limit set would influence P^O and E^O, so a process needs to transition to PA when it changes L.

A process that manipulates its privileges is marked `NOCD` which awards the process additional protection.

3.3.5 Manipulating *pas* directly

Using **getpflags(2)** and **setpflags(2)** processes can query and change *pas*. The latter function enforces the rules outlined in section 3.2.

3.3.6 Manipulating Privileges through proc(4)

The **proc(4)** interface is extended to allow inspecting and changing privileges through the `/proc` filesystem. The modifications through this interface are severely restricted:

- A process can only attach to a target process when all the uids and gids of the target process are equal to the its effective uid and gid, respectively, and when the process does not have the `NOCD` flag set or when the process has the `PRIV_PROC_OWNER`

[9] Linux capabilities introduce a similar feature; unrestricted use triggered a problem in **sendmail(1m)** mailer: http://www.sendmail.org/sendmail.8.10.1.LINUX-SECURITY.txt

privilege. The attaching process must also have all the privileges available in the target process in its E and its limit set must be a superset of the target's L. I.e., $C_T.P \subseteq C_A.E \wedge C_T.I \subseteq C_A.E \wedge C_T.L \subseteq C_A.L$. This prevents the attaching process from gaining privileges directly or after **exec(2)**.

- $C_T.L$ cannot be grown, only shrunk (following the rules in section 3.3.4).
- Only privileges in E of the attaching process can be added to the other sets of the target process.
- A process that has its privileges manipulated directly becomes PA unless it can swap to NPA mode following the rules in section 3.2.

In addition, if a process can modify the state of a process, it can always switch *db* on or off.

The above rules help enforce the fact that the special privilege `PRIV_PROC_OWNER` is *not* sufficient to gain control over a process when such control can lead to *escalation of privileges*. In the special case where any of the user ids in the target process is zero the full set of privileges is required. The privilege does give unrestricted read access.

3.4 Privilege Escalation Prevention

We define privilege escalation as *The process whereby a subject can obtain one or more privileges by performing an action that does not require those privileges if that action is not specifically allowed by the security policy*.

One of the oft recognized problems with privilege implementation is that some privileges provide back doors to others. We can think of many cases where this can happen:

- A privilege to modify all processes privileges would allow a process to obtain all privileges by modifying its own privileges.
- A privilege sufficient to attach to all processes is equivalent to obtaining the superset of all privileges currently in permitted sets.
- A privilege sufficient to modify the system's administration files could allow a process to assign all privileges to a user.
- A privilege to write directly to `/dev/kmem` allows a process to change its own credentials so this is equivalent to having all privileges.

- A privilege to write directly to `/dev/dsk/*` allows a process full access over all file contents, this is equivalent to having all the `file_*` privileges or even all privileges if the disks in question contain system configuration files or allow set-uid execution.

While L could make certain that such privileges don't befall ordinary users, such restrictions on L may make perfectly valid, controlled, uses of such privileges by privileged programs impossible and we feel that we should put extra hurdles whenever we find a place where certain privileges can be leveraged to more privileges. Where we can identify such cases of *Privilege Escalation*, we will require a process to either possess as many privileges as it would be able to obtain with the specific action or, in the specific instance of a process P_S granting privileges to a process P_O, we will require that those privileges do not exceed $P_S.E \cap P_O.L$.

3.5 The Trouble with uid 0

The Super-user's user-id is not only used for privilege checks but also for discretionary access control on files and devices as the Super-user owns most files and devices on the system. Even without any of the `PRIV_FILE_DAC*` discretionary access overrides, a process with uid 0 still has near complete reign of the system. It therefore stands to reason that we run as few system processes with uid 0 as possible. Rather, we run the daemons that need privileges as user `daemon` with some privileges rather than as `root` with most privileges removed.

When we look at the privileges (for the complete set used in our implementation see **privileges(5)** in appendix A), we find that many privileges are potentially too powerful as they make no distinction between objects owned by `root` and objects owned by ordinary users. And without distinction between being able to switch to just any uid or to `root`. This is where MAC labeling would really help. In keeping with our *Prevention of Privilege Escalation* policy, we will require the full set of privileges if the effective uid of a process is not 0 when that process needs a privilege to modify an object owned by uid 0 or wants to control a process with any of its uids 0. If a process switches from any uid to 0 and none of its other uids is 0, it will also require all privileges.

Ordinary file, directory and other permissions still apply to root owned objects; it is not until a privilege is needed that the additional requirement comes into force. For all other intends and purposes, root owned objects behave exactly the same as ordinary objects.

The more interesting applications of *Process Privileges* will be the use in processes not running with user id 0. Our implementation runs a few daemons with a very restricted set of privileges. We can run daemons under different uids and we document which privileges we actually use at the same time. The daemons themselves have been changed to drop the excess privileges and run under a different uid.

```
170:        /usr/sbin/rpcbind
flags = 0x2
        E: net_privaddr,proc_fork,sys_nfs
        I: none
        P: net_privaddr,proc_fork,sys_nfs
        L: all
328:        /usr/lib/nfs/statd
flags = 0x2
        E: proc_fork
        I: none
        P: proc_fork
        L: all
335:        /usr/lib/nfs/nfsd
flags = 0x2
        E: sys_nfs
        I: none
        P: sys_nfs
        L: all
330:        /usr/lib/nfs/lockd
flags = 0x2
        E: sys_nfs
        I: none
        P: sys_nfs
        L: all
```

3.6 Basic Privileges

In certain situations, customers feel that the standard Solaris user has too many privileges enabled by default. Or they might want to be able to enable unrestricted **chown(2)** for just a handful of users. Or make sure that a process never forks or execs, a feature that can be used to make any editor safe for use in a restricted menu environment. Customers using quotas sometimes want to prevent users from creating hard links to files they do not own.

To facilitate such features, we have implemented what we call *Basic Privileges*. Basic privileges are just like ordinary privileges except that the default I contains all Basic privileges. Administrators can assign different privileges to users or take the basic privileges away from specific users. The initial set of privileges, I^0 is assigned the full set of basic privileges, B. This initial set is inherited by all processes and also determines the privileges associated with ordinary users authenticated to the kernel services using a form of RPC, such as NFS.

The **system(4)** variable `rstchown` determines whether the privilege `file_chown_self` is present in I^0; the ability to change ownership of files owned by the current effective uid is

now per-process and can be awarded on a user-by-user basis.

The set of basic privileges is part of the information the kernel supplies about its configuration; therefore it is easy to determine whether a process runs with more than the basic set of privileges.

In order to allow migration of more standard functionality into the set of basic privileges, the functions that convert privilege sets to strings will expand the basic privileges into the string `"basic"` by default. If one or more privileges from B are missing from a set, the expansion will read `"basic,!basic_missing1,!basic_missing2"`, thus preserving the privileges the set conveys over future expansions of B. Options are provided to change the default behaviour for those cases where the literal privilege set is required or where the shortest output form is wanted.

In order to write a process or daemon which uses privileges that needs to be ported forward, the programmer must take care that the process or daemon requires B in addition to the other privileges they may need. Individual basic privileges that are known not to be required can be removed from P. E.g., if we would make `open(..., O_*WR*)` a basic privilege in future, such a daemon would retain that privilege simply by retaining the basic privilege set at startup. In the OS release with the additional basic privilege the process or daemon would automatically get that additional privilege.

3.7 Privileges and the runtime environment

The Solaris runtime environment consisting of the runtime linker and the runtime libraries, are user configurable to a considerable degree using mostly environment variables. The runtime environment restricts the configurability for processes that are set-uid or set-gid. Applications that completely assume different identities in sub processes, e.g., **su(1m)** and **sendmail(1m)**, clean the environment before executing sub processes that cannot determine that they run under an assumed uid, e.g., because the effective and real uid are now identical.

The runtime environment does not impose restrictions on privileged processes merely because they are privileged; only when it senses that the privileged processes derive from unprivileged processes, e.g., through set-uid.

In our proposal, this mechanism remains largely unchanged; it remains the responsibility of applications to vet the environment if privileges are moved to I. The OS will take care of those cases where extra privileges are gained through set-uid or, in the future, forced privileges.

We propose to do this by extending **issetugid(2)**. In the initial implementation, this will not be necessary as the only way to gain privileges is through set-uid 0 executables; those are already caught by the current mechanism.

The interface between the runtime linker and kernel is slightly changed; the kernel has better knowledge about how the process transitioned and how it was started; a new aux vector, AT_SUN_AUXFLAGS, is passed to the runtime linker. It's value is a bit mask for which we define a single bit, AF_SUN_SETUGID, which, when set, indicates that the runtime linker can only trust secure directories.

3.8 Privileges and NFS

The Network Filesystem, NFS, authorizes operations on files based on network credentials. In the insecure, default, mode of operation, the NFS client sends the identity of the caller directly with each requests, without giving the server any way to verify the credentials of the process on the NFS client. Proper authorizations schemes have been developed, but all of them focus on the "user" as granularity of access control. But because proper authorization is not widespread, NFS servers typically map requests by the Super-user to a guest or nobody credential. Allowing privileged operations over NFS is atypical.

There is one use of NFS where this is not fine; the use for diskless clients, including the case of network install. The former being slightly more complicated as root write access is also required.

In both cases, the NFS file systems are exported with full root access; we achieve compatibility by still having our system and install processes run as root; this gives those systems full filesystem access for those insecure exports. In those cases where daemons are converted to run as a uid other than root, the appropriate ownership or permission changes will be made.

In the future we can think of client verified, e.g., digitally signed, additional credentials that hold the effective privileges of a process. However, we believe that allowing clients to wield privileges on NFS servers is typically ill-advised.

It is NFS which prevents us from replacing the kernel tunable `rstchown` with the privilege PRIV_FILE_CHOWN_SELF altogether.

Several kernel RPC functions subject to contract 2001-699-01 had their implementation changed; the NFS specific privilege check was moved to the `nfssys` system call as it was out of place in the RPC layer. The contract parties were informed of the change and had no objections.

The NFS system generates kernel credentials from network credentials in several locations; we change the code which generates credentials and the functions to generate full credentials. This changes the signatures of certain kernel rpc functions. The function `authdes_getucred` is renamed with `kauthdes_getucred` as the former also exists in the

rpc libraries. The signature of `sec_svc_getcred` just changes.

```
/* Old signatures */
int sec_svc_getcred(struct svc_req *req, uid_t *uid, gid_t *gid,
    short *ngroups, gid_t *groups, caddr_t *principal, int *secmod)

int authdes_getucred(const struct authdes_cred *, uid_t *, gid_t *, short *,
                gid_t *);

/* New signatures */
int sec_svc_getcred(struct svc_req *req, cred_t *cr, caddr_t *principal,
   int *secmod);

int kauthdes_getucred(const struct authdes_cred *adc, cred_t *cr);
```

3.9 Privileges and Sun Cluster

The Sun Cluster software marshals credentials into packets for communication between cluster nodes in the context of filesystem operations. In discussions with the Clustering group, we found that the intention is to provide for the co-existence for short periods of time of mixed release clusters in order to facilitate rolling upgrades. Because of the semantic gap between releases containing our changes and those that do not, an exact credential mapping is not possible, just as is the case with NFS. We do not foresee any problems with making the two different operating releases talk, however, as the credential marshaling functions on the Solaris 10 side can be taught to do the right thing and have agreed to work with Clustering to make rolling upgrades possible.

3.10 Privileges and 3rd party filesystems

There is currently no documentation nor a supported mechanism for adding filesystems to Solaris; some 3rd party filesystems are available, typically using a Solaris source license or some partnering agreement. It is expected that such filesystem need to be re-released for minor releases. We have made no changes to the filesystem interfaces so pre-existing filesystems will continue to work unless they reference the `cr_groups` field directly; none of our Solaris filesystems do. But unmodified filesystems will not honor privileges; they most likely will still require uid 0 for privileged operations until they are converted to using our filesystem policy routines.

4 Proposed Interfaces

In this section we describe the details of how the *Process Privileges* implementation is layered in the kernel and the interfaces offered between various subsystems.

4.1 Bit sets and constants conspire against progress

The most convenient data structure for privileges and privilege sets are integers and bit sets. A few words of memory indexed by bit and a privilege number as an index. This is how the implementation in Trusted Solaris works as well as most implementations based on the defunct POSIX draft P1003.2c capabilities.

Early in the design review, however, it was clear that bit sets with implementation visible sizes and manifest constants will soon be an impediment to future expansion and interoperability. Either we waste memory by picking a likely upper bound for privilege sets sizes or, more likely, we will pick a number that is too small.

For efficiency reasons the core kernel still operates with fixed size bit sets and manifest constants. But the constants and set sizes are not visible to user applications or DDI-compliant kernel modules at compile time. The data structures exported by the kernel are all self-describing; they contain a header with a field that denotes the size of the header as well as the size of additional data following the header.

The kernel publishes all relevant information to user processes in that fashion. The following parameters of the system are not fixed in any of the kernel / userland or kernel / kernel interfaces:

- The number of privilege sets
- The size of the privilege sets
- The names of the privilege sets
- The number of privileges
- The name to number mapping of privileges.

These parameters are then used to configure the library interfaces to the privilege system and should not be used by user programs directly.

Applications will use privilege names and privilege set names as *strings* and the library will take care of name / number mappings in most cases; in general applications will not need to convert individual privileges to numbers.

The current implementation does fix all these values in the kernel, but they could be made fully dynamic, e.g., if we choose to allow kernel modules to allocate preposterous numbers of privileges, we could introduce a kernel parameter that would grow privilege sets to a specific size at boot.

The privilege names are not localized; the conversion of privileges and privilege sets to and from strings is locale neutral.

The library routine **priv_gettext(3C)** which maps privileges to a descriptive text is localized; it obtains the textual description from `/etc/security/priv_names` in the C locale and uses an algorithm similar to the one for **magic(4)** to find a localized version of the text messages.

Another important outcome of the early design reviews was that *privilege sets as bits* should not exist in a permanent form anywhere, except when accompanied with sufficient information to interpret the bits. We use this in one place, process core dumps.

4.2 Privilege Names and Constants

In **privileges(5)** we describe the privileges in use in our implementation. The privileges are part of the interface specification in several ways. They are available as manifest integer constants for use in the ON Consolidation; they are available as manifest string constants as a public interface.

The names of privileges are looked up in tables maintained by the kernel; there is an obvious mapping between the manifest constants and the actual privilege names. The manifest constants are upper case and prefixed with `PRIV_`. The strings themselves are lower case without the prefix. The name lookup routines are case-insensitive; they also accept the `"priv_"` prefix which is stripped before doing the actual lookup.

Some of the privileges are very specific; we believe they should be classified as stable. Some privileges are *Evolving* as they are too generic (`PRIV_SYS_CONFIG`) or might be obsoleted (`PRIV_SYS_SUSER_COMPAT`). The privilege *constants* are all classified as stable.

Privileges are logically grouped on the basis of the scope of the privilege. The `FILE` privileges operate on file system objects. The sub group `FILE_DAC` overrides *discretionary access control* on files. The `IPC` privileges override IPC object access controls. The `NET` privileges give access to specific network functionality; the `PROC` privileges allow processes to modify restricted properties of the process itself and give access to features with a process scope, such as high resolution timers, locked memory, etc. The `SYS` family gives processes unrestricted access to various system properties.

4.3 Kernel Data Structures

When implementing *Process Privileges* in the kernel it is immediately obvious that the privilege sets belong in only one place: `cred_t`; that is the data structure which currently carries the information about process privileges; it is also the data structure available at those locations where we need to test for privileges.

Extending `cred_t` also gives us our first challenge. While it can be maintained that the data structure itself *Consolidation Private*, its existence, the fact that you can have a reference to it, and some of its contents certain have a higher classification. It is a highly visible kernel data structure and often used by tools such as **lsof(1m)**[10]; since such tools typically require some work for each minor release and even for some micro and patch releases, this is not an issue.

More problematic is the use in kernel code; we have seen both 3*rd* party modules and Solaris code allocating `cred_t`'s on the stack, writing them to file or even modifying them, which should not be done as `cred_t` is a shared, refcounted, data structure which should be allocated and deallocated using the interfaces defined for it and modified using *Copy-On-Write*.

Device drivers that dereference various fields of `cred_t` are not DDI compliant; the only actions allowed by the DDI/DKI appear to be passing it on as a parameter or passing it to **drv_priv(9f)**.

We suspect most use of `cred_t` external to ON is made by 3*rd* party filesystems; since those need to be recompiled for each and every release, they are not likely to fall victim to problems caused by changes in the layout of `cred_t`. Nevertheless, we propose to make `cred_t` less visible and to clarify its classification as *Consolidation Private* by hiding the actual definition of `cred_t` in a new include file, `<sys/cred_impl.h>`, while retaining `typedef struct cred cred_t` in `<sys/cred.h>`. We further propose adding a number of new functions to access various fields of `cred_t` regardless of the actual size or composition of the structure.

This project chose not to use the interfaces defined in the *Kernel Structure Extension Mechanism*[11]; the project team strongly believes that the use of such mechanisms should be reserved for optional parts of Solaris.

[10] lsof is a public domain utility that is a combination of **fuser(1m)**, **pfiles(1m)** and more; it is available at `ftp://vic.cc.purdue.edu/pub/tools/unix/lsof`

[11] PSARC 2000/175

In our proposal, <sys/cred.h> is essentially reduced to this:

```
typedef struct cred cred_t;

int prochasprocperm(struct proc *, struct proc *, const cred_t *);
int supgroupmember(gid_t, const cred_t *);
uint_t crgetref(const cred_t *);
uid_t crgetuid(const cred_t *);
uid_t crgetruid(const cred_t *);
uid_t crgetsuid(const cred_t *);
gid_t crgetgid(const cred_t *);
gid_t crgetrgid(const cred_t *);
gid_t crgetsgid(const cred_t *);
const gid_t *crgetgroups(const cred_t *);
int crgetngroups(const cred_t *);
int crsetresuid(cred_t *, uid_t, uid_t, uid_t);
int crsetresgid(cred_t *, gid_t, gid_t, gid_t);
int crsetugid(cred_t *, uid_t, gid_t);
int crsetgroups(cred_t *, int, gid_t *);
```

For the most part, these are the obvious accessor functions; they should have a classification of *Public, Evolving* and made part of the Solaris specific DDI/DKI. The function `crgetref()`, an implementation artifact, and the `crset*()` functions are defined as *Consolidation Private*.

The new function `prochasprocperm()` behaves like the original `hasprocperm()` but takes two processes as argument; this allows the function to check for identical processes, session ids, etc.

The function `supgroupmember()` behaves like `groupmember()` but only checks the supplemental groups and not the effective group id; the existing interface did not allow **exec(2)** to determine correctly whether a process was set-gid.

By making `cred_t` an incomplete type, we can guarantee that all code which declares objects of type `cred_t` and all code which dereferences `cred_t` breaks on the first recompile, requiring further investigation by the developer. Note that such uses were suspect already as `cred_t` is a dynamically sized structure.

The intention is that this further investigation leads to using the proper interfaces as defined in <sys/cred.h>.

The second reason we have to make `cred_t` opaque is that we want the ability to evolve the implementation. Inside the kernel, process privileges are carried around as bit sets for efficiency reasons. But we don't want to fix privilege numbers, the size of privilege sets or even the number of privilege sets. Yet we want to carry them all directly in `cred_t`. We

also like to enable other Solaris projects, such as Trusted Solaris and Kevlar, to extend the credential with data types of their choosing and allow certain data structures that logically belong in `cred_t` to be moved there. Further changes to `cred_t` can then be made with *Patch* or *Micro Release Binding*.

The full `cred_t` is defined in `<sys/cred_impl.h>`:

```
struct cred {
        uint_t          cr_ref;         /* reference count */
        uid_t           cr_uid;         /* effective user id */
        gid_t           cr_gid;         /* effective group id */
        uid_t           cr_ruid;        /* real user id */
        gid_t           cr_rgid;        /* real group id */
        uid_t           cr_suid;        /* "saved" user id (from exec) */
        gid_t           cr_sgid;        /* "saved" group id (from exec) */
        uint_t          cr_ngroups;     /* number of groups returned by */
                                        /* crgetgroups() */
        cred_priv_t     cr_priv;        /* privileges */
        gid_t           cr_groups[1];   /* cr_groups; size not fixed */
};

extern int ngroups_max;
```

This new definition is not binary compatible as the `cr_groups` field was moved down.

The privilege sets are defined in `<sys/priv_impl.h>`:

```
typedef uint32_t priv_chunk_t;

typedef struct priv_set {
        priv_chunk_t    pbits[PRIV_SETSIZE];
} priv_set_t;

typedef struct cred_priv_s {
        priv_set_t      crprivs[PRIV_NSET];     /* Priv sets */
        uint_t          crpriv_flags;           /* Privilege flags */
} cred_priv_t;
```

The manifest constants `PRIV_SETSIZE` and `PRIV_NSET` are generated at kernel compile time and sized according the number of actually defined privileges and sets. For all privileges, a manifest constant is generated as well; all privilege manifest constants are consolidation private. They are included in the generated header file `<sys/priv_const.h>` which is shipped to allow *kernel browsers* to continue to compile.

Another existing kernel data structure, the STREAMS data block `dblk_t`, is changed; the `db_uid` field is replaced with a `db_credp` field, allowing us to make security policy decisions based on the sender of the message, rather than the credentials of the process opening the devices. This allow us to return to BSD socket semantics and use the privileges at **bind(3socket)** time rather than the privileges at **socket(3socket)** time to determine whether `bind()` to a privileged port can succeed. This makes Solaris more compatible with other UNIX socket implementations.

By reclaiming an unused field, dropping `db_uid`, and slightly rearranging the non-public fields of `dblk_t`, we succeeded in shrinking it by 8 bytes. Macros were defined to access the field and new functions are defined to allocate `mblk_t` with data blocks initialized either with a credential or from a template message block.

4.4 Kernel Interfaces

At the heart of the privilege code are the `priv_policy*` routines; these routines get passed a credential, a privilege to check, and possibly some additional information for debugging. The functions take care of auditing, the antiquated `ASU` flag for **acct(2)** accounting, logging and debugging. On failure, the missing privilege is recorded in the `lwp` structure.

```
int priv_policy(const cred_t *, int, int, const char *);
boolean_t priv_policy_only(const cred_t *, int);
boolean_t priv_policy_choice(const cred_t *, int);
```

These functions are generally not called directly from kernel modules as they require inlining privilege constants. We defined additional functions, `secpolicy_<name>`, to be used instead of direct calls. Most of the functions map directly onto a single privilege, but in an N-M and not a 1-1 mapping; `secpolicy_vnode_setattr()` moves all policy decisions typically found in `VOP_SETATTR` to a single function. The side effect of these changes is that all filesystems using these new interfaces will make identical policy decisions. The functions are defined in `<sys/policy.h>`; once stabilized, some of these interfaces should be available to other kernel developers. Others, such as `ipc_config`, are specific to core kernel functionality and don't warrant being public. Some should be put under a file system interface umbrella.

```
int secpolicy_acct(const cred_t *);
int secpolicy_allow_setid(const cred_t *, uid_t, boolean_t);
int secpolicy_audit_config(const cred_t *);
int secpolicy_audit_getattr(const cred_t *);
```

```
int secpolicy_audit_modify(const cred_t *);
int secpolicy_chroot(const cred_t *);
int secpolicy_clock_highres(const cred_t *);
int secpolicy_console(const cred_t *);
int secpolicy_coreadm(const cred_t *);
int secpolicy_dispadm(const cred_t *);
int secpolicy_excl_open(const cred_t *);
int secpolicy_fs_config(const cred_t *);
int secpolicy_fs_linkdir(const cred_t *);
int secpolicy_fs_minfree(const cred_t *);
int secpolicy_fs_mount(const cred_t *, vnode_t *);
int secpolicy_fs_quota(const cred_t *);
int secpolicy_ipc_access(const cred_t *, const struct kipc_perm *, mode_t);
int secpolicy_ipc_config(const cred_t *);
int secpolicy_ipc_owner(const cred_t *, const struct kipc_perm *);
int secpolicy_lock_memory(const cred_t *);
int secpolicy_modctl(const cred_t *, int);
int secpolicy_net(const cred_t *, int, boolean_t);
int secpolicy_net_config(const cred_t *, boolean_t);
int secpolicy_net_privaddr(const cred_t *, in_port_t);
int secpolicy_net_rawaccess(const cred_t *);
int secpolicy_newproc(const cred_t *);
int secpolicy_nfs(const cred_t *);
int secpolicy_pcfs_modify_bootpartition(const cred_t *);
int secpolicy_ponline(const cred_t *);
int secpolicy_power_mgmt(const cred_t *);
int secpolicy_proc_access(const cred_t *);
int secpolicy_proc_excl_open(const cred_t *);
int secpolicy_proc_owner(const cred_t *, const cred_t *, int);
int secpolicy_pset(const cred_t *);
int secpolicy_rctlsys(const cred_t *);
int secpolicy_resource(const cred_t *);
int secpolicy_rpcmod_open(const cred_t *);
int secpolicy_rsm_access(const cred_t *, uid_t, mode_t);
int secpolicy_setpriority(const cred_t *);
int secpolicy_settime(const cred_t *);
int secpolicy_spec_open(const cred_t *, struct snode *, int, vtype_t);
int secpolicy_sti(const cred_t *cr);
int secpolicy_sys_config(const cred_t *, boolean_t);
int secpolicy_sys_devices(const cred_t *);
int secpolicy_tasksys(const cred_t *);
int secpolicy_vnode_access(const cred_t *, vnode_t *, uid_t, mode_t);
int secpolicy_vnode_create_gid(const cred_t *);
int secpolicy_vnode_owner(const cred_t *, uid_t);
int secpolicy_vnode_remove(const cred_t *);
```

```
int secpolicy_vnode_setdac(const cred_t *);
int secpolicy_vnode_setid_retain(const cred_t *, boolean_t);
int secpolicy_vnode_setids_setgids(const cred_t *, gid_t);
int secpolicy_vnode_stky_modify(const cred_t *cr);

int secpolicy_basic_exec(const cred_t *);
int secpolicy_basic_fork(const cred_t *);
int secpolicy_basic_proc(const cred_t *);
int secpolicy_basic_link(const cred_t *);

int secpolicy_vnode_setattr(const cred_t *, struct vnode *, struct vattr *,
    const struct vattr *, int, int iaccess(/* void *, int, cred_t **/), void *);
```

The privilege checks in the kernel are all replaced with calls to the appropriate `secpolicy*()` functions which are passed sufficient arguments, always including the current process credential and often more information, such as a pointer to the object the operation is performed on. Different security policy functions may map to a check for the presence of the same privilege.

Privilege numbers, privilege set numbers and set sizes are *Consolidation Private*. Other components must call the kernel policy functions; if a driver needs to obtain the number of a specific privilege, the driver can lookup its number using **priv_getbyname(9f)**.

The function **priv_getbyname(9f)** also allows kernel modules to allocate new privileges by specifying `PRIV_ALLOC` as `flags` argument. Privileges allocated using this function are limited in size to `PRIVNAME_MAX`(32) characters and can only contain alphanumeric characters and underscores. Privilege names are case insensitive but case preserving. The number of slots for allocating new privilege is limited by both the number of unaccounted bits in the bit sets and the amount of memory reserved for the additional privilege names. While we advise an algorithm to pick unique names, non-unique privileges names will not cause fatal clashes of any kind; the ”clashing” privilege will allow a process to perform both restricted operations, taking just a little bit out of the “Least” in *Least Privilege*.

At this time, the secpolicy functions are a private implementation detail and the interface is unstable. The priv_policy functions are intended for public consumption.

4.5 System Call Interfaces

The privilege system defines a number of new system calls; **getprivinfo(2)** returns a self-describing data structure which contains the parameters of the privilege implementation on the currently running kernel. These parameters include the number of privilege sets, the

names of the privilege sets, the size of each privilege set and the names of all privileges and other system wide information. The privilege set size is specified in units of `priv_chunk_t`, the virtual privilege state definition looks like:

```
priv_chunk_t privs[info.priv_nsets][info.priv_setsize]
```

Even though additional privileges may be allocated later, the data structure returned has a fixed size so it can be kept at the same location by libc thus obviating the need for locking out accesses to those parts of the structure that are fixed by the implementation; i.e., all characteristics of the implementation except for the number of privileges and the names of the privileges added later.

The system include file `<sys/priv.h>` defines the main data structures used. The `priv_impl_info` objects can be extended by one or more objects with a `priv_info` header which contains length and size. The basic type used throughout is `uint32_t` which is a convenient, type with the same size in each compilation environment. This relieves the implementation of most of the 32/64 bit conversion chores.

```
typedef struct priv_impl_info {
        uint32_t        priv_headersize;        /* sizeof (priv_impl_info) */
        uint32_t        priv_flags;             /* additional flags */
        uint32_t        priv_nsets;             /* number of priv sets */
        uint32_t        priv_setsize;           /* size in priv_chunk_t */
        uint32_t        priv_max;               /* highest actual valid priv */
        uint32_t        priv_infosize;          /* Per proc. additional info */
        uint32_t        priv_globalinfosize;    /* Per system info */
} priv_impl_info_t;

/*
 * Header of the privilege info data structure; multiple structures can
 * follow the privilege sets and priv_impl_info structure.
 */
typedef struct priv_info {
        uint32_t        priv_info_type;
        uint32_t        priv_info_size;
} priv_info_t;

typedef struct priv_info_uid {
        priv_info_t     info;
        uid_t           uid;
} priv_info_uid_t;

typedef struct priv_info_uint {
```

```
        priv_info_t     info;
        uint_t          val;
} priv_info_uint_t;

/*
 * Global privilege set information item; the actual size of the array is
 * {priv_setsize}.
 */
typedef struct priv_info_set {
        priv_info_t     info;
        priv_chunk_t    set[1];
} priv_info_set_t;

/*
 * names[1] is a place holder which can contain multiple NUL terminated,
 * non-empty strings.
 */

typedef struct priv_info_names {
        priv_info_t     info;
        int             cnt;            /* number of strings */
        char            names[1];       /* "string1\0string2\0 ..stringN\0" */
} priv_info_names_t;

/*
 * Privilege information types.
 */
#define PRIV_INFO_SETNAMES              0x0001
#define PRIV_INFO_PRIVNAMES             0x0002
#define PRIV_INFO_BASICPRIVS            0x0003
#define PRIV_INFO_FLAGS                 0x0004
```

The system calls **setppriv(2)** and **getppriv(2)** allow a process to change and inspect its privilege sets.

The system calls **setpflags(2)** and **getpflags(2)** allow a process to change and inspect the process flags such as *pas* and *db*.

The system call `modctl()` is extended with a number of subcodes to allow the device configuration command **devfsadm(1m)** to install the device policy and to allow the allocation of additional privileges. It too is subjected to *escalation of privilege prevention* in that only processes with all privileges asserted can change the device policy.

4.6 Library Interfaces

In this section we describe two different sets of interfaces; one set of interfaces for manipulating privilege set and a second set of interfaces that abstracts the kernel credentials, `cred_t`, into an opaque user credential, `ucred_t`, with a functional interface.

4.6.1 Privilege Specific Library Interfaces

The library interfaces are primarily for manipulating privilege sets and converting from privilege names to numbers and back. As the implementation protects the programmer from the details of the implementation in the currently running kernel, functions are provided to allocate sufficient memory for privilege sets as well as functions that take privileges and privilege sets and do computations.

The library loads the privilege implementation details the first time it is required. It is kept around for future calls to the library. When library calls query any part of the structure that may have been changed, the library checks with the kernel and updates the internal information if necessary. An application will learn about privileges added after it first caused the implementation details to be loaded by the C library. A *Consolidation Private* secondary set of functions that accepts the implementation details as argument is provided for use by **libproc(4)** and other programs inspecting core dumps.

The manifest constants, defined by including `<priv.h>` in user code, map to strings only. The manifest constants are present to detect typos in privilege names early. They constants are cast to `(const char *)` to allow compilers to merge identical strings and to prevent accidental string concatenation.

```
#define PRIV_IPC_DAC_READ       ((const char *)"ipc_dac_read")
#define PRIV_IPC_DAC_WRITE      ((const char *)"ipc_dac_write")
#define PRIV_IPC_OWNER          ((const char *)"ipc_owner")
#define PRIV_SYS_IPC_CONFIG     ((const char *)"sys_ipc_config")
```

The privilege sets are represented as opaque pointers; applications should only declare pointers to `priv_set_t` and allocate, free and manipulate them using appropriate functions, keeping the actual structure hidden from view. Utility functions that parse strings representing privilege sets and convert privilege sets back to strings are provided both for pretty printing, ease of input and permanent storage of privilege sets. The strings can contain those privileges currently defined in the kernel and special tokens such as `all` for *P*, `basic` for *B*, none for no privileges. A single dash "-" or exclamation mark "!" preceding a privilege negates its presence, allowing short-hand notation for *all privileges except ...*. The implementation prefers "!" as the negation character on output but will use "-" if the exclamation mark

is specified as separator. Both can be used on input; the exclamation mark looks nicer but needs to be escaped by certain shells; the dash can be more convenient in such cases.

The special token `all` represents the set of all bits set, not just those privilege defined in the system. Similarly, tests against the full set and the set comparison and manipulation functions always take all bits of the underlying implementation into account even those that do not represent valid privileges.

A full set of functions to manipulate sets in user code is provided in `<priv.h>`, including a number of functions that allow manipulating individual privileges easily:

```
int setppriv(priv_op_t, priv_ptype_t, const priv_set_t *);
int getppriv(priv_ptype_t, priv_set_t *);
int setpflags(uint_t, uint_t);
uint_t getpflags(uint_t);
const priv_impl_info_t *getprivimplinfo(void);

int priv_set(priv_op_t, priv_ptype_t, ...);
boolean_t priv_ineffect(const char *);
priv_set_t *priv_str_to_set(const char *, const char *, const char **);
char *priv_set_to_str(const priv_set_t *, char, int);

int priv_getbyname(const char *);
const char *priv_getbynum(int);
int priv_getsetbyname(const char *);
const char *priv_getsetbynum(int);
char *priv_gettext(const char *);

priv_set_t *priv_allocset(void);
void priv_freeset(priv_set_t *);

void priv_emptyset(priv_set_t *);
void priv_fillset(priv_set_t *);
boolean_t priv_isemptyset(const priv_set_t *);
boolean_t priv_isfullset(const priv_set_t *);
boolean_t priv_isequalset(const priv_set_t *, const priv_set_t *);
boolean_t priv_issubset(const priv_set_t *, const priv_set_t *);
void priv_intersect(const priv_set_t *, priv_set_t *);
void priv_union(const priv_set_t *, priv_set_t *);
void priv_inverse(priv_set_t *);
int priv_addset(priv_set_t *, const char *);
void priv_copyset(const priv_set_t *, priv_set_t *);
int priv_delset(priv_set_t *, const char *);
boolean_t priv_ismember(const priv_set_t *, const char *);
```

The name to number and back mapping functions are provided for completeness but they are not generally useful. The other interfaces discourage the use of privileges and sets as numbers; no function takes privileges or sets as numbers except for the mapping functions. The actual size of privilege sets can only be determined using **getprivimplinfo(3c)**; writing privilege sets to files as bits requires getting the set and obtaining the system specific set size. The interface to convert a privilege set to a string is more convenient.

The short hand privilege bracketing functions allow you to switch privileges on or off without having to do the set allocation and other grunt work by hand using variable argument functions with `NULL` as terminator:

```
priv_set(PRIV_ON, PRIV_EFFECTIVE, PRIV_NET_PRIVADDR, NULL);

bind(... reserved port ...);

priv_set(PRIV_OFF, PRIV_EFFECTIVE, PRIV_NET_PRIVADDR, NULL);

/* Permanently giving up our privilege */

priv_set(PRIV_OFF, PRIV_PERMITTED, PRIV_NET_PRIVADDR, NULL);
```

4.6.2 User Credential Library Interfaces

For certain types of IPC, there is a distinct need for the daemon handling the request of the client to establish the client's credentials. Several mechanisms have been used in the past; mechanisms such as *reserved ports*, where the client had to be a privileged process were discarded long ago; newer mechanisms use trusted kernel mechanisms to forward an ad-hoc subset of the kernel credential of the client process to the server process such as **door_cred(3door)** and private interfaces such as `__rpc_get_local_uid()` which was soon replaced with a more complete but still inextensible `__rpc_get_local_cred()`.

As we are extending the kernel credential, it stands to reason that we also try to rationalize the userland representation of the kernel credential and make it extensible.

In order to obtain maximum flexibility, we define an opaque type `ucred_t`, size unknown, and the only way to retrieve information from this opaque type is by using access functions as defined in `<ucred.h>`. The functions may return `-1` or `NULL` if the underlying user credential is incomplete as underlying interfaces may not provide a complete user credential. For completeness we add a function **ucred_get(3c)** to retrieve the complete user credential from any process through `/proc`.

We implement the old **door_cred(3door)** interface on top of these new interfaces and mark it obsolete. The rpc functions are not extended; not only are they private, the rpc credentials include only user and group ids across the board.

```
typedef struct ucred_s ucred_t;

ucred_t *ucred_get(pid_t pid);

void ucred_free(ucred_t *);

uid_t ucred_geteuid(const ucred_t *);
uid_t ucred_getruid(const ucred_t *);
uid_t ucred_getsuid(const ucred_t *);
gid_t ucred_getegid(const ucred_t *);
gid_t ucred_getrgid(const ucred_t *);
gid_t ucred_getsgid(const ucred_t *);
int   ucred_getgroups(const ucred_t *, const gid_t **);

const priv_set_t *ucred_getprivset(const ucred_t *, priv_ptype_t);
uint_t ucred_getpflags(const ucred_t *, uint_t);

pid_t ucred_getpid(const ucred_t *);
```

4.6.3 Private Daemon and Set-uid Interfaces

To get a higher return on the initial Least Privilege implementation, we add two private utility interfaces, `__init_daemon_priv()` and `__init_suid_priv()`. These interfaces allow daemons to specify which privileges they need on top of B. The library calls put those privileges in P and E and resets the effective uid for set-uid applications or sets a specific daemon uid and clears the group list for a daemon. If requested, the library routine also clears I and L or sets I equal to P. The basic set is preserved insofar present in the inheritable set to keep us from requiring code changes for every basic privilege added. Utilities and daemons can revoke those basic privileges that existed at the time the code was written and aren't needed by the program. In the following example, the set-uid program never needs to call **fork(2)** or **exec(2)** so it rescinds those privileges. The added benefit is that when the program is exploited the exploit code will need to run in the original executable as execing a shell is not possible. The function is a no-op when run by a process that otherwise posses all privileges, e.g., processes started by the Super-user.

The special purpose function `__priv_bracket()` switches the privileges requested by the initial call to `__init_suid_priv()` on and off by adding them to E or removing them

from *E* depending on the argument; `__priv_relinquish()` gives up the requested privileges permanently by removing them from *P*.

By making judicious use of `__priv_bracket()` in the **rcmd(3socket)** implementation, set-uid root programs that need to use those function can be adapted with ease: add one call to `__init_suid_priv()` and remove all the application's uid juggling code.

```
/* Returns with the required privilege in P but not in E */
__init_suid_priv(PRIVUTIL_CLEARLIMIT, PRIV_NET_RAWACCESS, NULL);


/* Basic privileges not needed by this program */
priv_set(PRIV_OFF, PRIV_ALLSETS, PRIV_PROC_EXEC, PRIV_PROC_FORK, NULL);

....

/* Need privilege for raw socket, adds it to E */
__priv_bracket(PRIV_ON);

s = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);

/* Don't need it for a while, remove it from E */
__priv_bracket(PRIV_OFF);

/* or don't need the privilege ever again, remove it from P and E */
__priv_relinquish();
```

Similarly, this daemon runs with as few privileges as required. A possible exploit also needs to run in the context of this process; it cannot escape and run a different executable.

```
/* NFS server: needs NFS privilege and access to the FX scheduler */

__init_daemon_priv(PRIVUTIL_CLEARLIMIT, DAEMON_UID, DAEMON_GID,
                PRIV_SYS_NFS, PRIV_PROC_PRIOCNTL, NULL);

/* Basic privileges I do not need */
priv_set(PRIV_OFF, PRIV_ALLSETS, PRIV_PROC_EXEC, PRIV_PROC_FORK,
        PRIV_PROC_SESSION, NULL);
```

4.7 Enhancements to RBAC

The RBAC[12] framework allows administrators to define *profiles* for users and roles as well as attributes for executables when running as part of a profile.

We extend the **exec_attr(4)** database with a new attribute as part of our project. The attribute specifies I for the program to run with; some existing entries can make immediate use of this feature:

```
Object Access Management:solaris:cmd:::/usr/bin/chmod:privs=file_setdac
Object Access Management:solaris:cmd:::/usr/bin/chgrp:privs=file_chown
Object Access Management:solaris:cmd:::/usr/bin/setfacl:privs=file_setdac
Object Access Management:solaris:cmd:::/usr/bin/chown:privs=file_chown
```

In addition, we renamed the current Solaris default policy from `suser`, which reflects the *old world* to `solaris`, which reflects the *new world*. The new world has only one model, the solaris model, which can evolve and incorporate new security attributes over time. This change is derived from Trusted Solaris, with the difference that we allow only symbolic and not numeric privileges. Old `suser` entries are kept on upgrade in case the file is used as a source for a name service supporting multiple Solaris releases; new `solaris` entries are added for commands which run correctly with a few privileges supplied. When matching an entry the code will first look for a `solaris` entry; if none is found, the code will try `suser` instead. This is also to support sharing information between old and new systems.

The project team found that the current procedures for upgrading the RBAC files are defunct; it is not possible to remove entries or attributes on upgrade and the output appears in random order in the file, making it difficult to edit. We improve on this situation by sorting the output in all cases.

A new feature which is specifically added for the benefit of those privileges that you might trust some users with but not all, are privilege specifications in **user_attr(4)**. The two keywords are `limitpriv`, L for the user's processes, `defaultpriv`, the inheritable privileges the user logs in with by default. The default for `defaultpriv` is I^0 and the default for `limitpriv` is P. Special care must be taken to document `limitpriv`; over eager restrictions may cause certain programs such as **dtsession(1)** to fail to unlock a user's terminal.

Additionally, privileges can be assigned to profiles in **prof_attr(4)**; users can use the privileges granted to their profiles by using **pfexec(1)**.

```
Real Time:::User with real time privileges:\
                privs=proc_priocntl,sys_cpu_config,proc_clock_highres
```

[12] PSARC 1997/332

These attributes can be manipulated using a single new option to the commands **useradd(1m)**, **usermod(1m)**, **roleadd(1m)**, **rolemod(1m)** as well as the SMC GUI tools and commands **smuser(1m)** and **smrole(1m)**.

We extend **policy.conf(4)** with two new keywords, `PRIV_LIMIT` and `PRIV_DEFAULT`. These specify the defaults for the two new `user_attr` keywords.

This example allows user *casper* to use high resolution timers. But it disallows all sessions originating from his logins to use the `PRIV_SYS_LINKDIR` privilege; i.e., he can not make hard links to directories or unlink directories, even after **su(1m)** to `root`; the first example show how to achieve this using **usermod(1m)**, the second example is the resulting **user_attr(4)** entry which lines wrapped for ease of reading.

```
# usermod -K defaultpriv=basic,proc_clock_highres \
        -K limitpriv=all,!sys_linkdir casper
```

```
casper::::type=normal;defaultpriv=basic,proc_clock_highres;\
        limitpriv=all,!sys_linkdir
```

It can be argued that some privileges should not be in the default limit set. A prime candidate would be "`sys_linkdir`" as there is no excuse for making hardlinks to directories.

It is noted in the manual page that using `limitprivs` is a useful but dangerous piece of rope handed to the system administrator. As long as B is preserved all processes with running with an effective uid not equal to 0 will run properly; processes that need to run with euid 0 may experience difficulties. The system administrator may very well intend for this to happen, but in some cases the side effects can be unintended. Looking over the set-uid root applications shipped with Solaris, we find that most applications require either a simple single privileges such as `PRIV_NET_PRIVADDR` or access require access to root owned files for which the effective uid of 0 suffices still.

It becomes more important for set-uid root applications to do proper error checking; in those specific cases where lack of error detection could result in either missing audit records or failure to give up privileges in the traditional way, we prevent the execution of set-uid 0 executables.

4.8 Enhancements to proc(4) and core dumps

The process privileges and flags will be made visible as a new entry in the `/proc/<pid>` directory, `priv`. A new utility, **ppriv(1)** is provided to examine and set process privileges. The same information is made available in the ELF note section of core dumps; additionally, core dumps contain the annotated information returned by **getprivimplinfo(3c)** in order to allow

ppriv(1) and debuggers to interpret the privilege set information included in core dumps. Two new ELF notes are introduced, NT_PRPRIV and NT_PRPRIVINFO, **gcore(1)** and the kernel are enhanced to add these notes to core dumps. This allows ppriv and other utilities to show the proper number of properly sized sets and to correctly map the bits in the bit sets to privilege names. Private interfaces in libc allow libproc to leverage the libc privilege set conversion routines for core dumps. The data structure used for PCSPRIV is another example of a self-describing data structure; the header expresses some fixed quantities from which the header size can be derived and the size of the multiple instances of priv_info_t that can follow it:

```
#define PCSPRIV  29L    /* set process privileges from prpriv_t argument */

/*
 * Process privileges.  PCSPRIV and /proc/<pid>/priv
 */
typedef struct prpriv {
        uint32_t        pr_nsets;               /* number of privilege set */
        uint32_t        pr_setsize;             /* size of privilege set */
        uint32_t        pr_infosize;            /* size of supplementary data */
        priv_chunk_t    pr_sets[1];             /* array of sets */
} prpriv_t;
```

As standards dictate we cannot change error codes returned in errno, those codes remain the same. But the lwpstatus_t structure defined in <sys/procfs.h> which is used by **truss(1)** to report system call return values can be extended, and we added a pr_errpriv field in place of one of the filler fields. It is set to PRIV_NONE if there are no missing privileges, the missing privilege if there is only one, PRIV_ALL when all privileges were required or PRIV_MULTIPLE when more than one privilege was required.

4.9 Privilege debugging

Process Privileges radically change the way in which privileged operations work; while we plan on keeping *just slap uid 0 on* working for the time being, it is important to be able to determine exactly which privileges are missing when trying to minimize the set of privileges for a new application. Privilege debugging can also be used to determine exactly what privileges specific Solaris utilities need for specific tasks.

During the development of our prototype we found that logging of failures alone was sufficient to compute the set of privileges needed for something as complex as a *nightly* build.

We have therefore introduced a per-process flag, `PRIV_DEBUG`, which causes privilege failures to be logged using kernel `printf()`. These messages then appear on the terminal associated with the process. `PRIV_DEBUG` can be set and unset with **setpflags(2)**. We have also introduced a global kernel tunable `priv_debug`, settable through **system(4)** or using **mdb(1)**, which turns on privilege failure logging for all applications using **cmn_err(9f)** which allows the capture of privilege debugging information in the system logs.

The **ppriv(1)** utility allows you to set and unset the per-process flag and run processes with the flag set. We can use this feature both to determine exactly which privileges are required for certain actions or what privileges are actually used by certain set-uid applications by running them as plain executables.

```
$ cat /etc/shadow
cat: cannot open /etc/shadow
$ ppriv -e -D cat /etc/shadow
cat[12341]: missing privilege "file_dac_read" (euid = 21782), needed at
         ufs_iaccess+0xfc
cat: cannot open /etc/shadow

$ cp /usr/sbin/ping /tmp
$ /tmp/ping localhost
/tmp/ping: socket Permission denied
$ ppriv -e -D /tmp/ping localhost
ping[12373]: missing privilege "proc_setid" (euid = 21782), needed at
        seteuid+0x76
ping[12373]: missing privilege "net_rawaccess" (euid = 21782), needed at
        icmp_open+0xd
/tmp/ping: socket Permission denied
```

The **seteuid(2)** failure can directly be attributed to old-style privilege bracketing done by **ping(1m)**; the return value is obviously not checked.

To make privilege failures more *in your face* for the uninitiated users of solaris enhanced with privileges, we use our extended `/proc` interfaces and report the missing privilege after

the error code in truss output.

```
% truss -t open cat /etc/shadow
open("/var/ld/ld.config", O_RDONLY)             Err#2 ENOENT
open("/usr/lib/libc.so.1", O_RDONLY)            = 3
open("/usr/lib/libdl.so.1", O_RDONLY)           = 3
open("/usr/lib/locale/en_US/en_US.so.2", O_RDONLY) = 3
open64("/etc/shadow", O_RDONLY)                 Err#13 EACCES [file_dac_read]
cat: cannot open /etc/shadow

% truss -t so_socket /tmp/ping localhost
so_socket(PF_INET6, SOCK_DGRAM, IPPROTO_IP, "", SOV_XPG4_2) = 5
so_socket(PF_INET6, SOCK_RAW, IPPROTO_ICMPV6, "", SOV_XPG4_2)
                                                Err#13 EACCES [net_rawaccess]
/tmp/ping: socket Permission denied
```

4.10 Privilege auditing

The project introduces a system call that needs auditing, **setppriv(2)**; it also adds two new opcodes to `modctl()` that need auditing too. Three new audit events, AUE_SETPPRIV, AUE_MODALLOCPRIV and AUE_MODDEVPLCY are generated, respectively for changing privilege sets, adding a privilege from outside the kernel and changing the device policy.

Solaris keeps track of whether `suser()` was called and whether it was called successfully or not and the information is recorded in the audit trail; many “privilege checks”, such as **drv_priv(9f)**, turned out to be direct comparisons of the effective uid to zero which are not recorded in the audit trail.

Least Privilege takes a cue from *Trusted Solaris* and records the privileges the process used or failed to use in the audit trail. The privileges are recorded in their textual representation. The successful use of basic privileges is not audited. The attempt to use a basic privilege that is missing is audited as it can be considered an event that should not have taken place. The already defined audit modifier tokens AUT_PRIV and AUT_UPRIV are used for that purpose.

4.11 Device Protection

Several of the restricted system interfaces are currently protected using discretionary access control on the `/dev/*` entries. In a privilege scenario, this has several shortcomings; privileges that override DAC might give full access to all of the system. File permissions still influence who can use raw sockets and such.

We define an additional level of access control, configured with **devfsadm(1m)** and in file `/etc/security/device_policy`. This is loosely modeled after Trusted Solaris **device_policy(4)**.

This new interface allows us to bind privilege sets with the open for reading or writing of devices. E.g., in the case of `/dev/ip` we change the file permission to mode 0666 but we require the `net_rawaccess` privilege to open the device. The intention is that the device policy replaces the current hard coded checks in the open routines of device drivers, allowing administrators more flexibility in granting users permission to open devices then in the current situation where the file permissions are sometimes amended with hard coded Super-user checks in the device open routine. In the particular case of `/dev/ip` this happens on the expense of users in group `sys`; they are no longer allowed to open the device, e.g., to read MIB2 information. We believe that his is a proper change to make as those users can currently send raw IP packets circumventing the implied security policy which is that only the Super-user can send such packets. Applications which need MIB2 information typically open `/dev/ip` and push the `arp`, `tcp` and `udp` STREAMs modules. The applications should open `/dev/arp` and push only `tcp` and `udp` instead. This requires no privileges.

Access to a number of devices is currently restricted to the Super-user. Where such access can lead escalation of privileges, we add access controls requiring those privileges that can be gained; typically that would be all privileges for writing, discretionary access control for reading. E.g., when opening any of the kernel memory devices for writing, the default device policy will require the process performing the **open(2)** to have all privileges.

The **add_drv(1m)**, **rem_drv(1m)** and **update_drv(1m)** commands are enhanced to allow the addition an removal of device policy entries and driver specific privileges, either for the device policy or for the driver proper, when installing, removing or updating a device driver. The **getdevpolicy(1m)** command allows users and administrators to query the system device policy and the device policy in effect for specific devices.

When the system boots, access to all devices is restricted until **devfsadm(1m)** is run for the first time during the boot sequence and the default policy is relaxed. The initial policy is designed to be fail-safe. The upgrade routines will relax the device permissions on a number of devices in order to allow for privilege only access controls; by making the initial policy as strict as possible, a missing devpolicy file will not allow all users to send out unrestricted IP packets. Rather, it will prevent all users except the Super-user from initiating connections. This failure mode is similar to the situation that arose when we introduced **soconfig(1m)**; the failure is obvious and needs to be corrected. This is vastly preferred over a failure mode where users have the run of the system with nobody noticing that anything is amiss for some time.

The **device_policy(4)** and **extra_priv(4)** file formats are private but a pseudo manual page is attached to this document for reference purposes.

4.12 Trusted Solaris

One of the benefits of our project is that it will be easier to implement Trusted Solaris on the new kernel than on previous Solaris kernel; the project team believes that only minor modifications are needed to make the current Solaris code that depends on the specifics of our implementation (privileges, set sizes), into a separate kernel module. We have not done this as part of our project because we strongly believe that we could not fully define the interfaces needed by Trusted Solaris without first implementing the bulk of it. We have, however, closely modeled some of our interfaces on the implementation of Trusted Solaris 8 and expended some effort in making the sizes of privilege sets, the actual privileges defined and their numerical values, visible in as few locations as possible.

Our suggestion to the team to implement Trusted Solaris on a kernel with *Least Privilege* is to link the OS policy like some modules; e.g., link `kernel/unix` with `policy/$POLICY` and have the kernel linker expand the variable, set in **system(4)**, of the policy of choice.

5 Documentation changes required

The documentation changes are logically divided in two groups; new documentation describing privilege semantics and interfaces and changes to old documentation that references privileged operations.

5.1 New documentation

The bulk of the new documentation should be an answerbook section on privilege semantics and the commands to set/inspect privileges.

In addition, new manual pages for **ppriv(1)**, the system call and library interfaces need to be written. Proposed new manual pages can be found in Appendix A.

5.2 Changes to existing documentation

Many of the existing manual pages need to be changed. The **fork(2)** and **exec(2)** manual pages need to specify that privileges are copied on `fork()` and modified in a specific way on `execve()`.

References to the super-user in manual pages should be changed to reflect the actual privileges needed with a reference to **privileges(5)** which explains the basics of privileges in detail.

```
--- chroot.2    Thu Oct  3 15:50:15 2002
+++ chroot.2.new        Thu Oct  3 16:19:27 2002
@@ -20,11 +20,11 @@
      fildes argument to fchroot() is the open file descriptor of the
      directory which is to become the root.

-     The effective user ID of the process must be super-user to change
-     the root directory. While it is always possible to change to the
-     system root using the fchroot() function, it is not guaranteed to
-     succeed in any other case, even should fildes be valid in all
-     respects.
+     The privilege {PRIV_PROC_CHROOT} must be asserted in the effective
+     set of the process to change the root directory. While it is
+     always possible to change to the system root using the fchroot()
+     function, it is not guaranteed to succeed in any other case, even
+     should fildes be valid in all respects.

      The ".." entry in the root directory is interpreted to mean the
      root directory itself. Therefore, ".." cannot be used to access
@@ -79,11 +79,11 @@
      ENOTDIR
            Any component of the path name is not a directory.

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM The {PRIV_PROC_CHROOT} privilege is not asserted in the
+           effective set of the calling process.

 SEE ALSO
-     chroot(1M), chdir(2)
+     chroot(1M), chdir(2), privileges(5)

 WARNINGS
      The only use of fchroot() that is appropriate is to change back
```

Device driver and kernel programming documentation needs to be adapted to emphasize that `cred_t` is an opaque type and that dereferencing fields is no longer good practice. Changes to the privileges implementation need to be communicated to companies like Veritas which develop filesystems in the continuing absence of documentation that offers supported interfaces for filesystem development.

A selection of changed manual pages in Appendix B.

5.3 Internal documentation needs

The documentation for Solaris developers at Sun needs to be changed to reflect the changes in the world necessitated by the introduction of privileges in Solaris. The use of `suser()` and `cred_t` must be discouraged; **drv_priv(9f)** is still a proper interface to use but the manual page now points to the in some cases more appropriate **add_drv(1m)** page.

Over time devices have been added that restrict access in **open(9e)** through **drv_priv(9f)** and device permissions at the same time. Such restriction should be replaced by entries in **device_policy(4)** and the device permissions should be relaxed to mode 0666.

Some projects may want define their own privileges; privileges can be added at any time; in minor or micro releases and even dynamically during or after booting. We advise not to renumber the predefined privileges in micro releases as that would require recompiling and shipping all the kernel modules that use the constants directly; the same applies to the size of the sets. It is, however, possible to change both in a micro release because of the way interfaces avoid making the implementation details visible and their commitment levels. Changing the size of privilege sets and thus `cred_t` should probably be avoided because of tools like **lsof(1m)**.

The flag day notice that will accompany this project will provide ON developers with information they need and pointers to the project pages and draft manual pages.

6 Interface tables

6.1 Commandline interfaces

Property	*Object*	*Commitment*	*Origin*
Location	/usr/bin/gcore	Stable	Imported
	/usr/bin/pfexec	Stable	Imported
	/usr/bin/ppriv	Stable	Exported
	/usr/sbin/getdevpolicy	Stable	Exported
	/usr/sbin/roleadd	Stable	Imported
	/usr/sbin/rolemod	Stable	Imported
	/usr/sbin/useradd	Stable	Imported
	/usr/sbin/usermod	Stable	Imported
	/usr/sbin/add_drv	Stable	Imported
	/usr/sbin/rem_drv	Stable	Imported
	/usr/sbin/update_drv	Stable	Imported
Command line options	pfexec	Evolving	Imported
	ppriv	Evolving	Exported
	roleadd	Evolving	Imported
	rolemod	Evolving	Imported
	useradd	Evolving	Imported
	usermod	Evolving	Imported
	add_drv	Evolving	Imported
	rem_drv	Evolving	Imported
	update_drv	Evolving	Imported
	getdevpolicy	Evolving	Exported
Supporting file locations	/etc/pam.conf	Stable	Imported
	/etc/project	Stable	Imported
	/etc/user_attr	Stable	Imported
	/etc/security/policy.conf	Stable	Imported
	/etc/security/exec_attr	Stable	Imported
	/etc/security/extra_privs	Project Private	Exported
	/etc/security/device_policy	Project Private	Exported
	/etc/security/priv_names	Stable	Exported
	{LC_MESSAGES dir}/priv_names	Stable	Exported
	/etc/security/prof_attr	Stable	Imported

6.2 Libraries and Shared Objects

Property	*Object*	*Commitment*	*Origin*
Location	`/usr/lib/libc.so.1`	Stable	Imported
	`/usr/lib/libdoor.so.1`	Stable	Imported
	`/usr/lib/libproc.so.1`	Consolidation Private	Imported
	`/usr/lib/libproject.so.1`	Consolidation Private	Imported
	`/usr/include/priv.h`	Stable	Exported
	`/usr/include/ucred.h`	Stable	Exported
	`/usr/include/sys/elf.h`	Stable	Imported
	`/usr/include/sys/priv.h`	Stable	Exported
	`/usr/include/sys/priv_const.h`	Project Private	Exported
	`/usr/include/sys/priv_impl.h`	Project Private	Exported
	`/usr/include/sys/priv_names.h`	Stable	Exported
	`/usr/include/sys/policy.h`	Stable	Exported
	`/usr/include/sys/cred.h`	Stable	Imported
	`/usr/include/sys/cred_impl.h`	Project Private	Exported

Property	*Object*	*Commitment*
Function signature	setppriv	Evolving
	getppriv	Evolving
	setpflags	Evolving
	getpflags	Evolving
	getprivimplinfo	Evolving
	priv_set	Evolving
	priv_ineffect	Evolving
	priv_str_to_set	Evolving
	priv_set_to_str	Evolving
	priv_getbyname	Evolving
	priv_getbynum	Evolving
	priv_getsetbyname	Evolving
	priv_getsetbynum	Evolving
	priv_allocset	Evolving
	priv_freeset	Evolving
	priv_emptyset	Evolving
	priv_fillset	Evolving
	priv_isemptyset	Evolving
	priv_isfullset	Evolving
	priv_isequalset	Evolving
	priv_issubset	Evolving
	priv_intersect	Evolving
	priv_union	Evolving
	priv_inverse	Evolving
	priv_addset	Evolving
	priv_copyset	Evolving
	priv_delset	Evolving
	priv_ismember	Evolving
	pam_unix_cred'pam_sm_setcred	Evolving
	_priv_getbyname	Project Private
	_priv_getbynum	Project Private
	_priv_getsetbyname	Project Private
	_priv_getsetbynum	Project Private
	_priv_ismember	Project Private
	_priv_parse_info	Project Private
	_priv_free_info	Project Private
	_priv_set_to_str	Project Private

Property	*Object*	*Commitment*
Function signature	`_init_suid_priv`	Consolidation Private
	`_init_daemon_priv`	Consolidation Private
	`_priv_bracket`	Consolidation Private
	`_priv_relinquish`	Consolidation Private
	`Ppriv`	Consolidation Private
	`Pprivinfo`	Consolidation Private
	`Psetpriv`	Consolidation Private
	`door_ucred`	Evolving
	`ucred_geteuid`	Evolving
	`ucred_getruid`	Evolving
	`ucred_getsuid`	Evolving
	`ucred_getegid`	Evolving
	`ucred_getrgid`	Evolving
	`ucred_getsgid`	Evolving
	`ucred_getgroups`	Evolving
	`ucred_getprivset`	Evolving
	`ucred_getpflags`	Evolving
	`ucred_get`	Evolving
	`ucred_free`	Evolving

6.3 Privilege types and constants

Property	*Object*	*Commitment*
Privilege constants	PRIV_FILE_CHOWN	Stable
	PRIV_FILE_CHOWN_SELF	Stable
	PRIV_FILE_DAC_EXECUTE	Stable
	PRIV_FILE_DAC_READ	Stable
	PRIV_FILE_DAC_SEARCH	Stable
	PRIV_FILE_DAC_WRITE	Stable
	PRIV_FILE_OWNER	Stable
	PRIV_FILE_SETDAC	Stable
	PRIV_FILE_SETID	Stable
	PRIV_IPC_DAC_READ	Stable
	PRIV_IPC_DAC_WRITE	Stable
	PRIV_IPC_OWNER	Stable
	PRIV_NET_PRIVADDR	Stable
	PRIV_NET_RAWACCESS	Stable
	PRIV_PROC_CHROOT	Stable
	PRIV_PROC_CLOCK_HIGHRES	Stable
	PRIV_PROC_AUDIT	Stable
	PRIV_PROC_LOCK_MEMORY	Stable
	PRIV_PROC_OWNER	Stable
	PRIV_PROC_PRIOCNTL	Stable
	PRIV_PROC_SETID	Stable
	PRIV_PROC_TASKID	Stable
	PRIV_SYS_ACCT	Stable
	PRIV_SYS_AUDIT	Stable
	PRIV_SYS_CONFIG	Stable
	PRIV_SYS_CPU_CONFIG	Stable
	PRIV_SYS_DEVICES	Stable
	PRIV_SYS_IPC_CONFIG	Stable
	PRIV_SYS_LINKDIR	Stable
	PRIV_SYS_MOUNT	Stable
	PRIV_SYS_NET_CONFIG	Stable
	PRIV_SYS_NFS	Stable
	PRIV_SYS_RESOURCE	Stable
	PRIV_SYS_SUSER_COMPAT	Stable
	PRIV_SYS_TIME	Stable

Property	*Object*	*Commitment*
Privilege constants (basic)	`PRIV_FILE_LINK_ANY`	Stable
	`PRIV_PROC_EXEC`	Stable
	`PRIV_PROC_FORK`	Stable
	`PRIV_PROC_SESSION`	Stable

Property	*Object*	*Commitment*
Defined types	priv_t	Stable
	priv_ptype_t	Stable
	priv_set_t	Stable
	priv_impl_info_t	Evolving
	priv_info_t	Stable
	priv_info_uint_t	Stable
	priv_info_names_t	Stable
	priv_info_set_t	Stable
Privilege info constants	PRIV_INFO_SETNAMES	Stable
	PRIV_INFO_PRIVNAMES	Stable
	PRIV_INFO_FLAGS	Stable
	PRIV_INFO_BASICPRIVS	Stable
Privilege set constant	PRIV_INHERITABLE	Stable
	PRIV_PERMITTED	Stable
	PRIV_EFFECTIVE	Stable
	PRIV_LIMIT	Stable
Constant for **priv_set(3c)**	PRIV_ALLSETS	Stable
Privilege operation enum	PRIV_OFF	Stable
	PRIV_ON	Stable
	PRIV_SET	Stable
Flags for priv_set_to_str	PRIV_STR_PORT	Stable
	PRIV_STR_LIT	Stable
	PRIV_STR_SHORT	Stable
Flags for [gs]etpflags	PRIV_DEBUG	Stable
	PRIV_AWARE	Stable
Special privilege values	PRIV_NONE	Stable
	PRIV_ALL	Stable
	PRIV_MULTIPLE	Stable
size of implementation description	PRIV_IMPL_INFO_SIZE	Stable
<exec_attr.h>	EXECATTR_LPRIV_KW	Stable
	EXECATTR_IPRIV_KW	Stable
<user_attr.h>	DEF_LIMITPRIV	Stable
	DEF_DFLTPRIV	Stable
	USERATTR_LIMPRIV_KW	Stable
	USERATTR_DFLTPRIV_KW	Stable
<prof_attr.h>	PROFATTR_PRIVS_KW	Stable

Property	*Object*	*Commitment*
modctl sub commands	MODSETDEVPOLICY	Project Private
	MODGETDEVPOLICY	Project Private
	MODGETDEVTDEVPLCY	Project Private
	MODALLOCPRIV	Project Private
New Audit Events	AUE_SETPPRIV	Stable
	AUE_MODDEVPLCY	Stable
	AUE_MODALLOCPRIV	Stable
New ELF Notes	NT_PRPRIV	Stable
	NT_PRPRIVINFO	Stable
priv_getbyname(9f) constants	PRIV_ALLOC	Stable
	PRIVNAME_MAX	Stable
<sys/auxv.h>	AT_SUN_AUXFLAGS	Consolidation Private
	AF_SUN_SETUGID	Consolidation Private
<sys/syscall.h>	SYS_privsys	Private
<sys/procfs.h>	PCSPRIV	Stable
	prpriv_t	Stable
	lwpstatus_t‘pr_errpriv	Stable
<sys/strsun.h>	DB_CRED	Consolidation Private
	DB_CREDDEF	Consolidation Private

6.4 Kernel Interfaces

Property	*Object*	*Commitment*
DDI/DDK interface	`drv_priv`	Standard
	`priv_getbyname`	Evolving
	`crgetref`	Consolidation Private
	`crgetuid`	Evolving
	`crgetruid`	Evolving
	`crgetsuid`	Evolving
	`crgetgid`	Evolving
	`crgetrgid`	Evolving
	`crgetsgid`	Evolving
	`crgetgroups`	Evolving
	`crgetngroups`	Evolving
	`crsetresuid`	Consolidation Private
	`crsetresgid`	Consolidation Private
	`crsetugid`	Consolidation Private
	`crsetgroups`	Consolidation Private
	`priv_policy`	Evolving
	`priv_policy_only`	Evolving
	`priv_policy_choice`	Evolving
	`allocb_tmpl`	Evolving
	`allocb_cred`	Consolidation Private
	`allocb_cred_wait`	Consolidation Private
	`mblk_setcred`	Consolidation Private
System call signature	`setppriv`	Evolving
	`getppriv`	Evolving
	`setpflags`	Evolving
	`getpflags`	Evolving
	`getprivinfo`	Evolving
Availability	`setppriv`	Public
	`getppriv`	Public
	`setpflags`	Public
	`getpflags`	Public
	`getprivinfo`	Private
/etc/system variable	`priv_debug`	Unstable

Property	*Object*	*Commitment*
<sys/policy.h>	secpolicy_acct	Consolidation Private
	secpolicy_allow_setid	Consolidation Private
	secpolicy_audit_config	Consolidation Private
	secpolicy_audit_getattr	Consolidation Private
	secpolicy_audit_modify	Consolidation Private
	secpolicy_basic_exec	Consolidation Private
	secpolicy_basic_fork	Consolidation Private
	secpolicy_basic_link	Consolidation Private
	secpolicy_basic_proc	Consolidation Private
	secpolicy_chroot	Consolidation Private
	secpolicy_clock_highres	Consolidation Private
	secpolicy_console	Consolidation Private
	secpolicy_coreadm	Consolidation Private
	secpolicy_dispadm	Consolidation Private
	secpolicy_excl_open	Consolidation Private
	secpolicy_lock_memory	Consolidation Private
	secpolicy_modctl	Consolidation Private
	secpolicy_net_config	Consolidation Private
	secpolicy_net_privaddr	Consolidation Private
	secpolicy_net_rawaccess	Consolidation Private
	secpolicy_net	Consolidation Private
	secpolicy_newproc	Consolidation Private
	secpolicy_nfs	Consolidation Private
	secpolicy_ponline	Consolidation Private
	secpolicy_power_mgmt	Consolidation Private
	secpolicy_pset	Consolidation Private
	secpolicy_rctlsys	Consolidation Private
	secpolicy_resource	Consolidation Private
	secpolicy_rpcmod_open	Consolidation Private
	secpolicy_rsm_access	Consolidation Private
	secpolicy_setpriority	Consolidation Private
	secpolicy_settime	Consolidation Private
	secpolicy_spec_open	Consolidation Private
	secpolicy_sti	Consolidation Private
	secpolicy_sys_config	Consolidation Private
	secpolicy_sys_devices	Consolidation Private
	secpolicy_tasksys	Consolidation Private

Property	*Object*	*Commitment*
<sys/policy.h>	secpolicy_fs_config	Consolidation Private
	secpolicy_fs_linkdir	Consolidation Private
	secpolicy_fs_minfree	Consolidation Private
	secpolicy_fs_mount	Consolidation Private
	secpolicy_fs_quota	Consolidation Private
	secpolicy_ipc_access	Consolidation Private
	secpolicy_ipc_config	Consolidation Private
	secpolicy_ipc_owner	Consolidation Private
	secpolicy_pcfs_modify_bootpartition	Consolidation Private
	secpolicy_proc_access	Consolidation Private
	secpolicy_proc_excl_open	Consolidation Private
	secpolicy_proc_owner	Consolidation Private
	secpolicy_vnode_access	Consolidation Private
	secpolicy_vnode_create_gid	Consolidation Private
	secpolicy_vnode_owner	Consolidation Private
	secpolicy_vnode_remove	Consolidation Private
	secpolicy_vnode_setattr	Consolidation Private
	secpolicy_vnode_setdac	Consolidation Private
	secpolicy_vnode_setid_retain	Consolidation Private
	secpolicy_vnode_setids_setgids	Consolidation Private
	secpolicy_vnode_stky_modify	Consolidation Private

A New Manual Pages

```
System Administration Commands                              getdevpolicy(1M)

NAME
     getdevpolicy - inspect the system's device policy

SYNOPSIS
     /usr/sbin/devpolicy [device ....]

DESCRIPTION
     When run without arguments, the command outputs the device policy
     currently in effect to standard output.

     If arguments are supplied, each argument is treated as a pathname
     to a device and the device policy in effect for that specific
     device is printed preceeded by the supplied pathname.

USAGE
     The device policy adds access restrictions over and above the
     file permissions.

EXIT STATUS
     The following exit values are returned:

     0     Successful operation.

     non-zero
           An error has occurred.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface stability	
Invocation	Evolving
Output	Unstable

```
    |_____________________________|_____________________________|


SEE ALSO
     add_drv(1M), rem_drv(1M), update_drv(1M), attributes(5),
     priviliges(5), devfs(7FS)


SunOS 5.10           Last change: 20 Feb 2003                   1
```

```
User Commands                                              ppriv(1)

NAME
     ppriv - inspect or modify process privilege sets and attributes

SYNOPSIS
     /usr/bin/ppriv -e [-D|-N] [-s spec] command [arg ...]

     /usr/bin/ppriv [-v] [-D|-N] [-s spec] [ pid | core ] ...

     /usr/bin/ppriv -l [-v] [privilege ...]

DESCRIPTION
     The first invocation of the ppriv command runs the ‘‘command’’
     specified with the privilege sets and flags modified according
     the arguments on the command line.

     The second invocation examines or changes the privilege state
     of running process and core files.

     The third invocation lists the privileges defined and information
     about specified privileges.

OPTIONS
     The following options are supported:

     -e    Interpret the remainder of the arguments as a command line
           and run the command line with specified privilege attributes
           and sets.

     -D    Turn on privilege debugging for the processes or
           command supplied.
```

```
     -N    Turn off privilege debugging for the processes or
           command supplied.

     -s [AEILP][-+=]privsetspec
           Modify a process's privilege sets according to ''spec''.  A
           specification exist of the initial letters of one or more
           sets (A for all sets), followed by a ''+'', ''-'' or ''=''
           sign followed by a comma separated privilege set
           specification as accepted by priv_str_to_set(3c) to
           respectively add, remove and assign the listed privileges to
           the specified set(s).  Modifying the same set with multiple -s
           options is possible as long as there is either precisely one
           assignment to an individual set or any number of additions
           and removals. I.e., assignment and addition/removal for one
           set are mutually exclusive.

     -v    Verbose. Print privilege sets completely expanded as
           individual privileges rather than the default PRIV_STR_SHORT
           form (see priv_set_to_set(3C).
           With -l: print text explaining privilege.

     -l    List all currently defined privileges on stdout.

USAGE

     The ppriv utility examines processes and core files and
     prints or changes their privilege sets.

     It can run commands with privilege debugging on or off or with
     fewer privileges than the invoking process.

     When executing a sub process, the only sets that can be modified
     are L and I; privileges can only be removed from L and I as
     ppriv starts with P=E=I.

     ppriv can also be used to remove privileges from processes or to
     convey privileges to other processes.  In order to control a process,
     ppriv's effective set must be a super set of the controlled process'
     E, I and P; ppriv's limit set must be a super set of the target's
     limit set.  If the target's process uid's do not match, the
     {PRIV_PROC_OWNER} privilege must be asserted in ppriv's effective set.
     If the controlled processes has any uid with the value 0, more
     restriction may exist, see privileges(5).

EXAMPLES
```

```
Example 1: Obtain the process privileges of the current shell:

        $ ppriv $$
        387:    -sh
        flags = 0x0
                E: basic
                I: basic
                P: basic
                L: all

Example 2: Removing a privilege from your shell's Inheritable and
Effective set.

        $ ppriv -s EI-proc_session $$

Note: the subprocess can still inspect the parent shell but it can
no longer influence the parent as the parent has more privileges
in its Permitted set than the ppriv child process, to wit:

        $ truss -p $$
        truss: permission denied: 387

        $ ppriv $$
        387:    -sh
        flags = 0x0
                E: basic,!proc_session
                I: basic,!proc_session
                P: basic
                L: all

Example 3: Running a process with privilege debugging.

        $ ppriv -e -D cat /etc/shadow
        cat[418]: missing privilege "file_dac_read" (euid = 21782,
           syscall = 225) needed at ufs_access+0x3c
        cat: cannot open /etc/shadow

Note: the privilege debugging error messages are send to the
controlling terminal of the current process; "needed at"
address specification is an artifact of the kernel implementation
and it can be changed at any time after a software update.

The system call number can be mapped to a system call using
/etc/name_to_sysnum.
```

```
EXIT STATUS
     The following exit values are returned:

     0     Successful operation.

     non-zero
           An error has occurred.

FILES
     /proc/*
           process files

     /etc/name_to_sysnum
           system call name to number mapping

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWesu (32-bit)
	SUNWesxu (64-bit)
Interface stability	
Invocation	Evolving
Output	Unstable

```
SEE ALSO
     gcore(1), truss(1), priv_str_to_set(3c), proc(4), attributes(5),
     privileges(5)

SunOS 5.10           Last change: 17 Oct 2002                   1
```

```
System Calls                                          setppriv(2)

NAME
```

```
     setppriv, getppriv - set or get privilege set

SYNOPSIS
     #include <priv.h>

     int setppriv(priv_op_t op, priv_ptype_t which, const priv_set_t *set);

     int getppriv(priv_ptype_t which, priv_set_t *set);


DESCRIPTION

      The setppriv() system call takes three arguments; the operation
      ‘‘op’’, which can be one of PRIV_OFF, PRIV_ON or PRIV_SET; the
      parameter ‘‘which’’, the name of the privilege set to change; and a
      privilege set ‘‘set’’.

      If the first argument is PRIV_OFF the privileges in ‘‘set’’ are
      removed from the process privilege set ‘‘which’’. There is no
      restriction on removing privileges from process privileges sets,
      but the following caveats apply:

            Privileges removed from PRIV_PERMITTED are silently removed
            from PRIV_EFFECTIVE.

            If privileges are removed from PRIV_LIMIT, they
            are not removed from the other sets until one of
            exec(2) variants was called successfully.

      If the first argument is PRIV_ON the privileges in ‘‘set’’ are
      added to the process privilege set ‘‘which’’.  The following
      operations are permitted:

            Privileges in PRIV_PERMITTED can be added to PRIV_EFFECTIVE
            without restriction.

            Privileges in PRIV_PERMITTED can be added to PRIV_INHERITABLE
            without restriction.

            All operations that attempt to add privileges which are
            already present are permitted.

      If the first argument is PRIV_SET the privileges in ‘‘set’’
      replace the process privilege set ‘‘which’’ completely; PRIV_SET
```

```
      is implemented in terms of PRIV_OFF and PRIV_ON; the same
      restrictions apply.

      The getppriv() system call returns the process privilege set
      ‘‘which’’ in the set pointed to by ‘‘set’’.


RETURN VALUES
      Upon successful completion, 0 is returned. Otherwise, -1  is
      returned and errno is set to indicate the error.

ERRORS

      The setppriv() and getppriv() functions will fail if:

      EINVAL
            The value of ‘‘op’’ or ‘‘which’’ is out of range.

      EFAULT
            The ‘‘set’’ argument points to an illegal address.

      The setppriv() function will fail if:

      EPERM The application attempted to add privileges to
            PRIV_LIMIT or PRIV_PERMITTED. The application attempted
            to add privileges to PRIV_INHERITABLE or PRIV_EFFECTIVE
            which weren’t in PRIV_PERMITTED.

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

     _____________________________________________________________
    |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE        |
    |_____________________________|_____________________________|
    | MT-Level                    | MT-Safe                     |
    |_____________________________|_____________________________|
    | Stability                   | Evolving                    |
    |_____________________________|_____________________________|


NOTES
      The C library glue for these system calls is not Async-Signal-Safe.

SEE ALSO
```

```
     exec(2), attributes(5), privileges(5)

SunOS 5.10           Last change:  08 Oct 2002                    1
```

```
System Calls                                          setpflags(2)

NAME

     setpflags, getpflags - set or get process flags

SYNOPSIS
     #include <sys/types.h>
     #include <priv.h>

     int setpflags(uint_t flag, uint_t value);

     uint_t getpflags(uint_t flag);


DESCRIPTION
      The system calls setpflags() and getpflags() modify
      and obtain the current per-process flags.

      The following flags are supported:

      PRIV_AWARE
            This one bit flag takes the value of 0 (unset) or 1 (set).
            Only if this flag is set, the current process is privilege
            aware.  A process can attempt to unset this flag but
            this will fail if the observed set invariance condition
            can't be met.  Setting this flag is always successful.
            See privileges(5) for a discussion of this flag.

      PRIV_DEBUG
            This one bit flag takes the value of 0 (unset) or 1 (set).
            Only if this flag is set, the current process has privilege
            debugging enabled.  Processes can set and unset this flag
            at will.


RETURN VALUES
     The system call getpflags() returns the per-process flags.  If the
     flag argument is invalid, (uint_t)-1 is returned and errno is set
```

```
     to indicate the error.

     Upon successful completion, setpflags() returns 0. Otherwise, -1  is
     returned and errno is set to indicate the error.

ERRORS

     The functions getflags(), setpflags() will fail if:

     EINVAL
           The value of flag or the value the flag is set to
           is out of range.

     The function setpflags() will fail if:

     EPERM
           An attempt is made to unset PRIV_AWARE but the observed
           set invariance condition isn't met.



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

     _____________________________________________________________
    |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE        |
    |_____________________________|_______________________________|
    | MT-Level                    | Async-Signal-Safe             |
    |_____________________________|_______________________________|
    | Stability                   | Evolving                      |
    |_____________________________|_______________________________|


SEE ALSO

     ppriv(1), attributes(5), privileges(5)

SunOS 5.10           Last change: 06 Jan 2003                    1
```

```
Door Library Functions                          door_ucred(3DOOR)

NAME
     door_ucred - return credential information associated with
     the client
```

```
SYNOPSIS
     cc -mt [ flag ... ] file ... -ldoor  [ library ... ]
     #include <door.h>

     int door_ucred(ucred_t **info)

DESCRIPTION
     The  door_ucred() function returns credential information
     associated with the client (if any) of the current door
     invocation.

     On successful completion, door_ucred will write a pointer
     to a user credential to the location given as argument.

     The resulting user credential will include information about
     the effective user and group id, the real user and group id,
     all privilege sets, process flags and the calling pid.

     The credential information associated with the client refers
     to  the  information  from  the immediate caller; not neces-
     sarily from the first thread in a chain of door calls.

     The value returned in ‘‘info’’ must be freed using ucred_free().

RETURN VALUES
     Upon successful completion, door_ucred()  returns  0.  Other-
     wise,  door_cred() returns -1 and sets errno to indicate the
     error.

ERRORS
     The  door_ucred() function will fail if:

     EFAULT
           The address of the info argument is invalid.

     EINVAL
           There is no associated door client.

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

    _____________________________________________________________
    |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE       |
```

Architecture	all
Availability	SUNWcsu
Stability	Evolving
MT-Level	Safe

SEE ALSO
door_call(3DOOR), door_create(3DOOR), ucred(3C), attributes(5)

SunOS 5.10 Last change: 31 Aug 2002 1

Standard C Library Functions priv_names(3C)

NAME
priv_str_to_set, priv_set_to_str, priv_getbyname, priv_getbynum, priv_getsetbyname, priv_getsetbynum - privilege name functions

SYNOPSIS
```
#include <priv.h>

priv_set_t *priv_str_to_set(const char *buf, const char *sep,
                            const char **endptr);

char *priv_set_to_str(const priv_set_t *set, char sep, int flag);

int priv_getbyname(const char *privname);

const char *priv_getbynum(int privnum);

int priv_getsetbyname(const char *privsetname);

const char *priv_getsetbynum(int privsetnum);

char *priv_gettext(const char *privname);
```

DESCRIPTION
The function priv_str_to_set() maps the privilege specification

in ‘‘buf’’ to a privilege set and returns a privilege set on success or NULL on failure. If an error occurs when parsing the string a pointer to the remainder of the string is stored in the object pointed to by endptr, provided that endptr is not a null pointer. If an error occurs when allocating memory errno is set and the object pointed to by endptr is set to the null pointer, provided that endptr is not a null pointer.

The application is responsible for freeing the privilege set returned using priv_freeset(3).

A privilege specification should contain one or more privilege names, separated by characters in ‘‘sep’’ using the same algorithm as strtok(). Privileges can optionally be preceded by a dash (‘‘-’’) or an exclamation mark (‘‘!’’) in which case they are excluded from the resulting set. The special strings ‘‘none’’ for the empty set, ‘‘all’’ for the set of all privileges and ‘‘basic’’ for the set of basic privileges are also recognized. Set specification are interpreted left-to-right.

The function priv_set_to_str() converts the privilege set ‘‘set’’ to a sequence of privileges separated by ‘‘sep’’, returning the a pointer to the dynamically allocated result. The application is responsible for freeing the memory using free(3C).

In order to maintain future compatibility, the ‘‘basic’’ set of privileges is included as ‘‘basic,!missing_basic_priv1,...’’. When further currently unprivileged operations migrate to the basic privilege set, the conversion back of the result with priv_str_to_set() will include the additional basic privileges, guaranteeing that the resulting privilege set carries the same privileges. When specifying a flag argument of ‘‘PRIV_STR_LIT’’, the result will not treat basic privileges differently and the privileges present will all be literally presented in the output. A flag argument of ‘‘PRIV_STR_SHORT’’ will try to arrive at the shortest output, using the tokens ‘‘basic’’, ‘‘all’’ and negated privileges. This output is most useful for trace output.

The functions priv_getbyname() and priv_getsetbyname() map privilege names and privilege set names to numbers. Note that the numbers returned are valid for the current kernel instance only and may change at the next boot. Numbers should *not* be committed to persistent storage, only the privilege names should. Both functions return -1 on error, setting errno to

```
      EINVAL.

      The functions priv_getbynum() and priv_getsetbynum() map
      privileges numbers to names.  The strings returned point to
      shared storage that should not be modified and which is valid for
      the lifetime of the process. Both functions return NULL on error,
      setting errno to EINVAL.

      The function priv_gettext() returns a pointer to a string of
      consisting of one or more newline separated lines of text
      describing the privilege. The text is localized using
      {LC_MESSAGES}.  It is the application's responsibility to
      free the memory returned.

      The routines will pick up privileges allocated during the
      lifetime of the process using priv_getbyname(9f) by refreshing
      the internal data structures when necessary.

      Example:

            #include <priv.h>
            #include <stdio.h>

            /* list all the sets and privileges defined in the system */

            const char *name;
            int i;

            printf("Each process has the following privilege sets:\n");
            for (i = 0; (name = priv_getsetbynum(i++)) != NULL; )
                    printf("\t%s\n", name);

            printf("Each set can contain the following privileges:\n");
            for (i = 0; (name = priv_getbynum(i++)) != NULL; )
                    printf("\t%s\n", name);



RETURN VALUES
      Upon successful completion, priv_str_to_set(), priv_set_to_str()
      return a non-NULL pointer to allocated memory which should be
      freed by the application using the appropriate functions when it
      is no longer referenced; priv_getbynum() and priv_getsetbynum()
      return non-NULL pointers to constant memory which should not be
      modified or freed by the application.  If an error occurs, NULL
```

```
      is returned and errno is set to indicate the error.

      Upon successful completion, priv_getbyname() and
      priv_getsetbyname() return a non-negative integer.  If an error
      occurs, -1 is returned and errno is set to indicate the error.

      Upon successful completion, priv_gettext() returns non-NULL;
      if an error occurs or no descriptive text for the specified
      privilege can be found, NULL is returned.

ERRORS

      The priv_str_to_set() and priv_set_to_str() functions will fail if:

      ENOMEM
           The physical limits of the system are exceeded by the
           memory allocation needed to hold a privilege set.

      EAGAIN
           There is not enough memory available to allocate sufficient
           memory to hold a privilege set; but the application could
           try again later.

      All functions will fail if:

      EINVAL
           One or more of the arguments is invalid.


FILES
     /etc/security/priv_names
          File mapping privilege names to descriptive text return
          ed by priv_gettext().

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

    _____________________________________________________________
   |        ATTRIBUTE TYPE       |        ATTRIBUTE VALUE       |
   | ____________________________|______________________________|
   |  Architecture               |  all                         |
   |_____________________________|______________________________|
   |  MT-Level                   |  MT-Safe                     |
   |_____________________________|______________________________|
```

```
   |  Stability                  |  Evolving                   |
   |_____________________________|_____________________________|


SEE ALSO
      free(3c), priv_set(3), attributes(5), privileges(5),
      priv_getbyname(9f)

NOTES
     Converting the token ‘‘all’’ returns a privilege set with
     all bits set; converting the enumeration of all currently
     defined privileges to a set will return a privilege set
     with just those bits set.

SunOS 5.10           Last change: 12 Feb 2003                  1
```

```
Standard C Library Functions                          priv_set(3C)

NAME

     priv_set, priv_ineffect - convenience functions to change
     privilege sets and check whether privileges are set.

SYNOPSIS
     #include <priv.h>

     int priv_set(priv_op_t op, priv_ptype_t which, ...);

     boolean_t priv_ineffect(const char *priv);

DESCRIPTION

      The priv_set() function is a convenient wrapper for the
      setppriv(2) system call.  It takes three or more arguments; the
      operation ‘‘op’’, which can be one of PRIV_OFF, PRIV_ON or
      PRIV_SET; the parameter ‘‘which’’, the name of the privilege set
      to change; and a list of zero or more privilege names terminated
      with a null pointer.  The special pseudo set PRIV_ALLSETS can
      be used as ‘‘which’’ argument indicating that the operation
      should be applied to all privilege sets.

      The specified privileges are converted to a binary privilege
      set and setppriv(2) is called with the same ‘‘op’’ and ‘‘which’’
      arguments.  When called with PRIV_ALLSETS as argument, setppriv(2)
```

```
      is called for each set in turn, aborting on the first failed
      call.

      The priv_ineffect() function is a conventient wrapper for the
      getppriv(2) system call.  It takes a single argument, the
      parameter ‘‘priv’’, the name of the privilege to check the
      presence of in the effective set.


RETURN VALUES
      Upon successful completion, priv_set() returns 0. Otherwise, -1  is
      returned and errno is set to indicate the error.

      If priv_ineffect() is passed a valid privilege and the
      privilege is a member of the effective set, priv_ineffect
      returns B_TRUE.  In all other cases priv_ineffect() returns
      B_FALSE.  It may set errno to indicate an error condition
      in the latter case.

ERRORS

      The priv_ineffect() function will fail if:

      EINVAL
            The specified privilege is invalid.

      ENOMEM
            priv_ineffect failed to allocate sufficient memory for
            its operation.

      The priv_set() function will fail if:

      EINVAL
            The value of ‘‘op’’ or ‘‘which’’ is out of range.
            One or more of the specified privileges is invalid.

      ENOMEM
            priv_set failed to allocate sufficient memory for
            its operation.

      EPERM
            The application attempted to add privileges to PRIV_LIMIT
            or PRIV_PERMITTED. The application attempted to add
            privileges to PRIV_INHERITABLE or PRIV_EFFECTIVE which
            weren’t in PRIV_PERMITTED.
```

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

     ____________________________________________________________
    |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE       |
    |_____________________________|_____________________________|
    | Architecture                | all                         |
    |_____________________________|_____________________________|
    | MT-Level                    | MT-Safe                     |
    |_____________________________|_____________________________|
    | Stability                   | Evolving                    |
    |_____________________________|_____________________________|


SEE ALSO

     setppriv(2), priv_names(3C), attributes(5), privileges(5)

SunOS 5.10           Last change: 08 Oct 2002                  1
```

```
Standard C Library Functions                          priv_sets(3C)

NAME
     priv_allocset, priv_freeset, priv_emptyset, priv_fillset,
     priv_isemptyset, priv_isfullset, priv_isequalset, priv_issubset,
     priv_intersect, priv_union, priv_inverse, priv_addset,
     priv_copyset, priv_delset, priv_ismember - privilege set
     manipulation routines

SYNOPSIS
     #include <priv.h>

     priv_set_t *priv_allocset(void);

     void priv_freeset(priv_set_t *sp);

     void priv_emptyset(priv_set_t *sp);

     void priv_fillset(priv_set_t *sp);

     boolean_t priv_isemptyset(const priv_set_t *sp);
```

```
    boolean_t priv_isfullset(const priv_set_t *sp);

    boolean_t priv_isequalset(const priv_set_t *src, const priv_set_t *dst);

    boolean_t priv_issubset(const priv_set_t *src, const priv_set_t *dst);

    void priv_intersect(const priv_set_t *src, priv_set_t *dst);

    void priv_union(const priv_set_t *src, priv_set_t *dst);

    void priv_inverse(priv_set_t *sp);

    int priv_addset(priv_set_t *sp, const char *priv);

    void priv_copyset(const priv_set_t *src, priv_set_t *dst);

    int priv_delset(priv_set_t *sp, const char *priv);

    boolean_t priv_ismember(const priv_set_t *sp, const char *priv);


DESCRIPTION

    The arguments ‘‘sp’’, ‘‘src’’, and ‘‘dst’’ point to privilege sets.
    The argument ‘‘priv’’ points to a named privilege.

  priv_allocset(), priv_freeset()

    The function priv_allocset() allocates sufficient memory to
    contain a privilege set.  The value of the privilege set returned
    by priv_allocset() is indeterminate; the function can return NULL
    and set errno when it fails to allocate memory.  The function
    priv_freeset() frees the storage allocated by priv_allocset().

  priv_emptyset()

    The function priv_emptyset() clears all privileges from ‘‘sp’’.

  priv_fillset()

    The function priv_fillset() asserts all privileges in ‘‘sp’’, this
    includes the privileges not currently defined in the system.

  priv_isemptyset(), priv_isfullset()
```

```
     The functions priv_isemptyset() and priv_isfullset() check whether
     the argument is an empty set or a full set respectively.
     The full set is the set with all bits sets, regardless whether
     the privilege is currently defined in the system.

  priv_isequalset(), priv_issubset()

     The functions priv_isequalset() and priv_issubset() check whether
     the privilege set src is equal to dst, respectively is a subset
     of dst.

   priv_intersect(), priv_union(), priv_copyset()

     The function priv_intersect() intersects ‘‘src’’ with ‘‘dst’’ and puts
     the results in ‘‘dst’’.

     The function priv_union() takes the union of ‘‘src’’ and ‘‘dst’’
     and puts the result in ‘‘dst’’.

     The function priv_copyset() copies the set ‘‘src’’ to ‘‘dst’’.

  priv_inverse()

     The function priv_inverse() inverts the privilege set given as
     argument in place.

  priv_addset()

      The function priv_addset() adds the named privilege ‘‘priv’’ to
      ‘‘sp’’.

  priv_delset()

      The function priv_delset() removes the named privilege ‘‘priv’’
      from ‘‘sp’’.

  priv_ismember()

      The function priv_ismember() checks whether the named
      privilege ‘‘priv’’ is a member of ‘‘sp’’.


RETURN VALUES

     The function priv_allocset() returns NULL if memory allocation
```

fails, a pointer to an opaque data structure on success.

The functions priv_isemptyset(), priv_isfullset(), priv_isequalset(), priv_issubset(), priv_ismember() return B_TRUE if the functions are successful, B_FALSE if they are not.

The functions priv_delset() and priv_addset() return 0 on success, -1 on failure.

ERRORS

The priv_allocset() function will will fail if:

ENOMEM
The physical limits of the system are exceeded by the memory allocation needed to hold a privilege set.

EAGAIN
There is not enough memory available to allocate sufficient memory to hold a privilege set; but the application could try again later.

The functions priv_delset() and priv_addset() will fail if:

EINVAL
The privilege argument is not a valid privilege name.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	all
MT-Level	MT-Safe
Stability	Evolving

SEE ALSO

```
      malloc(3C), privileges(5)

NOTES
      The functions which compare sets operate on all bits of the
      set, regardless of whether the specific privileges are
      currently defined in the system.

SunOS 5.10           Last change: 12 Feb 2003                   1
```

```
Standard C Library Functions                          ucred(3C)

NAME
     ucred_geteuid, ucred_getruid, ucred_getsuid, ucred_getegid,
     ucred_getrgid, ucred_getsgid, ucred_getgroups, ucred_getpid,
     ucred_free, ucred_getprivset, ucred_getpflags

SYNOPSIS
     #include <ucred.h>

     ucred_t *ucred_get(pid_t pid);

     void ucred_free(ucred_t *uc);

     uid_t ucred_geteuid(const ucred_t *uc);

     uid_t ucred_getruid(const ucred_t *uc);

     uid_t ucred_getsuid(const ucred_t *uc);

     gid_t ucred_getegid(const ucred_t *uc);

     gid_t ucred_getrgid(const ucred_t *uc);

     gid_t ucred_getsgid(const ucred_t *uc);

     int ucred_getgroups(const ucred_t *uc, const gid_t **groups);

     const priv_set_t *ucred_getprivset(const ucred_t *uc, const char *set);

     pid_t ucred_getpid(const ucred_t *uc);

     uint_t ucred_getpflags(const ucred_t *uc, uint_t flags);
```

```
DESCRIPTION
      This manual page describes functions that return or act on
      a user credential, ucred_t.  User credentials are returned by
      various functions and describe the credentials of a process;
      information about the process can then be obtained by calling
      the access functions.  Access functions can fail if the underlying
      mechanism did not return sufficient information.

      The function ucred_get() returns the user credential of
      the specified pid or NULL if none can be obtained.  A pid
      value of P_MYID will return information about the calling process.
      The return value is dynamically allocated and must be freed using
      ucred_free().

      The functions ucred_geteuid, ucred_getruid, ucred_getsuid,
      ucred_getegid, ucred_getrgid, ucred_getsgid return the
      effective uid, real uid, saved uid, effective gid, real gid,
      saved gid, respectively, or -1 if the user credential does not
      contain sufficient information.

      The function ucred_getgroups stores a pointer to the group list
      in the gid_t * pointed to by the second argument and returns the
      number of groups in the list; it returns -1 if the information is
      not available.  The returned group list is valid until ucred_free()
      is called on the user credential given as argument.

      The function ucred_getpid() returns the process id of the process
      or -1 if the process id is not available.

      The function ucred_getprivset() returns the specified privilege
      set specified as second argument, or NULL if the requested
      information is not available or the priv_ptype_t argument is
      invalid.  The returned privilege set is valid until ucred_free()
      is called on the user credential given as argument.

      The function ucred_getpflags() returns the value of the specified
      process flags from the ucred structure or (uint_t)-1 if none
      was present.

      The function ucred_free() frees the memory allocated for the
      user credential given as argument.

ERRORS

     The ucred_get() function will will fail if:
```

```
     ENOMEM
           The physical limits of the system are exceeded by the
           memory allocation needed to hold a user credential.

     EAGAIN
           There is not enough memory available to allocate sufficient
           memory to hold a user credential; but the application could
           try again later.

     EACCES
           The caller does not have sufficient privileges to examine the
           target process.

     ESRCH
           The target process does not exist or /proc is not mounted.

     ENFILE
     EMFILE
           The calling process cannot open any more files.

     The function ucred_getprivset() will fail if:

     EINVAL
           The privilege set argument given to ucred_getprivset() is
           invalid.


     The functions ucred_geteuid(), ucred_getruid(), ucred_getsuid(),
     ucred_getegid(), ucred_getrgid(), ucred_getsgid(), ucred_getgroups(),
     ucred_getpflags() and ucred_getprivset() will fail if:

     EINVAL
           The requested user credential attribute is not available
           in the user credential given as argument.

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

    _____________________________________________________________
   |        ATTRIBUTE TYPE       |        ATTRIBUTE VALUE        |
   |_____________________________|_______________________________|
   |  Architecture               |  all                          |
   |_____________________________|_______________________________|
```

MT-Level	MT-Safe
Stability	Evolving

```
SEE ALSO
      door_ucred(3door), getppriv(2), getpflags(2), priv_set(3),
      attributes(5), privileges(5)

SunOS 5.10           Last change:  08 Oct 2002                    1
```

```
NOTE: PSARC MATERIAL ONLY; PRIVATE FILE FORMAT NOT AN ACTUAL MANUAL PAGE

File Formats                                        device_policy(4)

NAME
     device_policy - file that defines additional open(2) restrictions
     for devices.

SYNOPSIS
     /etc/security/device_policy

DESCRIPTION
     The device policy file, /etc/security/device_policy, is a
     system file that contains the definitions of additional
     protection for device files.

     The devfsadm(1M) utility loads the device_policy file during the
     booting sequence and when a driver is added or updated.

     The format and layout of this file is unstable; it can change
     with any software update.  Entries should be added through
     add_drv(1m) and update_drv(1m) and removed using rem_drv(1m).

     The file contains device policy specifications, one per line.
     Each specification consists of a device specification and one
     or more tokens separated by white space.  Additionally, lines
     starting with ‘‘#’’ (pound sign) are treated as comments and
     ‘‘\’’ (backslash) escapes newlines outside comments.

     A device specification is either "*", the default entry, or
     major:minor-expression, where ‘‘major’’ is the name of a driver
     as specified in /etc/name_to_major and where ‘‘minor-expression’’
     is has one of the following formats:
```

```
        <string>                a string matching exactly one minor node
        <string with one *>     a string with exactly one ‘‘*’’, matching
                                any character.

    The matching is performed in the following order: entries with
    minor number specified or without a wild card are matched first
    by the kernel.  Entries with wild card are matched in order, longest
    pattern first.

    The following tokens are currently recognized:

        write_priv_set=<privilege-set>

                Privilege set required when the device is opened for
                writing.

        read_priv_set=<privilege-set>

                Privilege set required when the device is opened for
                reading only.

     The first entry in the device_policy file must be the default
     entry, ‘‘*’’, missing tokens for all following devices take their
     value from the default entry.

     The file should be modified using the add_drv(1m), update_drv(1m)
     and rem_drv(1m) commands.



EXAMPLES
     Example 1: A Sample device_policy File

     The following is a sample device_policy file:

     #
     # The default entry: no additional checks needed.
     #
     *  \
                read_priv_set=none \
                write_priv_set=none

    #
    # Write access to the memory devices allows a process to gain
    # all privileges.  Therefor we require a process to have all
```

```
    # privileges first.
    #
    mm:*mem     write_priv_set=all
```

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	all
Availability	SUNWcsr
Interface Stability	Unstable

```
SEE ALSO
     add_drv(1m), devfsadm(1M), init(1M), rem_drv(1m), update_drv(1m),
     open(2), priv_str_to_set(3C), extra_privs(4)

SunOS 5.10           Last change: 19 Oct 2002                    1

NOTE: PSARC MATERIAL ONLY; PRIVATE FILE FORMAT NOT AN ACTUAL MANUAL PAGE
```

```
NOTE: PSARC MATERIAL ONLY; PRIVATE FILE FORMAT NOT AN ACTUAL MANUAL PAGE

File Formats                                      extra_privs(4)

NAME
     extra_privs - file that defines additional system wide privileges

SYNOPSIS
     /etc/security/extra_privs

DESCRIPTION
     The extra privileges file, /etc/security/extra_privs, is a
     system file that contains the definitions of additional
     per driver privileges, e.g., when those need to be defined
     prior to loading the device policy.

     The privileges are specified using "drivername:privilege",
     where "privilege" is the actual privilege allocated; it is
     only loaded when "drivername" is known to the system as driver.

     The file contains a privilege, one per line.
     Lines starting with ‘‘#’’ (pound sign) are treated as comments.
```

```
     The commands add_drv(1m), update_drv(1m) and rem_drv(1m) are
     used to maintain this file.

EXAMPLES

    #
    # We define this privilege so we can restrict all uses of
    # tcp/ip sockets using device_policy(4).
    #
    ip:needed_for_ip


SEE ALSO
     add_drv(1M), devfsadm(1M), init(1M), rem_drv(1m), update_drv(1m),
     device_policy(4)

SunOS 5.10           Last change: 01 Oct 2002                    1

NOTE: PSARC MATERIAL ONLY; PRIVATE FILE FORMAT NOT AN ACTUAL MANUAL PAGE
```

```
Standards, Environments, and Macros                  privileges(5)

NAME
    privileges - the Solaris privilege model

DESCRIPTION

    The Solaris privilege mechanisms provides fine grained control
    over the privileges of a process.  The possession of
    a certain privilege allows a process to perform a specific
    set of restricted operations.
    The change to a primarily privilege based security model in
    the Solaris operating environment gives developers an
    opportunity to restrict processes to those privileged
    operations actually needed instead of all (Super-user)
    or no privileges (non-0 uids).  Additionally, a set of
    previously unrestricted operations now requires a privilege;
    these privileges are dubbed the "basic" privileges and
    are by default given to all processes.

    Taken together, all defined privileges with the exception of the
    "basic" privileges compose the set of privileges that are
    traditionally associated with the root user.  The "basic"
    privileges are "privileges" unprivileged processes were accustomed
```

```
to having.

The defined privileges are:

PRIV_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_CHOWN_SELF

        Allows a process to give away its files; a process with
        this privilege will run as if {_POSIX_CHOWN_RESTRICTED} is
        not in effect.

PRIV_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

        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

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

PRIV_FILE_DAC_WRITE

        Allows a process to write a file or directory whose
        permission bits or ACL do not allow the process write
        permission.  In order to write files owned by uid 0 in the
        absence of an effective uid of 0 ALL privileges are
        required.

PRIV_FILE_LINK_ANY

        Allows a process to create hardlinks to files owned by a
```

uid different from the process' effective uid.

PRIV_FILE_OWNER

Allows a process which is not the owner of a file to modify that file's access and modification times. Allows a process which is not the owner of a directory to modify that directory's access and modification times. 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

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 except for the set-uid and set-gid bits.

PRIV_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-group-ID bit on a file or directory whose group is not the process' effective group or one of the process' supplemental groups. Allows a process to set the set-user-ID bit on a file with different ownership in the presence of PRIV_FILE_SETDAC. Additional restrictions apply when creating or modifying a set-uid 0 file.

PRIV_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. Allows a process to read remote shared memory whose permission bits do not allow the process read permission.

PRIV_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.  Allows a
        process to read remote shared memory whose permission bits
        do not allow the process write permission.  Additional
        restrictions apply if the owner of the object has uid 0 and
        the effective uid of the current process is not 0.

PRIV_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.  Additional restrictions apply if the owner of the
        object has uid 0 and the effective uid of the current
        process is not 0.

PRIV_NET_PRIVADDR

        Allows a process to bind to a privileged port number. The
        privilege port numbers are 1-1023 (the traditional UNIX
        privileged ports) as well as those ports marked as
        "udp/tcp_extra_priv_ports" with the exception of the ports
        reserved for use by NFS.

PRIV_NET_RAWACCESS

        Allows a process to have direct access to the network
        layer.

PRIV_PROC_CHROOT

        Allows a process to change its root directory.

PRIV_PROC_CLOCK_HIGHRES

        Allows a process to use high resolution timers.

PRIV_PROC_AUDIT

        Allows a process to generate audit records.  Allows a
        process to get its own audit pre-selection information.

PRIV_PROC_EXEC

        Allows a process to call execve().
```

```
PRIV_PROC_FORK

        Allows a process to call fork()/fork1()/vfork()

PRIV_PROC_LOCK_MEMORY

        Allows a process to lock pages in physical memory.

PRIV_PROC_OWNER

        Allows a process to send signals to other processes,
        inspect and modify process state to other processes
        regardless of ownership.  When modifying another process,
        additional restrictions apply:  the effective privilege set
        of the attaching process must be a superset of the target
        process' effective, permitted and inheritable sets; the
        limit set must be a superset of the target's limit set; if
        the target process has any uid set to 0 all privilege must
        be asserted unless the effective uid is 0.  Allows a
        process to bind arbitrary processes to CPUs.

PRIV_PROC_PRIOCNTL

        Allows a process to elevate its priority above its current
        level.  Allows a process to change its scheduling class to
        any scheduling class, including the RT class.

PRIV_PROC_SESSION

        Allows a process to send signals or trace processes outside
        its session.

PRIV_PROC_SETID

        Allows a process to set its uids at will.  Assuming uid 0
        requires all privileges to be asserted.

PRIV_PROC_TASKID

        Allows a process to assign a new task ID to the calling
        process.

PRIV_SYS_ACCT
```

```
        Allows a process to enable and disable and manage
        accounting through acct(2), getacct(2), putacct(2) and
        wracct(2).

PRIV_SYS_AUDIT

        Allows a process to start the (kernel) audit daemon.
        Allows a process to view and set audit state (audit user
        ID, audit terminal ID, audit sessions ID, audit
        pre-selection 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_CONFIG

        Allows a process to perform various system configuration
        tasks.  Allows filesystem specific administrative
        procedures, such as filesystem configuration ioctls, quota
        calls, creation/deletion of snapshots, manipulating the
        PCFS bootsector.

PRIV_SYS_CPU_CONFIG

        Allows a process to create and delete processor sets,
        assign CPUs to processor sets and override the
        PSET_NOESCAPE property.  Allows a process to change the
        operational status of CPUs in the system using
        p_online(2).

PRIV_SYS_DEVICES

        Allows a process to create device special files.  Allows a
        process to successfully call a kernel module that calls the
        kernel drv_priv(9F) function to check for allowed access.
        Allows a process to open the real console device directly.
        Allows a process to open devices that have been exclusively
        opened.

PRIV_SYS_IPC_CONFIG

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

PRIV_SYS_LINKDIR
```

```
        Allows a process to unlink and link directories.

PRIV_SYS_MOUNT

        Allows a process to mount and unmount filesystems which
        would otherwise be restricted (i.e., most filesystems
        except namefs) Allows a process to add and remove swap
        devices.

PRIV_SYS_NET_CONFIG

        Allows a process to configure a system's network interfaces
        and routes.  Allows a process to configure network
        parameters using ndd.  Allows a process access to otherwise
        restricted information using ndd.  Allows a process to push
        the rpcmod STREAMs module.  Allows a process to pop
        anchored STREAMs modules.  Allows a process to
        INSERT/REMOVE STREAMs modules on locations other than the
        top of the module stack.  Allows a process to configure
        IPsec.

PRIV_SYS_NFS

        Allows a process to perform Sun private NFS specific system
        calls.  Allows a process to bind to ports reserved by NFS:
        ports 2049 (nfs) and port 4045 (lockd).

PRIV_SYS_RESOURCE

        Allows a process to modify the resource limits with by
        setrlimit(2) and setrctl(2) without restriction.  Allows a
        process to exceed the per-user maximum number of
        processes.  Allows a process to configure filesystem
        quotas.  Allows a process to extend or create files on a
        filesystem that has less than minfree space in reserve.

PRIV_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 Solaris proper.
```

```
PRIV_SYS_TIME

        Allows a process to manipulate system time using any of the
        appropriate system calls: stime, adjtime, ntp_adjtime and
        the IA specific RTC calls.

Of the above privileges, the privileges PRIV_FILE_LINK_ANY,
PRIV_PROC_SESSION, PRIV_PROC_FORK and PRIV_PROC_EXEC are
so called "basic" privileges; privileges that used to be
always available to unprivileged processes.  By default,
processes still have those privileges.

The privileges PRIV_SYS_RESOURCE, PRIV_PROC_SETID and
PRIV_PROC_AUDIT must be present in the Limit set of a
process in order for set-uid root execs to be successful,
i.e., get an effective uid of 0 and additional privileges.

The Solaris privilege implementation extends the process
credential with four privilege sets:

    I, the inheritable set, the privileges inherited on exec.
    P, the permitted set, the maximum set of privileges for
    the process,
    E, the effective set, the privileges currently in effect.
    L, the limit set, the upper bound of the privileges a process
    and its offspring can obtain.  Changes to L take effect on
    the next exec.

The sets I, P and E are typically identical to the "basic" set
of privileges for unprivileged processes.  The limit set is
typically the full set of privileges.

Additionally, each process has a Privilege Awareness State (PAS)
which can take the value PA (Privilege Aware) and NPA (Not-PA).
PAS is a transitional mechanism which allows a choice between
full compatibility with the old Super-user model and completely
ignoring the effective uid.

In order to facilitate the discussion, we introduce the notion of
"observed effective set" (oE) and "observed permitted set" (oP) and
the implementation sets iE and iP.

When a process is Privilege Aware, oE and oP are invariant under
uid changes; when a process is not privilege aware, oE and oP
are observed as follows:
```

```
        oE = euid == 0 ? L : iE
        oP = (euid == 0 || ruid == 0 || suid == 0) ? L : iP
```

I.e., when a non-privilege aware process has an effective uid of 0, it can wield the privileges contained in its limit set, the upper bound of its privileges. If a non-privilege aware process has any of the uids 0, it will appear to be able to potentially wield all privileges in L.

A process becomes privilege aware either by manipulating the effective, permitted or limit privilege sets through setppriv(2) or, alternatively, by using setpflags(2). In all cases, oE and oP are invariant in the process of becoming privilege aware; in the process of becoming privilege aware, the following assignments take place:

```
            iE = oE
            iP = oP
```

It is possible for a process to return to the non-privilege aware state; this operation is only permitted if the following conditions are met:

    If any of the uids is equal to 0, P must be equal to L
    If the effective uid is equal to 0, E must be equal to L

When a process gives up privilege awareness, the following assignments take place:

```
            if (euid == 0) iE = L & I
            if (any uid == 0) iP = L & I
```

I.e., the privileges when not being having a uid of 0 will be the inheritable set of the process restricted by the limit set.

A process can attempt to become NPA using setpflags(2); the kernel will always attempt this on exec(2).

Only privileges in the process' (observed) effective privilege set allow the process to perform restricted operations. A process may use any of the privilege manipulation functions to add or remove privileges from the privilege sets. Privileges can be removed always; only privileges found in the permitted set can be added to the effective and inheritable set. The limit set cannot grow.

```
    The inheritable set can be larger than the permitted set.

    When a process performs an exec(2), the kernel will first try
    to relinquish privilege awareness and then the following privilege
    set modifications take place:

                E' = P' = I' = L & I
                L is unchanged

    If a process has not manipulated its privileges, then the privilege
    sets effectively remain the same as E, P and I are already identical.

    It is at exec(2) time that the limit set is enforced.


PRIVILEGE ESCALATION

    In certain circumstances a single privilege could lead to a process
    gaining one or more additional privileges.  To prevent such
    an escalation of privileges, which we defined as ''the process whereby
    a subject can obtain one or more privileges by performing an action
    that does not require those privileges if that action is not
    specifically allowed by the security policy'', the security policy
    will require those additional privileges.

    Typical examples are those mechanisms that allow modification of
    system resources through ''raw'' interfaces; e.g., changing kernel
    data structures through /dev/kmem or changing files through
    /dev/dsk/*, and  also include controlling processes with
    potentially more privileges than the controlling process.  A
    special case of this is manipulating or creating objects owned by
    uid 0 or trying to obtain uid 0 using setuid(2).  The special
    treatment of uid 0 is warranted because the uid 0 owns all system
    configuration files and ordinary file protection mechanisms allow
    processes with uid 0 to modify the system configuration to the
    extend that all privileges can be gained by processes running with
    an effective uid of 0.

    In such cases, the security policy will require additional
    privileges, upto the full set of privileges.  Future Solaris
    releases may relax such restrictions when additional mechanisms for
    protection of system files come available.

    The use of uid 0 processes should be limited as much as possible;
    they should be replaced with programs running under a different uid
```

```
    but with exactly the privileges they need.

    Daemons that never need to exec subprocesses should remove the
    PRIV_PROC_EXEC privilege from their permitted and limit sets.

PRIVILEGE DEBUGGING

    When a system call fails with a permission error, it is not always
    immediately obvious what the problem.  To aid in debugging this
    problem we've introduced a tool called ''privilege debugging'';
    when privilege debugging is enabled for a process, the kernel
    will flag missing privileges on the controlling terminal of the
    process.  Additionally, the administrator can enable system wide
    privilege debugging by setting the system(4) variable ''priv_debug''
    using:

        set priv_debug = 1

    This variable can be changed after the kernel has booted using
    mdb(1).

    The command truss(1) will report missing privileges for failed
    system calls.


SEE ALSO
    add_drv(1m), ifconfig(1m), lockd(1m), mdb(1), nfsd(1m), ppriv(1),
    rem_drv(1m), update_drv(1m), Intro(2), access(2), acct(2), acl(2),
    adjtime(2), audit(2), auditon(2), auditsvc(2), chmod(2), chown(2),
    chroot(2), creat(2), exec(2), fork(2), fpathconf(2), getacct(2),
    getauid(2), getgroups(2), getrlimit(2), gettimeofday(2), kill(2),
    link(2), memcntl(2), mknod(2), mount(2), msgctl(2), nice(2),
    ntp_adjtime(2), open(2), p_online(2), priocntl(2), priocntlset(2),
    processor_bind(2), pset_bind(2), pset_create(2), readlink(2),
    resolvepath(2), rmdir(2), semctl(2), setpflags(2), setppriv(2),
    setrctl(2), setregid(2), setreuid(2), settaskid(2), setuid(2),
    shmctl(2), shmget(2), shmop(2), sigsend(2), stat(2), statvfs(2),
    stime(2), swapctl(2), sysinfo(2), uadmin(2), ulimit(2), umount(2),
    unlink(2), utime(2), utimes(2), bind(3socket), bind(3xnet),
    door_ucred(3door), priv_names(3c), priv_set(3c), priv_sets(3c),
    socket(3socket), t_bind(3nsl), timer_create(3rt), ucred(3C),
    exec_attr(4), proc(4), system(4), user_attr(4), ddi_cred(9f),
    drv_priv(9f), priv_getbyname(9f), priv_policy(9f)
```

```
SunOS 5.10          Last change:  11 Dec 2002                    1
```

```
SunOS 5.10        Kernel Functions for Drivers       allocb_tmpl(9F)

NAME
     allocb_tmpl - allocate a message block using a template

SYNOPSIS
     #include <sys/stream.h>

     mblk_t *allocb_tmpl(size_t size, const mblk_t *tmpl);

INTERFACE LEVEL
     Solaris DDI specific (Solaris DDI)

DESCRIPTION
     allocb_tmpl() tries to allocate a  STREAMS message block using
     allocb(9f).  If the allocation is successful, the data block
     structure (dblk_t, see datab(9S)) db_type field as well as some
     implementation private data are copied from the dblk_t associated
     with tmpl.

     allocb_tmpl() should be used when a new STREAMs message block is
     allocated which is then used to contain data derived from another
     STREAMs message block.  The original message is used as tmpl
     argument.

PARAMETERS
     size  The number of bytes in the message block.

     tmpl  The template message block.

RETURN VALUES
     Upon success, allocb_tmpl() returns a pointer to the allocated
     message block of the same type as tmpl. On failure, allocb_tmpl()
     returns a NULL pointer.

CONTEXT
     allocb_tmpl() can be called from user or interrupt context.

SEE ALSO
     allocb(9F), datab(9S), msgb(9S)

     Writing Device Drivers
```

```
     STREAMS Programming Guide

modified 17 februari 2003                                allocb_tmpl(9F)
```

```
Kernel Functions for Drivers                              priv_policy(9F)

NAME
     priv_policy, priv_policy_only, priv_policy_choice - check,
     report and audit privileges

SYNOPSIS
     #include <sys/ddi.h>
     #include <sys/priv.h>
     #include <sys/sunddi.h>

     int priv_policy(const cred_t *cr, int priv, int err, const char *msg);

     boolean_t priv_policy_only(const cred_t *cr, int priv);

     boolean_t priv_policy_choice(const cred_t *cr, int priv);

INTERFACE LEVEL
     Solaris DDI specific (Solaris DDI).

PARAMETERS
     cr
           The credential to check.

     priv
           The integer value of the privilege to test.

     err
           The error code to return.

DESCRIPTION
     These functions aid in privilege checking and privilege debugging.

     priv_policy(), priv_policy_only() and priv_policy_choice() all
     check whether ‘‘priv’’ is asserted in the effective set of the
     credential.  The special value ‘‘PRIV_ALL’’ tests for all
     privileges.

     priv_policy() updates the ASU accounting flag and records the
     privilege used on success in the audit trail if the required
```

```
     privilege wasn't a basic privilege.  priv_policy audits successful
     and unsuccesful use of privileges.

     priv_policy_only() returns a boolean indicating whether a privilege
     is asserted and has no further side effects.

     priv_policy_choice() behaves like priv_policy_only() but records
     the successfully used non-basic privileges in the audit trail.

RETURN VALUES

      On success, priv_policy() return 0.  On failure it returns
      its parameter ''err''.

      On success, priv_policy_choice() and priv_policy_only()
      return B_TRUE, on failure both return B_FALSE.

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

    _____________________________________________________________
   |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE        |
   |_____________________________|_____________________________|
   | Interface Stability         | Evolving                    |
   |_____________________________|_____________________________|

CONTEXT
     The functions can be called from user or kernel context.

SEE ALSO
     acct(3head), attributes(5), privileges(5)

NOTES

SunOS 5.10          Last change: 10 Dec 2002                  1
```

```
Kernel Functions for Drivers                          ddi_cred(9F)

NAME
     crgetuid, crgetruid, crgetsuid, crgetgid, crgetrgid, crgetsgid,
     crgetgroups, crgetngroups - access parts of the cred_t structure
```

```
SYNOPSIS
     #include <sys/ddi.h>
     #include <sys/cred.h>

     uid_t crgetuid(const cred_t *cr);

     uid_t crgetruid(const cred_t *cr);

     uid_t crgetsuid(const cred_t *cr);

     gid_t crgetgid(const cred_t *cr);

     gid_t crgetrgid(const cred_t *cr);

     gid_t crgetsgid(const cred_t *cr);

     const gid_t *crgetgroups(const cred_t *cr);

     int crgetngroups(const cred_t *cr);


INTERFACE LEVEL
     Solaris DDI specific (Solaris DDI).

PARAMETERS

     cr          Pointer to the user credential structure.
     uid, ruid, euid, suid
                 New user id, real, effective and saved user id.
     gid, rgid, egid, sgid
                 New group id, real, effective and saved group id.

     ngroups   Number of groups in the group array.

     gids      Pointer to array of new groups.


DESCRIPTION

     The user credential is a shared, read-only, ref-counted
     data structure. It's actual size and layout can change;
     the function in this manual page allow the programmer to
     retrieve fields from the structure and to initialize
     newly allocated credential structures.
```

```
     crgetuid(), crgetruid(), crgetsuid() return the effective,
     real and saved user id from the user credential pointed to
     by cr, respectively.

     crgetgid(), crgetrgid(), crgetsgid() return the effective,
     real and saved group id from the user credential pointed to
     by cr, respectively.

     crgetgroups() returns the group list of the user credential
     pointed to by cr.

     crgetngroups() returns the number of groups in the user credential
     pointed to by cr

RETURN VALUES

     The crget*() function return the requested information.

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

    _____________________________________________________________
    |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE       |
    |_____________________________|_____________________________|
    | Architecture                | all                         |
    |_____________________________|_____________________________|
    | Interface Stability         | Evolving                    |
    |_____________________________|_____________________________|


CONTEXT
     These functions can be called from user and kernel context.

SEE ALSO
     attributes(5), privileges(5), drv_priv(9f)

     Writing Device Drivers

NOTES

SunOS 5.10          Last change: 08 Oct 2002                  1
```

```
Kernel Functions for Drivers                  priv_getbyname(9F)

NAME
     priv_getbyname - map a privilege name to a number

SYNOPSIS
     #include <sys/priv.h>

     int priv_getbyname(const char *priv, uint_t flags);

INTERFACE LEVEL
     Solaris DDI specific (Solaris DDI).

PARAMETERS
     priv
           The name of the privilege.

     flags
           Flags, must be zero or PRIV_ALLOC.

DESCRIPTION
     This routine maps a Solaris privilege name to a privilege
     number for use with the priv_*() kernel interfaces.

     If PRIV_ALLOC is passed as a flag parameter, an attempt
     is made to allocate a privilege if it is not yet defined.
     The newly allocated privilege number is returned.

     Privilege names can be specified with an optional ‘‘priv_’’
     prefix which is stripped.

     Privilege names are case insensitive but allocated privileges
     preserve case.

     Allocated privileges can be at most {PRIVNAME_MAX} characters long
     and can only contain alphanumeric characters and the underscore
     character.

     Privileges allocated through this interface should be made
     vendor unique, e.g., by prefixing them with the company’s stock
     symbol.

RETURN VALUES
```

```
     This routine returns the privilege number which is >= 0 if it
     succeeds; it returns a negative error number if an error occurs.


ERRORS

     EINVAL
          The flags parameter is invalid.
          The specified privilege does not exist.
          A request was made to allocate a privilege with invalid
          characters.

     ENOMEM
          There is no room to allocate another privilege.

     ENAMETOOLONG
          An attempt was made to allocate a privilege that was
          longer than {PRIVNAME_MAX} characters.


CONTEXT
     These functions can not be called from interrupt context.

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

     _____________________________________________________________
    |       ATTRIBUTE TYPE        |       ATTRIBUTE VALUE        |
    |_____________________________|_______________________________|
    | Architecture                | all                           |
    |_____________________________|_______________________________|
    | Interface Stability         | Evolving                      |
    |_____________________________|_______________________________|

SEE ALSO
     attributes(5), privileges(5)

     Writing Device Drivers


SunOS 5.10          Last change: 08 Oct 2002                  1
```

B Changed Manual Pages

```
--- mdb.1       Fri Jan 24 12:45:12 2003
+++ mdb.1.new   Fri Jan 24 12:47:51 2003
@@ -2077,8 +2077,8 @@
            these files are operating system crash dump files. If no
            object or core operand is specified, but the -k option is
            specified, mdb defaults to an object file of /dev/ksyms and
-           a core file of /dev/kmem. Access to /dev/kmem is restricted
-           to group sys.
+           a core file of /dev/kmem. Read access to /dev/kmem is
+           restricted to group sys.  Write access requires ALL privileges.

      -L path
            Sets default path for locating debugger modules. Modules
```

```
--- pfexec.1    Sun Oct  6 17:46:23 2002
+++ pfexec.1.new        Fri Nov 15 15:53:09 2002
@@ -6,6 +6,8 @@
 SYNOPSIS
      /usr/bin/pfexec  command

+     /usr/bin/pfexec -P privspec command [arg ...]
+
      /usr/bin/pfsh [ options ] [ argument ...]

      /usr/bin/pfcsh [ options ] [ argument ...]
@@ -24,6 +26,22 @@
      than one profile, the profile shell uses the first matching
      entry.

+     The second form, pfexec -P privspec, allows a user to obtain the
+     additional privileges awarded to his profiles in prof_attr(4).
+     The privileges specification on the commands line is parsed using
+     priv_str_to_set(3C) and the resulting privileges are intersected
+     with the union of the privileges specified using the "privs"
+     keyword in prof_attr(4) for all the user's profiles and added to
+     the inheritable set prior to executing the command.
+
+     Example 1:
+
+            pfexec -P all chown user file
+
+     This command runs "chown user file" with all privileges assigned
+     to the current user, not necessarily all privileges.
```

```
+
+
 USAGE
      pfexec is used to execute commands with predefined process attri-
      butes,  such as specific user or group IDs.
@@ -49,7 +67,7 @@

 SEE ALSO
      csh(1), ksh(1), profiles(1), sh(1), exec_attr(4), prof_attr(4),
-     user_attr(4), attributes(5)
+     user_attr(4), attributes(5), privileges(5)

 modified 10 Sep 1999                                      pfexec(1)
```

```
--- truss.1     Thu Oct 31 12:37:57 2002
+++ truss.1.new Thu Oct 31 12:41:19 2002
@@ -18,7 +18,10 @@
      displayed symbolically when possible using defines from relevant
      system headers; for any path name pointer argument, the pointed-
      to string is displayed. Error returns are reported using the
-     error code names described in intro(3).
+     error code names described in intro(3).  If in the case of an
+     error the kernel reports a missing privilege, a privilege name
+     as described in privileges(5) is reported in square brackets after
+     the error code name.

      Optionally (see the -u option), truss will also produce an
      entry/exit trace of user-level function calls executed by the
```

```
--- add_drv.1m  Thu Oct  3 15:41:30 2002
+++ add_drv.1m.new      Fri Jan 24 12:35:07 2003
@@ -5,7 +5,8 @@

 SYNOPSIS
      add_drv [-b basedir] [-c class_name] [ -i 'identify_name...'] [
-     -m 'permission','...'] [-n] [-f] [-v] device_driver
+     -m 'permission','...'] [-p 'policy'] [-P privilege] [-n] [-f]
+     [-v] device_driver

 DESCRIPTION
      The add_drv command is used to inform the system about newly
@@ -53,6 +54,34 @@
            Specify the file system permissions for device nodes
```

```
            created by the system on behalf of device_driver.

+     -p 'policy'
+           Specify an additional device security policy.  The device
+           security policy constists of several whitespace separated
+           tokens:
+
+                {<minorspec>  {token=value}+}+
+
+           The minor spec is a simple wildcard pattern for a minor device;
+           a single '*' matches all minor devices; only one '*' is
+           allowed in the pattern.  The patterns will be matched in the
+           following order:
+
+                       entries without a wildcard
+                       entries with wildcards, longest wildcard first
+
+           Currently the following tokens are defined "read_priv_set"
+           and "write_priv_set", the privileges that need to be
+           asserted in the effective set of the calling process when
+           opening a device for reading or writing, respectively.
+           See privileges(5).
+
+           A missing minor spec is interpreted as a ''*''.
+
+     -P 'privilege'
+           Specify additional, comma separated, privileges used by this
+           driver; the specific privileges can also be used in the device's
+           policy.
+
      -n    Do not try to load and attach device_driver, just modify
            the system configuration files for the device_driver.

@@ -74,12 +103,16 @@
      has already been copied to /usr/kernel/drv.

      example# add_drv -m '* 0666 bin bin','a 0644 root sys' \
+             -p 'a write_priv_set=sys_config  * write_priv_set=none' \
              -i 'SUNW,alias' SUNW,example

      Every minor node created by the system for the SUNW,example
      driver will have the permission 0666, and be owned by user bin in
      the group bin, except for the minor device a, which will be owned
-     by root, group sys, and have a permission of 0644.
+     by root, group sys, and have a permission of 0644.  The device
```

```
+     policy specified requires no additional privileges to open all
+     minor nodes, except minor device a, which requires the "sys_config"
+     privilege when opening the device for writing.

      Example 2: Adding Driver to the Client /export/root/sun1

@@ -189,6 +222,12 @@
      /etc/name_to_major
            major number binding

+     /etc/security/device_policy
+           the device policy
+
+     /etc/security/extra_privs
+           device privileges
+
 ATTRIBUTES
      See attributes(5) for descriptions of the following attributes:

@@ -200,7 +239,7 @@
 SEE ALSO
      boot(1M), devfsadm(1M), kernel(1M), modinfo(1M), rem_drv(1M),
      update_drv(1M), driver.conf(4), system(4), attributes(5),
-     devfs(7FS), ddi_create_minor_node(9F)
+     privileges(5), devfs(7FS), ddi_create_minor_node(9F)

      Writing Device Drivers
```

```
--- devfsadm.1m Fri Oct 18 19:10:55 2002
+++ devfsadm.1m.new     Fri Jan 24 12:35:17 2003
@@ -16,7 +16,8 @@

      The default operation is to attempt to load every driver in the
      system and attach to all possible device instances. devfsadm then
-     creates logical links in /dev to device nodes in /devices.
+     creates logical links in /dev to device nodes in /devices and
+     load the device policy.

      devfsadmd(1M) is the daemon version of devfsadm(1M). The daemon
      is started by the /etc/rc* scripts during system startup and is
@@ -98,6 +99,12 @@
      /dev/.devfsadm_daemon.lock
            daemon lock file
```

```
+     /etc/security/device_policy
+           device policy file
+
+     /etc/security/extra_privs
+           additional device privileges
+
 ATTRIBUTES
      See attributes(5) for descriptions of the following attributes:

@@ -109,7 +116,7 @@
 SEE ALSO
      add_drv(1M), modinfo(1M), modload(1M), modunload(1M),
      rem_drv(1M), tapes(1M), path_to_inst(4), attributes(5),
-     devfs(7FS)
+     privileges(5), devfs(7FS)

 NOTES
      This document does not constitute an API. The /devices directory
```

```
--- ifconfig.1m Fri Jan 24 12:42:36 2003
+++ ifconfig.1m.new     Fri Jan 24 12:43:40 2003
@@ -51,13 +51,12 @@
      ifconfig command must be used at boot time to define the network
      address of each interface present on a machine; it may also be
      used at a later time to redefine an interface's address or other
-     operating parameters. If no option is specified, ifconfig
-     displays the current configuration for a network interface. If an
-     address family is specified, ifconfig reports only the details
-     specific to that address family. Only the superuser may modify
-     the configuration of a network interface. Options appearing
-     within braces ({}) indicate that one of the options must be
-     specified.
+     operating parameters. If no option is specified, ifconfig displays
+     the current configuration for a network interface. If an address
+     family is specified, ifconfig reports only the details specific to
+     that address family. Only privileged users may modify the
+     configuration of a network interface. Options appearing within
+     braces ({}) indicate that one of the options must be specified.

      The two versions of ifconfig, /sbin/ifconfig and
      /usr/sbin/ifconfig, behave differently with respect to name ser-
@@ -1026,8 +1025,8 @@
      dhcpinfo(1), dhcpagent(1M), in.mpathd(1M), in.routed(1M),
      ndd(1M), netstat(1M), ethers(3SOCKET), gethostbyname(3NSL),
      getnetbyname(3SOCKET), hosts(4), netmasks(4), networks(4),
```

```
-     nsswitch.conf(4), attributes(5), arp(7P), ipsecah(7P),
-     ipsecesp(7P), tun(7M)
+     nsswitch.conf(4), attributes(5), privileges(5), arp(7P),
+     ipsecah(7P), ipsecesp(7P), tun(7M)

      System Administration Guide, Volume 3
```

```
--- nfsd.1m     Fri Jan 24 12:50:09 2003
+++ nfsd.1m.new Fri Jan 24 12:51:34 2003
@@ -9,7 +9,8 @@

 DESCRIPTION
      nfsd is the daemon that handles client file system requests. Only
-     the super-user can run this daemon.
+     users with {PRIV_SYS_NFS} and sufficient privileges to write to
+     /var/run can run this daemon.

      The nfsd daemon is automatically invoked using share(1M) with the
      -a option.
```

```
--- update_drv.1m       Fri Oct  4 16:37:51 2002
+++ update_drv.1m.new   Mon Feb 17 14:25:19 2003
@@ -6,20 +6,14 @@
 SYNOPSIS
      update_drv [-f] device_driver

-     update_drv [-b basedir] [-f] -a -i 'identify-name' device_driver
+     update_drv [ -b basedir ] [ -f | -v ] -a
+            [-m 'permission'] [-i 'identify_name']
+            [-P privilege] [-p 'policy']  <driver_module>
+
+     update_drv [ -b basedir ] [ -f | -v ] -d
+            [-m 'permission'] [-i 'identify_name']
+            [-P privilege] [-p 'policy']  <driver_module>

-     update_drv [-b basedir] [-f] -a -m 'permission' device_driver
-
-     update_drv [-b basedir] [-f] -a -i 'identify-name' -m 'permis-
-     sion' device_driver
-
-     update_drv [-b basedir] [-f] -d -i 'identify-name' device_driver
-
-     update_drv [-b basedir] [-f] -d -m 'permission' device_driver
```

```
-
-     update_drv [-b basedir] [-f] -d -i 'identify-name' -m 'permis-
-     sion' device_driver
-
 DESCRIPTION
       The update_drv command informs the system about attribute
      changes to an installed device driver. It can be used to re-read
@@ -38,15 +32,16 @@
 OPTIONS
      The following options are supported:

-     -a    Add a permission or an aliases entry.
+     -a    Add a permission, aliases, privilege or policy entry.

            With the -a option specified, a permission entry (using the
-           -m option), or a driver's aliases entry (using the -i
-           option) can be added or updated. If a matching minor node
-           permissions entry is encountered (having the same driver
-           name and the minor node), it is replaced. If a matching
-           aliases entry is encountered (having a different driver
-           name and the same alias), an error is reported.
+           -m option), a driver's aliases entry (using the -i option),
+           a device privilege (using the -P option) or a a device
+           policy (using the -p option) can be added or updated.  If a
+           matching minor node permissions entry is encountered (having
+           the same driver name and the minor node), it is replaced. If
+           a matching aliases entry is encountered (having a different
+           driver name and the same alias), an error is reported.

            The -a and -d options are mutually exclusive.

@@ -55,12 +50,14 @@
            directory of basedir rather than installing on the system
            executing update_drv.

-     -d    Deletes a permission or an aliases entry.
+     -d    Delete a permission, aliases, privilege or policy entry.

-           Either the -m permission or -i identify-name option needs
-           to be specified with the -d option.The -d and -a options
-           are mutually exclusive.
+           The -m permission, -i identify-name, -P privilege or the
+           -p policy option needs to be specified with the -d
+           option.
```

```
+           The -d and -a options are mutually exclusive.
+
            If the entry doesn't exist update_drv returns an error.

      -f    Force the system to reread the driver.conf file even if the
@@ -79,6 +76,21 @@
            option is not specified then, this option is ignored.  The
            permission string is mandatory.

+      -p 'policy'
+           With the -a option, policy is a white-space separated list of
+           complete device policies.  For the -d option, policy is a white
+           space separated list of minor device specifications.  The
+           minor device specifications are matched exactly against the
+           entries in /etc/security/device_policy, i.e., no wildcard
+           matching is performed.
+
+      -P 'privilege'
+           With the -a option, privilege is a comma separated list of
+           additional driver privileges.  For the -d option, privilege
+           is a single privilege.  The privileges are added to or removed
+           from the /etc/security/extra_privs fie.
+
+
 EXAMPLES
      Example 1: Adding or Modifying an Existing Minor Permissions
      Entry
@@ -108,6 +120,11 @@

       example# update_drv ohci

+     Example 5: Requiring a self-defined privilege to open a tcp socket:
+
+     example# update_drv -a -P net_tcp -p \
+        'write_priv_set=net_tcp read_priv_set=net_tcp' tcp
+
 EXIT STATUS
      The following exit values are returned:

@@ -124,8 +141,8 @@
     |_____________________________|_____________________________|

 SEE ALSO
-     add_drv(1M), modunload(1M), rem_drv(1M), driver.conf(4), attri-
-     butes(5)
```

```
+     add_drv(1M), modunload(1M), rem_drv(1M), driver.conf(4),
+     attributes(5), privileges(5)

 NOTES
      If -a or -d options are specified, update_drv does not re-read
@@ -141,11 +158,11 @@
      It is possible to add an alias , which changes the driver binding
      of a device already being managed by a different driver. A force
      update with the -a option will try to bind to the new driver and
-     report error if it cannot. If both -m option and -i option are
-     specified a force flag tries to modify aliases or permissions
-     even if the other operation fails and vice-versa. A force update
-     with the -d option tries to delete entries and report the error
-     if it can not.
+     report error if it cannot. If more than one of the -m, -i, -P or -p
+     options is specified a force flag tries to modify aliases or
+     permissions even if the other operation fails and vice-versa. A
+     force update with the -d option tries to delete entries and report
+     the error if it can not.

 modified 9 Apr 2002                                   update_drv(1M)
```

```
--- Intro.2     Fri Oct  4 16:55:16 2002
+++ Intro.2.new Tue Oct  8 16:42:07 2002
@@ -36,10 +36,14 @@
            Not superuser

            Typically this error indicates an attempt to modify a file
-           in some way forbidden except to its owner or the super-
-           user. It is also returned for attempts by ordinary users to
-           do things allowed only to the super-user.
+           in some way forbidden except to its owner or an
+           appropriately privileged process.  It is also returned for
+           attempts by ordinary users to do things allowed only to
+           processes with certain privileges.

+           Manual pages document which privileges are needed to
+           override the restriction.
+
      2 ENOENT
            No such file or directory

@@ -137,6 +141,9 @@
            An attempt was made to access a file in a way forbidden by
```

```
            the protection system.

+           The manual pages document which privileges are needed to
+           override the protection system.
+
      14 EFAULT
            Bad address

@@ -1582,6 +1589,9 @@
      getmsg(2)
            get next message off a stream

+     getpflags(2)
+           get process flags
+
      getpgid(2)
            See getpid(2)

@@ -1597,6 +1607,9 @@
      getppid(2)
            See getpid(2)

+     getppriv(2)
+           get process privilege sets
+
      getprojid(2)
            See settaskid(2)

@@ -1842,11 +1855,17 @@
      setitimer(2)
            See getitimer(2)

+     setpflags(2)
+           set process flags
+
      setpgid(2)
            set process group ID

      setpgrp(2)
            set process group ID
+
+     setppriv(2)
+           manipulate process privilege sets

      setrctl(2)
```

```
            set or get resource control values
```

```
--- acct.2      Thu Oct  3 15:57:22 2002
+++ acct.2.new  Thu Oct  3 16:17:28 2002
@@ -14,7 +14,7 @@
      record will be written in an accounting file for each process
      that terminates. The termination of a process can be caused by
      either an exit(2) call or a signal(3C)). The effective user ID of
-     the process calling acct() must be super-user.
+     the process calling acct() must have the appropriate privileges.

      The path argument points to the pathname of the accounting file,
      whose file format is described on the acct.h(3HEAD) manual page.
@@ -55,13 +55,13 @@
      ENOTDIR
            A component of the path prefix is not a directory.

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM The {PRIV_SYS_ACCT} privilege is not asserted in the
+           effective set of the calling process.

      EROFS The named file resides on a read-only file system.

 SEE ALSO
-     exit(2), signal(3C), acct.h(3HEAD)
+     exit(2), signal(3C), acct.h(3HEAD), privileges(5)

 modified 5 Jul 1990                                        acct(2)
```

```
--- adjtime.2   Thu Oct  3 15:59:35 2002
+++ adjtime.2.new       Thu Oct  3 16:17:57 2002
@@ -35,7 +35,8 @@
      slow down the clocks of some machines and speed up the clocks of
      others to bring them to the average network time.

-     Only the super-user may adjust the time of day.
+     Only the processes with appropriate priveleges may adjust the time
+     of day.

      The adjustment value will be silently rounded to the resolution
      of the system clock.
@@ -56,8 +57,8 @@
```

```
            The tv_usec member of delta is not within valid range
            (-1000000 to 1000000).

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM The {PRIV_SYS_TIME} privilege is not asserted in the
+           effective set of the calling process.

      Additionally, the adjtime() function will fail for 32-bit inter-
      faces if:
@@ -68,7 +69,7 @@
            number of seconds.

 SEE ALSO
-     date(1), gettimeofday(3C)
+     date(1), gettimeofday(3C), privileges(5)

 modified 25 Sep 1997                                    adjtime(2)
```

```
--- audit.2     Thu Oct  3 16:06:52 2002
+++ audit.2.new Thu Oct  3 16:58:31 2002
@@ -40,10 +40,11 @@
            either less than the header token size or greater than
            MAXAUDITDATA.

-     EPERM The process's effective user ID is not superuser.
+     EPERM The {PRIV_PROC_AUDIT} privilege is not asserted in the
+           effective set of the calling process.

 USAGE
-     Only the superuser can successfully execute this call.
+     Only privileged processes can successfully execute this call.

 ATTRIBUTES
      See attributes(5) for descriptions of the following attributes:
@@ -56,7 +57,7 @@

 SEE ALSO
      bsmconv(1M), auditd(1M), auditon(2), auditsvc(2), getaudit(2),
-     audit.log(4), attributes(5)
+     audit.log(4), attributes(5), privileges(5)

 NOTES
      The functionality described in this man page is available only if
```

```
--- auditon.2   Thu Oct  3 16:08:24 2002
+++ auditon.2.new       Tue Oct  8 16:43:39 2002
@@ -240,11 +240,17 @@
            One of the arguments was illegal, or BSM has not been
            installed.

-     EPERM The process's effective user ID is not superuser.
+     EPERM The {PRIV_SYS_AUDIT} privilege is not asserted in the
+           effective set of the calling process.

+     EPERM Neither the {PRIV_PROC_AUDIT} nor the {PRIV_SYS_AUDIT}
+           privilege is asserted in the effective set of the
+           calling process and the command is one of A_GETCAR,
+           A_GETCLASS, A_GETCOND, A_GETCWD, A_GETPINFO, A_GETPOLICY.
+
 USAGE
      The auditon() function can be invoked only by processes with
-     superuser privileges.
+     appropriate privileges.

 ATTRIBUTES
      See attributes(5) for descriptions of the following attributes:
@@ -257,7 +263,7 @@

 SEE ALSO
      auditconfig(1M), auditd(1M), bsmconv(1M), audit(2), auditsvc(2),
-     exec(2), audit.log(4), attributes(5)
+     exec(2), audit.log(4), attributes(5), privileges(5)

 NOTES
      The functionality described in this man page is available only if
```

```
--- auditsvc.2  Thu Oct  3 16:13:05 2002
+++ auditsvc.2.new      Thu Oct  3 16:13:51 2002
@@ -74,14 +74,15 @@

      ENXIO A hangup occurred on the stream being written to.

-     EPERM The process's effective user ID is not superuser.
+     EPERM The {PRIV_SYS_AUDIT} privilege is not asserted in the
+           effective set of the calling process.

      EWOULDBLOCK
            The file was marked for 4.2 BSD-style non-blocking I/O, and
```

```
            no data could be written immediately.

 USAGE
-     Only processes with an effective user ID of superuser can execute
+     Only processes with appropriate privileges can perform
      this call successfully.

 ATTRIBUTES
```

```
--- chmod.2     Thu Oct  3 16:16:09 2002
+++ chmod.2.new Tue Oct  8 16:48:17 2002
@@ -111,12 +111,20 @@

      EPERM The effective user ID does not match the owner of the file
            and the process does not have appropriate privilege.
+           The {PRIV_FILE_SETDAC} privilege overrides constraints on
+           ownership when changing permissions on a file.
+           The {PRIV_FILE_SETID} privilege overrides constraints on
+           ownership when adding the set-uid or set-gid bits to an
+           executable or a directory.  When adding the set-uid bit
+           to a root owned executable, additional restrictions apply.
+           See privileges(5).

      The chmod() function will fail if:

      EACCES
            Search permission is denied on a component of the path pre-
-           fix of path.
+           fix of path.  The privilege {FILE_DAC_SEARCH} overrides
+           file permissions restrictions in that case.

      EFAULT
            The path argument points to an illegal address.
@@ -244,7 +252,7 @@
 SEE ALSO
      chmod(1), chown(2), creat(2), fcntl(2), mknod(2), open(2),
      read(2), rename(2), stat(2), write(2), fattach(3C), mkfifo(3C),
-     stat.h(3HEAD), attributes(5), standards(5)
+     stat.h(3HEAD), attributes(5), privileges(5), standards(5)

      Programming Interfaces Guide
```

```
--- chown.2     Thu Oct  3 16:20:04 2002
+++ chown.2.new Tue Oct  8 16:51:48 2002
```

```
@@ -43,22 +43,32 @@
      is a null pointer, the function behaves like fchown().

      If  chown(), lchown(), fchown(), or fchownat() is invoked by a
-     process other than super-user, the set-user-ID and set-group-ID
-     bits of the file mode, S_ISUID and  S_ISGID respectively, are
-     cleared (see  chmod(2)).
+     process that doesn't have {PRIV_FILE_SETID} asserted in its
+     effective set, the set-user-ID and set-group-ID bits of the file
+     mode, S_ISUID and S_ISGID respectively, are cleared (see
+     chmod(2)).  Additional restrictions apply when changing
+     the ownership to uid 0.
+
+     The operating system defines several privileges to override
+     restrictions on the chown() family of functions.  When the
+     {PRIV_FILE_CHOWN} privilege is asserted in the effective set of
+     the current process, there are no restrictions except in the
+     special circumstances of chowning to or from uid 0; when the
+     {PRIV_FILE_CHOWN_SELF} privilege is asserted, ownership changes
+     are restricted to the files of which the ownership matches the
+     effective user id of the current process.  If neither privilege
+     is asserted in the effective set of the calling process,
+     ownership changes are limited to changes of the group of the
+     file to the list of supplementary group IDs and the effective
+     group id.

      The operating system provides a configuration option,
-     {_POSIX_CHOWN_RESTRICTED}, to restrict ownership changes for the
-     chown(), lchown(), and  fchown() functions. When
-     {_POSIX_CHOWN_RESTRICTED} is not in effect, either the effective
-     user ID of the process must match the owner of the file or the
-     process must be the super-user to change the  ownership of a
-     file. When {_POSIX_CHOWN_RESTRICTED} is in effect (the default
-     behavior), the chown(), lchown(), and fchown() functions, for
-     users other than super-user, prevent the owner of the file from
-     changing the owner ID  of the file and restrict the change of the
-     group of the file to the list of supplementary group IDs. To set
-     this configuration option, include the following line in
+     {_POSIX_CHOWN_RESTRICTED}, to control the default behaviour
+     of processes and the behaviour of the NFS server.
+     If {_POSIX_CHOWN_RESTRICTED} is not in effect, the privilege
+     {PRIV_FILE_CHOWN_SELF} is asserted in the inheritable set
+     of all processes unless overridden by policy.conf(4) or user_attr(4).
+     To set this configuration option, include the following line in
      /etc/system:
```

```
           set rstchown = 1
@@ -120,9 +130,9 @@
            descriptor provided does not refer to a valid directory.

      EPERM The effective user ID does not match the owner of the file
-           or the process is not the super-user and
-           _POSIX_CHOWN_RESTRICTED indicates that such privilege is
-           required.
+           or the {PRIV_FILE_CHOWN} privilege is not asserted in the
+           effective set of the calling process and neither is the
+           {PRIV_FILE_CHOWN_SELF} privilege.

      EROFS The named file  resides on a read-only file system.

@@ -146,10 +156,10 @@
      EINVAL
            The group or owner argument is out of range.

-     EPERM The effective user ID does not match the owner of the file,
-           or the process is not the super-user and
-           _POSIX_CHOWN_RESTRICTED indicates that such privilege is
-           required.
+     EPERM The effective user ID does not match the owner of the file
+           or the {PRIV_FILE_CHOWN} privilege is not asserted in the
+           effective set of the calling process and neither is the
+           {PRIV_FILE_CHOWN_SELF} privilege.

      EROFS The named file referred to by fildes resides on a read-only
            file system.
```

```
--- chroot.2    Thu Oct  3 15:50:15 2002
+++ chroot.2.new        Thu Oct  3 16:19:27 2002
@@ -20,11 +20,11 @@
      fildes argument to fchroot() is the open file descriptor of the
      directory which is to become the root.

-     The effective user ID of the process must be super-user to change
-     the root directory. While it is always possible to change to the
-     system root using the fchroot() function, it is not guaranteed to
-     succeed in any other case, even should fildes be valid in all
-     respects.
+     The privilege {PRIV_PROC_CHROOT} must be asserted in the effective
+     set of the process to change the root directory. While it is
+     always possible to change to the system root using the fchroot()
```

```
+     function, it is not guaranteed to succeed in any other case, even
+     should fildes be valid in all respects.

      The ".." entry in the root directory is interpreted to mean the
      root directory itself. Therefore, ".." cannot be used to access
@@ -79,11 +79,11 @@
      ENOTDIR
            Any component of the path name is not a directory.

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM The {PRIV_PROC_CHROOT} privilege is not asserted in the
+           effective set of the calling process.

 SEE ALSO
-     chroot(1M), chdir(2)
+     chroot(1M), chdir(2), privileges(5)

 WARNINGS
      The only use of fchroot() that is appropriate is to change back
```

```
--- exec.2      Fri Oct  4 11:51:56 2002
+++ exec.2.new  Mon Jan  6 12:27:50 2003
@@ -162,9 +162,23 @@
      of the new process image are saved (as the saved set-user-ID and
      the saved set-group-ID for use by setuid(2).

-     If the effective user-ID is root or superuser, the set-user-ID
-     and set-group-ID bits will be honored when the process is being
-     controlled by ptrace().
+     The privilege sets are changed using the following rules; first,
+     the inheritable set, I, is intersected with the limit set, L.
+     It is this mechanisms that enforces the limit set for processes.
+     Secondly, the effective set, E, and the permitted set, P, are
+     made equal to the new inheritable set.
+
+     The system will attempt to set the privilege aware state to
+     non-PA both before doing any modifications to the process ids and
+     privilege sets as well as after completing the transition to
+     new uids and privilege sets, following the rules outlined in
+     privileges(5).
+
+     If the {PRIV_PROC_OWNER} privilege is asserted in the effective
+     set, the set-user-ID and set-group-ID bits will be honored when
+     the process is being controlled by ptrace(); additional
```

```
+     restriction may apply when the traced process has an effective
+     uid of 0 (see privileges(5)).

      Any shared memory segments attached to the calling process image
      will not be attached to the new process image (see shmop(2)). Any
@@ -264,6 +278,10 @@

         o  processor set bindings (see pset_bind(2))

+        o  limit privilege set
+
+        o  privilege debugging flag (see privileges(5), setpflags(2))
+
      A call to any exec function from a process with more than one
      thread results in all threads being terminated and the new exe-
      cutable image being loaded and executed. No destructor functions
@@ -303,7 +321,9 @@
            Search permission is denied for a directory listed in the
            new process file's path prefix; the new process file is not
            an ordinary file; or the new process file mode denies exe-
-           cute permission.
+           cute permission.  The {FILE_DAC_SEARCH} privilege overrides
+           restriction on directory searches; the {FILE_DAC_EXECUTE}
+           privilege overrides lack of execute permission.

      EAGAIN
            Total amount of system memory available when reading using
@@ -387,7 +407,7 @@
      priocntl(2), profil(2), semop(2), shmop(2), sigpending(2), sig-
      procmask(2), times(2), umask(2), lockf(3C), ptrace(3C),
      setlocale(3C), signal(3C), system(3C), timer_create(3RT),
-     a.out(4), attributes(5), environ(5), standards(5)
+     a.out(4), attributes(5), environ(5), privileges(5), standards(5)

 WARNINGS
      If a program is setuid to a user ID other than the superuser, and
```

```
--- fork.2      Thu Oct  3 16:01:07 2002
+++ fork.2.new  Thu Oct  3 16:01:45 2002
@@ -162,6 +162,9 @@
      ENOMEM
            There is not enough swap space.

+     EPERM The {PRIV_PROC_FORK} privilege is not asserted in the
+           effective set of the calling process.
```

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

@@ -176,7 +179,8 @@
      memcntl(2), mmap(2), nice(2), priocntl(2), semop(2), shmop(2),
      times(2), umask(2), wait(2), exit(3C), plock(3C),
      pthread_atfork(3THR), ptrace(3C), signal(3C), system(3C),
-     thr_create(3THR) timer_create(3RT), attributes(5), standards(5)
+     thr_create(3THR) timer_create(3RT), attributes(5), privileges(5),
+     standards(5)

 NOTES
      An applications should call _exit() rather than exit(3C) if it
```

```
--- getacct.2   Thu Oct  3 16:33:48 2002
+++ getacct.2.new       Thu Oct  3 16:34:58 2002
@@ -83,8 +83,8 @@
            getacct() attempted to retrieve accounting data for a pro-
            cess when extended process accounting was inactive.

-     EPERM The invoking process lacks sufficient permission to perform
-           the request operation.
+     EPERM The {PRIV_SYS_ACCT} privilege is not asserted in the
+           effective set of the calling process.

      ERSCH The id argument does not refer to a presently active system
            task ID or process ID.
```

```
--- getauid.2   Thu Jan 23 12:51:26 2003
+++ getauid.2.new       Thu Jan 23 12:52:59 2003
@@ -37,7 +37,13 @@
      EFAULT
            The auid argument points to an invalid address.

-     EPERM The process’s effective user ID is not super-user.
+     EPERM {PRIV_SYS_AUDIT} is not asserted in the effective set
+           of the calling process.
+
+     The getauid() function will fail if:
+
+     EPERM {PRIV_PROC_AUDIT} is not asserted in the effective set
+           of the calling process.
```

```
 USAGE
      Only the super-user may successfully execute these calls.
```

```
--- getgroups.2 Thu Jan 23 12:54:21 2003
+++ getgroups.2.new     Fri Jan 24 12:35:25 2003
@@ -51,11 +51,12 @@
      EINVAL
            The value of ngroups is greater than {NGROUPS_MAX}.

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM {PRIV_PROC_SETID} is not asserted in the effective set
+           of the calling process.

 USAGE
-     Use of the setgroups() function requires superuser privileges.
+     Use of the setgroups() function requires the {PRIV_PROC_SETID}
+     privilege.

 ATTRIBUTES
      See attributes(5) for descriptions of the following attributes:
@@ -68,7 +69,7 @@

 SEE ALSO
      groups(1), chown(2), getuid(2), setuid(2), getgrnam(3C),
-     initgroups(3C), attributes(5), standards(5)
+     initgroups(3C), attributes(5), privileges(5), standards(5)

 modified 25 Jul 2001                                  getgroups(2)
```

```
--- getrlimit.2 Thu Jan 23 22:00:21 2003
+++ getrlimit.2.new     Fri Jan 24 12:35:28 2003
@@ -23,12 +23,12 @@
      its may be changed by a process to any value that is less than or
      equal to the hard limit. A process may (irreversibly) lower its
      hard limit to any value that is greater than or equal to the soft
-     limit. Only a process with an effective user ID of super-user can
-     raise a hard limit.  Both hard and soft limits can be changed in
-     a single call to setrlimit() subject to the constraints described
-     above. Limits may have an "infinite" value of RLIM_INFINITY. The
-     rlp argument is a pointer to struct rlimit that includes the fol-
-     lowing members:
+     limit. Only a process with {PRIV_SYS_RESOURCE} asserted in the
```

```
+     effective set can raise a hard limit.  Both hard and soft limits
+     can be changed in a single call to setrlimit() subject to the
+     constraints described above. Limits may have an "infinite" value
+     of RLIM_INFINITY. The rlp argument is a pointer to struct rlimit
+     that includes the following members:

      rlim_t    rlim_cur;     /* current (soft) limit */
      rlim_t    rlim_max;     /* hard limit */
@@ -161,8 +161,8 @@
            call, the new rlim_cur exceeds the new rlim_max.

      EPERM The limit specified to setrlimit() would have raised the
-           maximum limit value, and the effective user of the calling
-           process is not super-user.
+           maximum limit value, and {PRIV_SYS_RESOURCE} is not
+           asserted in the effective set of the current process.

      The setrlimit() function may fail if:
```

```
--- issetugid.2 Thu Oct 24 17:02:28 2002
+++ issetugid.2.new     Thu Oct 24 17:05:44 2002
@@ -12,7 +12,8 @@
 DESCRIPTION
      The issetugid() function enables library functions (in libterm-
      lib, libc, or other libraries) to guarantee safe behavior when
-     used in setuid or setgid programs. Some library functions might
+     used in setuid, setgid programs or programs which run with more
+     privileges after a succesful exec(2). Some library functions might
      be passed insufficient information and not know whether the
      current program was started setuid or setgid because a higher
      level calling code might have made changes to the uid, euid, gid,
@@ -35,8 +36,9 @@
      uid, euid, gid, and egid permissions and on the modes of the exe-
      cutable file. If the new executable file modes are setuid or set-
      gid, or if the existing process is executing the new image with
-     uid != euid or gid != egid, issetugid() will return 1 in the new
-     process.
+     uid != euid or gid != egid or when the permitted set before
+     exec is not a superset of the inheritable set at that time,
+     issetugid() will return 1 in the new process.

 RETURN VALUES
      The issetugid() function returns 1 if the process was made setuid
@@ -57,7 +59,8 @@
```

```
     |_____________________________|_____________________________|

 SEE ALSO
-     exec(2), fork(2), setuid(2), getenv(3C), attributes(5)
+     exec(2), fork(2), setuid(2), getenv(3C), attributes(5).
+     privileges(5)

 modified 5 Oct 2001                                  issetugid(2)
```

```
--- kill.2      Thu Oct  3 16:39:04 2002
+++ kill.2.new  Thu Oct  3 16:43:31 2002
@@ -20,10 +20,13 @@

      The real or effective user ID of the sending process must match
      the real or saved (from one of functions in the exec family, see
-     exec(2)) user ID of the receiving process unless the effective
-     user ID of the sending process is superuser, (see intro(2)), or
-     sig is SIGCONT and the sending process has the same session ID as
-     the receiving process.
+     exec(2)) user ID of the receiving process unless the privilege
+     {PRIV_PROC_OWNER} is asserted in the effective set of the sending
+     process, (see intro(2)), or sig is SIGCONT and the sending process
+     has the same session ID as the receiving process.  Additionally,
+     a process needs the basic privilege (see privileges(5))
+     {PRIV_PROC_SESSION} to send signals to processes with a different
+     session ID.

      If pid is greater than 0, sig will be sent to the process whose
      process ID is equal to  pid.
@@ -60,9 +63,13 @@
            (pid_t)1 (that is, the calling process does not have per-
            mission to send the signal to any of the processes speci-
            fied by pid); or the effective user of the calling process
-           does not match the real or saved user and is not super-
-           user, and the calling process is not sending SIGCONT to a
-           process that shares the same session ID.
+           does not match the real or saved user and the calling
+           process does not have the {PRIV_PROC_OWNER} privilege
+           asserted in the effective set, and the calling process is
+           not sending SIGCONT to a process that shares the same
+           session ID or the calling process does not have the
+           {PRIV_PROC_SESSION} privilege asserted and is trying to send
+           a signal to a process with a different session ID.
```

```
      ESRCH No process or process group can be found corresponding to
            that specified by pid.
```

```
--- link.2      Thu Oct  3 16:35:11 2002
+++ link.2.new  Fri Oct  4 16:54:28 2002
@@ -17,8 +17,8 @@

      To create hard links, both files must be on the same file system.
      Both the old and the new link share equal access and rights to
-     the underlying object. The super-user may make multiple links to
-     a directory. Unless the caller is the super-user, the file named
+     the underlying object. Privileged processes may make multiple links to
+     a directory. Unless the caller is the privileged, the file named
      by existing must not be a directory.

      Upon successful completion, link() marks for update the st_ctime
@@ -79,8 +79,13 @@
      ENOTDIR
            A component of either path prefix is not a directory.

-     EPERM The file named by existing is a directory and the effective
-           user of the calling process is not super-user.
+     EPERM The file named by existing is a directory and the
+           {PRIV_SYS_LINKDIR} privilege is not asserted in the
+           effective set of the calling process.
+
+     EPERM The effective user ID does not match the owner of the file
+           and the {PRIV_FILE_LINK_ANY} privilege is not asserted in the
+           effective set of the calling process.

      EROFS The requested link requires writing in a directory on a
            read-only file system.
@@ -98,7 +103,7 @@
     |_____________________________|_____________________________|

 SEE ALSO
-     symlink(2), unlink(2), attributes(5), standards(5)
+     symlink(2), unlink(2), attributes(5), privileges(5), standards(5)

 modified 28 Dec 1996                                        link(2)
```

```
--- memcntl.2   Fri Jan 24 12:46:02 2003
+++ memcntl.2.new       Fri Jan 24 12:47:43 2003
```

```
@@ -248,9 +248,9 @@
            the address space of a process or specify one or more pages
            which are not mapped.

-     EPERM The process's effective user ID is not superuser and
-           MC_LOCK, MC_LOCKAS, MC_UNLOCK, or MC_UNLOCKAS was speci-
-           fied.
+     EPERM {PRIV_PROC_LOCK_MEMORY} is not asserted in the effective
+           set of the calling process and MC_LOCK, MC_LOCKAS,
+           MC_UNLOCK, or MC_UNLOCKAS was speci- fied.

 ATTRIBUTES
      See attributes(5) for descriptions of the following attributes:
@@ -263,7 +263,7 @@
 SEE ALSO
      ppgsz(1), fork(2) mmap(2), mprotect(2), getpagesizes(3C),
      mctl(3UCB), mlock(3C), mlockall(3C), msync(3C), plock(3C),
-     sysconf(3C), attributes(5)
+     sysconf(3C), attributes(5), privileges(5)

 modified 11 Dec 2001                                    memcntl(2)
```

```
--- mknod.2     Fri Jan 24 12:48:10 2003
+++ mknod.2.new Fri Jan 24 12:48:49 2003
@@ -133,8 +133,8 @@
      ENOTDIR
            A component of the path prefix is not a directory.

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM {RPIV_SYS_DEVICES} is not asserted in the effective set
+           of the calling process.

      EROFS The directory in which the file is to be created is located
            on a read-only file system.
@@ -172,8 +172,8 @@
 SEE ALSO
      chmod(2), creat(2), exec(2), mkdir(2), open(2), stat(2), sym-
      link(2), umask(2), door_create(3DOOR), fattach(3C), makedev(3C),
-     mkfifo(3C), socket(3SOCKET), stat.h(3HEAD), attributes(5), stan-
-     dards(5)
+     mkfifo(3C), socket(3SOCKET), stat.h(3HEAD), attributes(5),
+     privileges(5), standards(5)
```

```
 modified 27 Aug 2002                                          mknod(2)
```

```
--- mount.2     Thu Oct  3 16:44:04 2002
+++ mount.2.new Thu Oct  3 16:44:56 2002
@@ -156,7 +156,8 @@
            optptr argument exceeds the size of the buffer specified by
            optlen.

-     EPERM The effective user ID is not superuser.
+     EPERM The {PRIV_SYS_MOUNT} privilege is not asserted in the
+           effective set of the calling process.

      EREMOTE
            The spec argument is remote and cannot be mounted.
@@ -166,7 +167,7 @@

 USAGE
      The mount() function can be invoked only by  processes with
-     superuser privileges.
+     appropriate privileges.

 SEE ALSO
      mount(1M), umount(2), mnttab(4)
```

```
--- msgctl.2    Thu Oct  3 16:52:39 2002
+++ msgctl.2.new        Fri Jan 24 12:35:36 2003
@@ -38,9 +38,10 @@
            Remove the message queue identifier specified by msqid from
            the system and destroy the message queue and data structure
            associated with it. This cmd can only be executed by a pro-
-           cess that has an effective user ID equal to either that of
-           super-user, or to the value of msg_perm.cuid or
-           msg_perm.uid in the data structure associated with msqid.
+           cess that has an effective user ID equal to that of to the
+           value of msg_perm.cuid or msg_perm.uid in the data structure
+           associated with msqid or with appropriate privileges
+           asserted in the effective set.
            The buf argument is ignored.

 RETURN VALUES
@@ -62,14 +63,16 @@
            or the cmd argument is not a valid command or is IPC_SET
            and msg_perm.uid or msg_perm.gid is not valid.
```

```
-     EPERM The cmd argument is IPC_RMID or IPC_SET and the effective
-           user ID of the calling process is not super-user and is not
-           equal to the value of msg_perm.cuid or msg_perm.uid in the
-           data structure associated with msqid.
+     EPERM The cmd argument is equal to IPC_RMID or IPC_SET and the
+           effective user ID of the calling process is not equal to the
+           value of shm_perm.cuid or shm_perm.uid in the data structure
+           associated with shmid and {PRIV_IPC_OWNER} is not asserted
+           in the effective set of the calling process.

      EPERM The cmd argument is IPC_SET, an attempt is being made to
-           increase to the value of msg_qbytes, and the effective user
-           ID of the calling process is not super-user.
+           increase to the value of msg_qbytes, and the
+           {PRIV_SYS_IPC_CONFIG} privilege is not asserted in the
+           effective set of the calling process.

      EOVERFLOW
           The cmd argument is IPC_STAT and uid or gid is too large to
@@ -84,8 +87,8 @@
     |_____________________________|_____________________________|

 SEE ALSO
-     intro(2), msgget(2), msgrcv(2), msgsnd(2), attributes(5), stan-
-     dards(5)
+     intro(2), msgget(2), msgrcv(2), msgsnd(2), attributes(5),
+     privileges(5), standards(5)

 modified 2 Feb 1996                                    msgctl(2)
```

```
--- nice.2      Thu Jan 23 12:28:23 2003
+++ nice.2.new  Fri Jan 24 12:35:39 2003
@@ -42,8 +42,9 @@
           The nice() function is called by a process in a scheduling
           class other than time-sharing or fixed-priority.

-     EPERM The incr argument is negative or greater than 40 and the
-           effective user ID of the calling process is not superuser.
+     EPERM The incr argument is negative or greater than 40 and
+           {PRIV_PROC_PRIOCNTL} is not asserted in the effective set
+           of the calling process.

 USAGE
```

```
      The priocntl(2) function is a more general interface to scheduler
@@ -65,7 +66,7 @@

 SEE ALSO
      nice(1), exec(2), priocntl(2), getpriority(3C), attributes(5),
-     standards(5)
+     privileges(5), standards(5)

 modified 6 Mar 2002                                          nice(2)
```

```
--- ntp_adjtime.2       Thu Jan 23 12:29:31 2003
+++ ntp_adjtime.2.new   Fri Jan 24 12:35:41 2003
@@ -54,10 +54,11 @@
            The constant member of the structure pointed to by tptr is
            less than 0 or greater than 30.

-     EPERM The user is not super-user.
+     EPERM {PRIV_SYS_TIME} is not asserted in the effective set of
+           the calling process.

 SEE ALSO
-     xntpd(1M), ntp_gettime(2)
+     xntpd(1M), ntp_gettime(2), privileges(5)

 modified 9 Nov 1999                                  ntp_adjtime(2)
```

```
--- open.2      Mon Oct  7 23:36:46 2002
+++ open.2.new  Mon Oct  7 23:45:10 2002
@@ -245,6 +245,14 @@
            mission is denied for the parent directory of the file to
            be created, or O_TRUNC is specified and write permission is
            denied.
+           The privilege {PRIV_FILE_DAC_SEARCH} allows processes to
+           search directories regardless of permission bits.
+           The privilege {PRIV_FILE_DAC_WRITE} allows processes to
+           open files for writing regardless of permission bits;
+           see privileges(5) for special considerations when
+           opening files owned by uid 0 for writing.  The privilege
+           {PRIV_FILE_DAC_READ} allows processes to open files for
+           reading regardless of permission bits.

      EBADF The file descriptor provided to openat() is invalid.
```

```
@@ -458,8 +466,8 @@
      intro(2), chmod(2), close(2), creat(2), dup(2), exec(2),
      fcntl(2), getmsg(2), getrlimit(2), lseek(2), putmsg(2), read(2),
      stat(2), umask(2), write(2), attropen(3C), fcntl.h(3HEAD),
-     stat.h(3HEAD), unlockpt(3C), attributes(5), lf64(5), stan-
-     dards(5), connld(7M), streamio(7I)
+     stat.h(3HEAD), unlockpt(3C), attributes(5), privileges(5),
+     lf64(5), standards(5), connld(7M), streamio(7I)

 NOTES
```

```
--- p_online.2  Thu Jan 23 12:30:36 2003
+++ p_online.2.new      Fri Jan 24 12:35:46 2003
@@ -61,8 +61,8 @@
 ERRORS
      The p_online() function will fail if:

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM {PRIV_SYS_CPU_CONFIG} is not asserted in the effective
+           set of the calling process.

      EINVAL
            A non-existent processor ID was specified or  flag was
```

```
--- priocntl.2  Thu Jan 23 12:31:48 2003
+++ priocntl.2.new      Fri Jan 24 12:35:49 2003
@@ -100,7 +100,7 @@
      PC_SETPARMS or PC_SETXPARMS command as explained below) , the
      real or effective user ID of the LWP calling priocntl() must
      match the real or effective user ID of the receiving LWP or the
-     effective user ID of the calling LWP must be superuser. These are
+     the calling LWP must have sufficient privileges. These are
      the minimum permission requirements enforced for all classes. An
      individual class might impose additional permissions requirements
      when setting LWPs to that class and/or when setting class-
@@ -584,9 +584,9 @@
      mand PC_SETXPARMS knows no special value RT_NOCHANGE.

      To change the class of an LWP to realtime from any other class,
-     the LWP invoking priocntl() must have superuser privileges. To
+     the LWP invoking priocntl() must have sufficient privileges. To
```

```
      change the priority or time quantum setting of a realtime LWP,
-     the LWP invoking priocntl() must have superuser privileges or
+     the LWP invoking priocntl() must have sufficient privileges or
      must itself be a realtime LWP whose real or effective user ID
      matches the real of effective user ID of the target LWP.

@@ -665,10 +665,10 @@

      Any time-sharing LWP can lower its own ts_uprilim (or that of
      another LWP with the same user ID). Only a time-sharing LWP with
-     superuser privileges can raise a ts_uprilim. When changing the
-     class of an LWP to time-sharing from some other class, superuser
+     sufficient privileges can raise a ts_uprilim. When changing the
+     class of an LWP to time-sharing from some other class, sufficient
      privileges are required to set the initial ts_uprilim to a value
-     greater than 0. Attempts by a non-superuser LWP to raise a
+     greater than 0. Attempts by a unprivileged LWP to raise a
      ts_uprilim or set an initial ts_uprilim greater than 0 fail with
      a return value of -1 and errno set to EPERM.

@@ -803,10 +803,10 @@

      Any fair-share LWP can lower its own fss_uprilim (or that of
      another LWP with the same user ID). Only a fair-share LWP with
-     superuser privileges can raise an fss_uprilim. When changing the
-     class of an LWP to fair-share from some other class, superuser
+     sufficient privileges can raise an fss_uprilim. When changing the
+     class of an LWP to fair-share from some other class,
      privileges are required to set the initial fss_uprilim to a value
-     greater than 0. Attempts by a non-superuser LWP to raise an
+     greater than 0. Attempts by a unprivileged LWP to raise an
      fs_uprilim or set an initial fs_uprilim greater than 0 fail with
      a return value of -1 and errno set to EPERM.

@@ -897,10 +897,10 @@

      Any fixed-priority LWP can lower its own fx_uprilim (or that of
      another LWP with the same user ID). Only a fixed-priority LWP
-     with superuser privileges can raise a fx_uprilim. When changing
+     with sufficient privileges can raise a fx_uprilim. When changing
      the class of an LWP to fixed-priority from some other class,
-     superuser privileges are required to set the initial fx_uprilim
-     to a value greater than 0. Attempts by a non-superuser LWP to
+     sufficient privileges are required to set the initial fx_uprilim
+     to a value greater than 0. Attempts by a unprivileged LWP to
```

```
      raise a fx_uprilim or set an initial fx_uprilim greater than 0
      fail with a return value of -1 and errno set to EPERM.

@@ -1043,8 +1043,12 @@
            An attempt to change the class of an LWP failed because of
            insufficient memory.

-     EPERM The effective user of the calling LWP is not superuser.
+     EPERM {PRIV_PROC_PRIOCNTL} is not asserted in the effective set of
+           the calling LWP.

+     EPERM The calling LWP does not have sufficient privileges to
+           affect the target LWP.
+
      ERANGE
            The requested time quantum is out of range.

@@ -1053,7 +1057,8 @@
 SEE ALSO

      priocntl(1), dispadmin(1M), init(1M), _lwp_create(2), exec(2),
-     fork(2), nice(2), priocntlset(2), fx_dptbl(4), rt_dptbl(4)
+     fork(2), nice(2), priocntlset(2), fx_dptbl(4), rt_dptbl(4),
+     privileges(5)

      System Administration Guide: Basic Administration
```

```
--- priocntlset.2       Thu Jan 23 12:33:00 2003
+++ priocntlset.2.new   Fri Jan 24 12:35:51 2003
@@ -89,8 +89,11 @@
            An attempt to change the class of a process failed because
            of insufficient memory.

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM {PRIV_PROC_PRIOCNTL} is not asserted in the effective set of
+           the calling LWP.
+
+     EPERM The calling LWP does not have sufficient privileges to
+           affect the target LWP.

      ERANGE
            The requested time quantum is out of range.
```

```
--- processor_bind.2    Thu Jan 23 12:37:41 2003
+++ processor_bind.2.new        Fri Jan 24 12:35:55 2003
@@ -64,9 +64,10 @@
            The specified processor is not on-line, or the idtype argu-
            ment was not P_PID, P_LWPID, P_PROJID, or P_TASKID.

-     EPERM The effective user of the calling process is not superuser,
-           and its real or effective user ID does not match the real
-           or effective user ID of one of the LWPs being bound.
+     EPERM {PRIV_PROC_OWNER} privilege is not asserted in the
+           effective set of the calling process and its real or
+           effective user ID does not match the real or effective user
+           ID of one of the LWPs being bound.

      ESRCH No processes, LWPs, or tasks were found to match the cri-
            teria specified by  idtype and  id.
```

```
--- pset_bind.2 Thu Jan 23 12:39:51 2003
+++ pset_bind.2.new     Fri Jan 24 12:36:02 2003
@@ -77,13 +77,15 @@
            An invalid processor set ID was specified;  or idtype was
            not P_PID, P_LWPID, P_PROJID, or P_TASKID.

-     EPERM The effective user of the calling process is not superuser
-           and either the real or effective user ID of the calling
-           process does not match the real or effective user ID of one
-           of the LWPs being bound, or the processor set from which
-           one or more of the LWPs are being unbound has the
-           PSET_NOESCAPE attribute set. See pset_setattr(2) for more
-           information about processor set attributes.
+     EPERM {PRIV_PROC_OWNER} is not asserted in the effecive set of
+           the calling process and either the real or effective user ID
+           of the calling process does not match the real or effective
+           user ID of one of the LWPs being bound, or the processor set
+           from which one or more of the LWPs are being unbound has the
+           PSET_NOESCAPE attribute set and {PRIV_SYS_CPU_CONFIG) is
+           not asserted in the effective set of the calling process.
+           See pset_setattr(2) for more information about processor set
+           attributes.

      ESRCH No processes, LWPs, or tasks were found to match the cri-
            teria specified by idtype and id.
@@ -100,7 +102,8 @@
 SEE ALSO
```

```
      pbind(1M), psrset(1M), exec(2), fork(2), processor_bind(2),
      pset_create(2), pset_info(2), pset_setattr(2),
-     pset_getloadavg(3C), project(4), attributes(5)
+     pset_getloadavg(3C), project(4), attributes(5),
+     privileges(5)

 modified 11 Sep 2001                                  pset_bind(2)
```

```
--- pset_create.2       Thu Jan 23 12:41:43 2003
+++ pset_create.2.new   Fri Jan 24 12:36:05 2003
@@ -52,7 +52,7 @@
      a processor set, processor cpu is released from its current pro-
      cessor set.

-     These functions are restricted to super-user use, except for
+     These functions are restricted to privileged processes, except for
      pset_assign() when pset is PS_QUERY.

 RETURN VALUES
@@ -79,8 +79,8 @@
            There was insufficient space for pset_create to create a
            new processor set.

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM {PRIV_PROC_CPU_CONFIG} os not asserted int he effectvie set
+           of the calling process.

 ATTRIBUTES
      See attributes(5) for descriptions of the following attributes:
@@ -94,7 +94,7 @@
 SEE ALSO
      psradm(1M), psrinfo(1M), psrset(1M), p_online(2),
      processor_bind(2), pset_bind (2), pset_info(2),
-     pset_getloadavg(3C), attributes(5)
+     pset_getloadavg(3C), attributes(5), privileges(5)

 NOTES
      Processors belonging to different processor sets of type
```

```
--- rmdir.2     Thu Jan 23 12:44:01 2003
+++ rmdir.2.new Fri Jan 24 12:36:06 2003
@@ -35,12 +35,13 @@
```

```
      EACCES
            Search permission is denied for a component of the path
-           prefix; write permission is denied on the directory con-
-           taining the directory to be removed; the parent directory
-           has the S_ISVTX variable set and
-            is not owned by the user; the directory is not owned by
-           the user and is not writable by the user; or the user is
-           not a super-user.
+           prefix and {PRIV_FILE_DAC_SEARCH} is not asserted in the
+           effective set of the calling process; write permission is
+           denied on the directory containing the directory to be removed
+           and {PRIV_FILE_DAC_WRITE} is not asserted; the parent directory
+           has the S_ISVTX variable set and is not owned by the user and
+           {PRIV_FILE_OWNER} is not asserted; the directory is not owned by
+           the user and is not writable by the user.

      EBUSY The directory to be removed is the mount point for a
            mounted file system.
@@ -89,5 +90,5 @@
     |_____________________________|_____________________________|

 SEE ALSO
-     mkdir(1), rm(1), mkdir(2), attributes(5), standards(5)
+     mkdir(1), rm(1), mkdir(2), attributes(5), privileges(5), standards(5)
```

```
--- semctl.2    Thu Jan 23 12:47:09 2003
+++ semctl.2.new        Fri Jan 24 12:36:10 2003
@@ -135,9 +135,10 @@
            IPC_SET and sem_perm.uid or sem_perm.gid is not valid.

      EPERM The cmd argument is equal to IPC_RMID or IPC_SET and the
-           effective user of the calling process is not super-user, or
-           cmd is equal to the value of sem_perm.cuid or sem_perm.uid
-           in the data structure associated with semid.
+           effective user ID of the calling process is not equal to the
+           value of shm_perm.cuid or shm_perm.uid in the data structure
+           associated with shmid and {PRIV_IPC_OWNER} is not asserted
+           in the effective set of the calling process.

      EOVERFLOW
            The cmd argument is IPC_STAT and uid or gid is too large to
@@ -157,8 +158,8 @@
     |_____________________________|_____________________________|
```

```
 SEE ALSO
-     ipcs(1), intro(2), semget(2), semop(2), attributes(5), stan-
-     dards(5)
+     ipcs(1), intro(2), semget(2), semop(2), attributes(5),
+     privileges(5), standards(5)

 modified 7 Jan 2001                                    semctl(2)
```

```
--- setregid.2  Thu Jan 23 21:57:02 2003
+++ setregid.2.new      Fri Jan 24 12:36:13 2003
@@ -41,10 +41,11 @@
            The value of rgid or egid is less than 0 or greater than
            UID_MAX (defined in <limits.h>).

-     EPERM The calling process's effective UID is not the super-user
-           and a change other than changing the real group ID to the
-           saved set-group-ID or changing the effective group ID to
-           the real group ID or the saved group ID, was specified.
+     EPERM The {PRIV_PROC_SETID} privilege is not asserted in the
+           effective set of the calling processes and a change other
+           than changing the real group ID to the saved set-group-ID or
+           changing the effective group ID to the real group ID or the
+           saved group ID, was specified.

 USAGE
      If a set-group-ID process sets its effective group ID to its real
@@ -61,7 +62,7 @@

 SEE ALSO
      execve(2), getgid(2), setreuid(2), setuid(2), attributes(5),
-     standards(5)
+     privileges(5), standards(5)

 modified 21 Nov 1996                                  setregid(2)
```

```
--- setreuid.2  Thu Jan 23 21:57:09 2003
+++ setreuid.2.new      Fri Jan 24 12:36:16 2003
@@ -41,10 +41,13 @@
            The value of ruid or euid is less than  0 or greater than
            UID_MAX (defined in <limits.h>).
```

```
-     EPERM The calling process's effective user ID is not the super-
-           user and a change other than changing the real user ID to
-           the effective user ID, or changing the effective user ID to
-           the real user ID or the saved set-user ID, was specified.
+     EPERM The {PRIV_PROC_SETID} privilege is not asserted in the
+           effective set of the calling processes and a change other
+           than changing the real user ID to the effective user ID, or
+           changing the effective user ID to the real user ID or the
+           saved set-user ID, was specified.  See privileegs(5) for
+           additional restrictions which apply when changing to
+           uid 0.

 USAGE
      If a set-user-ID process sets its effective user ID to its real
@@ -60,8 +63,8 @@
     |_____________________________|_____________________________|

 SEE ALSO
-     exec(2), getuid(2), setregid(2), setuid(2), attributes(5), stan-
-     dards(5)
+     exec(2), getuid(2), setregid(2), setuid(2), attributes(5),
+     privileges(5), standards(5)

 modified 21 Nov 1996                                  setreuid(2)
```

```
--- settaskid.2 Thu Jan 23 22:02:20 2003
+++ settaskid.2.new     Fri Jan 24 12:36:20 2003
@@ -23,7 +23,7 @@
      The settaskid() function makes a request of the system to assign
      a new task ID to the calling process, changing the associated
      project ID to that specified. The calling process must have
-     superuser privileges to perform this operation. The flags argu-
+     sufficient privileges to perform this operation. The flags argu-
      ment should be either TASK_NORMAL for a regular task, or
      TASK_FINAL, which disallows subsequent settaskid() calls by the
      created task.
@@ -45,7 +45,8 @@
      EACCES
           The invoking task was created with the TASK_FINAL flag.

-     EPERM The effective user of the calling process is not superuser.
+     EPERM {PRIV_PROC_TASKID} is not asserted in the effective set
+           of the calling process.
```

```
      EINVAL
            The given project ID is not within the valid project ID
@@ -60,7 +61,7 @@
     |_____________________________|_____________________________|

 SEE ALSO
-     setsid(2), project(4), attributes(5)
+     setsid(2), project(4), attributes(5), privileges(5)

 modified 17 Apr 2002                                  settaskid(2)
```

```
--- setuid.2    Thu Oct  3 17:04:38 2002
+++ setuid.2.new        Tue Oct  8 17:09:42 2002
@@ -41,20 +41,22 @@
      file, the effective user ID, saved user ID, effective group ID,
      and saved group ID are not changed.

-     If the effective user ID of the process calling setuid() is the
-     super-user, the real, effective, and saved user IDs are set to
-     the uid argument.
-
-     If the effective user ID of the calling process is not the
-     super-user, but uid is either the real user ID or the saved user
-     ID of the calling process, the effective user ID is set to uid.
-
-     If the effective user ID of the process calling setgid() is the
-     super-user, the real, effective, and saved group IDs are set to
-     the gid argument.
+     If the {PRIV_PROC_SETID} privilege is asserted in the effective
+     set of the process calling setuid(), the real, effective, and
+     saved user IDs are set to the uid argument.  If the uid argument
+     is 0 and none of the saved, effective or real uid is 0, additional
+     restrictions apply (see privileges(5)).
+
+     If the {PRIV_PROC_SETID} privilege is not asserted in the
+     effective set, but uid is either the real user ID or the saved user ID
+     of the calling process, the effective user ID is set to uid.
+
+     If the {PRIV_PROC_SETID} privilege is asserted in the effective
+     set of the process calling setgid, the real, effective, and saved
+     group IDs are set to the gid argument.

-     If the effective user ID of the calling process is not the
-     super-user, but gid is either the real group ID or the saved
```

```
+     If the {PRIV_PROC_SETID} privilege is not asserted in the
+     effective set, but gid is either the real group ID or the saved
      group ID of the calling process, the effective group ID is set to
      gid.

@@ -68,12 +70,14 @@
      EINVAL
            The value of uid or gid is out of range.

-     EPERM For setuid() and seteuid() the effective user of the cal-
-           ling process is not super-user, and the uid argument does
-           not match either the real or saved user IDs.  For setgid()
-           and setegid() the effective user of the calling process is
-           not the super-user, and the gid argument does not match
-           either the real or saved group IDs.
+     EPERM For setuid() and seteuid() the {PRIV_PROC_SETID} privilege
+           is not asserted in the effective set of the calling process,
+           and the uid argument does not match either the real or saved
+           user IDs or an attempt is made to change to uid 0 and non of
+           the existing uids is 0; in that case additional privileges
+           are required. For setgid() and setegid() the {PRIV_PROC_SETID}
+           privilege is not asserted in the effective set, and the gid
+           argument does not match either the real or saved group IDs.

 ATTRIBUTES
      See attributes(5) for descriptions of the following attributes:
@@ -87,7 +91,7 @@

 SEE ALSO
      intro(2), exec(2), getgroups(2), getuid(2), stat.h(3HEAD), attri-
-     butes(5), standards(5)
+     butes(5), privileges(5), standards(5)

 modified 28 Dec 1996                                        setuid(2)
```

```
--- shmctl.2    Fri Jan 24 12:12:24 2003
+++ shmctl.2.new        Fri Jan 24 12:36:24 2003
@@ -98,13 +98,14 @@
            be stored in the structure pointed to by buf.

      EPERM The cmd argument is equal to IPC_RMID or IPC_SET and the
-           effective user ID of the calling process is not super-user
-           and it is not equal to the value of shm_perm.cuid or
-           shm_perm.uid in the data structure associated with shmid.
```

```
+           effective user ID of the calling process is not equal to the
+           value of shm_perm.cuid or shm_perm.uid in the data structure
+           associated with shmid and {PRIV_IPC_OWNER} is not asserted
+           in the effective set of the calling process.

-     EPERM The cmd argument is equal to SHM_LOCK or SHM_UNLOCK and the
-           effective user ID of the calling process is not equal to
-           that of super-user.
+     EPERM The cmd argument is equal to SHM_LOCK or SHM_UNLOCK and
+           {PRIV_PROC_LOCK_MEMORY} is not asserted in the effective
+           set of the calling process.

 ATTRIBUTES
      See attributes(5) for descriptions of the following attributes:
@@ -115,8 +116,8 @@
     |_____________________________|_____________________________|

 SEE ALSO
-     ipcs(1), intro(2), shmget(2), shmop(2), attributes(5), stan-
-     dards(5)
+     ipcs(1), intro(2), shmget(2), shmop(2), attributes(5),
+     privileges(5), standards(5)

 modified 29 Jul 1991                                    shmctl(2)
```

```
--- sigsend.2   Fri Jan 24 12:17:27 2003
+++ sigsend.2.new       Fri Jan 24 12:36:30 2003
@@ -109,11 +109,14 @@
           The sig argument is  SIGKILL, idtype is  P_PID and id is  1
           (proc1).

-     EPERM The effective user of the calling process is not superuser
-           and its real or effective user ID does not match the real
-           or effective user ID of the receiving process, and the cal-
-           ling process is not sending SIGCONT to a process that
-           shares the same session.
+     EPERM The effective user of the calling process
+           does not match the real or saved user and the calling
+           process does not have the {PRIV_PROC_OWNER} privilege
+           asserted in the effective set, and the calling process is
+           not sending SIGCONT to a process that shares the same
+           session ID or the calling process does not have the
+           {PRIV_PROC_SESSION} privilege asserted and is trying to send
+           a signal to a process with a different session ID.
```

```
      ESRCH No process can be found corresponding to that specified by
            id and idtype.
@@ -125,7 +128,7 @@

 SEE ALSO
      kill(1), getpid(2), kill(2), priocntl(2), signal(3C),
-     signal.h(3HEAD)
+     signal.h(3HEAD), privileges(5)

 modified 10 Dec 1999                                    sigsend(2)
```

```
--- stime.2     Fri Jan 24 12:22:23 2003
+++ stime.2.new Fri Jan 24 12:36:34 2003
@@ -23,11 +23,11 @@
      EINVAL
            The tp argument points to an invalid (negative) time value.

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM {PRIV_SYS_TIME} is not asserted in the effective set of
+           the calling process.

 SEE ALSO
-     time(2)
+     time(2), privileges(5)

 modified 5 Jul 1990                                      stime(2)
```

```
--- swapctl.2   Fri Jan 24 12:23:04 2003
+++ swapctl.2.new       Fri Jan 24 12:36:42 2003
@@ -119,8 +119,8 @@
            Pathname provided to SC_ADD or SC_REMOVE contained a com-
            ponent in the path prefix that was not a directory.

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM {PRIV_SYS_MOUNT} was not asserted in the effective set
+           of the calling process.

      EROFS The pathname specified for SC_ADD is a read-only file sys-
            tem.
```

```
--- sysinfo.2   Fri Jan 24 12:24:20 2003
+++ sysinfo.2.new       Fri Jan 24 12:36:45 2003
@@ -41,11 +41,12 @@
            up to 256 bytes in length (plus the terminating null).

      SI_SET_HOSTNAME
-           Copy the null-terminated contents of the array pointed to
-           by buf into the string maintained by the kernel whose value
+           Copy the null-terminated contents of the array pointed to by
+           buf into the string maintained by the kernel whose value
            will be returned by succeeding calls to sysinfo() with the
-           command SI_HOSTNAME. This command requires that the
-           effective-user-ID be superuser.
+           command SI_HOSTNAME. This command requires that
+           {PRIV_SYS_CONFIG} is asserted in the effective set of the
+           calling process.

      SI_RELEASE
            Copy into the array pointed to by buf the string that would
@@ -121,8 +122,9 @@
      SI_SET_SRPC_DOMAIN
            Set the string to be returned by sysinfo() with the
            SI_SRPC_DOMAIN command to the value contained in the array
-           pointed to by buf. This command requires that the
-           effective-user-ID be superuser.
+           pointed to by buf. This command requires that
+           {PRIV_SYS_CONFIG} is asserted in the effective set of the
+           calling process.

      SI_DHCP_CACHE
            Copy into the array pointed to by buf an ASCII string con-
@@ -154,7 +156,8 @@
            the data for a SET command exceeds the limits established
            by the implementation.

-     EPERM The effective user of the calling process is not superuser.
+     EPERM {PRIV_SYS_CONFIG} was not asserted in the effective set of
+           the calling process.

 USAGE
      In many cases there is no corresponding programming interface to
@@ -169,7 +172,8 @@

 SEE ALSO
```

```
      boot(1M), dhcpagent(1M), uname(2), gethostid(3C),
-     gethostname(3C), sysconf(3C), isalist(5), standards(5)
+     gethostname(3C), sysconf(3C), isalist(5), privileges(5),
+     standards(5)

 modified 30 May 2002                                    sysinfo(2)
```

```
--- uadmin.2    Fri Jan 24 12:29:51 2003
+++ uadmin.2.new        Fri Jan 24 12:36:52 2003
@@ -92,8 +92,8 @@
 ERRORS
      The uadmin() function will fail if:

-     EPERM The effective user of the calling process is not super-
-           user.
+     EPERM {PRIV_SYS_CONFIG} is not asserted in the effective set of
+           the calling process.

      ENOMEM
            Suspend/resume ran out of physical memory.
@@ -110,7 +110,7 @@
      EBUSY Suspend already in progress.

 SEE ALSO
-     dumpadm(1M), kernel(1M), uadmin(1M)
+     dumpadm(1M), kernel(1M), uadmin(1M), privileges(5)

 modified 12 Jan 2001                                     uadmin(2)
```

```
--- ulimit.2    Fri Jan 24 12:31:17 2003
+++ ulimit.2.new        Fri Jan 24 12:36:55 2003
@@ -51,8 +51,8 @@
      EINVAL
            The cmd argument is not valid.

-     EPERM A process not having appropriate privileges attempts to
-           increase its file size limit.
+     EPERM A process which has not asserted {PRIV_SYS_RESOURCE} in
+           its effective set is trying to increase its file size limit.

 USAGE
      Since all return values are permissible in a successful situa-
```

```
@@ -73,7 +73,8 @@
     |_____________________________________|_____________________________________|

 SEE ALSO
-     brk(2), getrlimit(2), write(2), attributes(5), standards(5)
+     brk(2), getrlimit(2), write(2), attributes(5), privileges(5),
+     standards(5)

 modified 18 Apr 1997                                        ulimit(2)
```

```
--- umount.2    Thu Oct  3 16:45:04 2002
+++ umount.2.new        Fri Jan 24 12:33:17 2003
@@ -68,7 +68,8 @@
      ENOTBLK
            The file pointed to by file is not a block special device.

-     EPERM The process's effective user ID is not superuser.
+     EPERM The {PRIV_SYS_MOUNT} privilege is not asserted in the
+           effective set of the calling process.

      EREMOTE
            The file pointed to by file is remote.
@@ -85,7 +86,7 @@
      umount2() function is preferred.

 SEE ALSO
-     mount(2)
+     mount(2), privileges(5)

 modified 9 Jun 1999                                         umount(2)
```

```
--- unlink.2    Fri Jan 24 12:34:01 2003
+++ unlink.2.new        Fri Jan 24 12:36:58 2003
@@ -97,8 +97,8 @@
            provided directory descriptor for unlinkat() is not
            AT_FDCWD or does not reference a directory.

-     EPERM The named file is a directory and the effective user of the
-           calling process is not superuser.
+     EPERM The named file is a directory and {PRIV_SYS_LINKDIR} is
+           not asserted in the effective set of the calling process.
```

```
      EROFS The directory entry to be unlinked is part of a read-only
            file system.
@@ -128,7 +128,7 @@

 SEE ALSO
      rm(1), close(2), link(2), open(2), rmdir(2), remove(3C), attri-
-     butes(5), fsattr(5)
+     butes(5), fsattr(5), privileges(5)

 modified 1 Aug 2002                                        unlink(2)
```

```
--- utime.2     Fri Jan 24 12:39:31 2003
+++ utime.2.new Fri Jan 24 12:40:45 2003
@@ -72,8 +72,10 @@
      ENOTDIR
            A component of the path prefix is not a directory.

-     EPERM The effective user of the calling process is not super-user
-           and not the owner of the file, and times is not NULL.
+     EPERM The effective user of the calling process is not
+           the owner of the file and {PRIV_FILE_OWNER} is not
+           asserted in the effective set of the calling process,
+           and times is not NULL.

      EROFS The file system containing the file is mounted read-only.

@@ -87,7 +89,7 @@
     |_____________________________|_____________________________|

 SEE ALSO
-     stat(2), attributes(5)
+     stat(2), attributes(5), privileges(5)

 modified 28 Dec 1996                                        utime(2)
```

```
--- gettimeofday.3c     Thu Jan 23 22:04:22 2003
+++ gettimeofday.3c.new Fri Jan 24 12:35:30 2003
@@ -32,7 +32,7 @@

      The tzp argument to gettimeofday() and settimeofday() is ignored.

-     Only the superuser can set the time of day.
```

```
+     Only privileged processes can set the time of day.

 RETURN VALUES
      Upon successful completion, 0 is returned.  Otherwise, -1 is
@@ -44,8 +44,8 @@
      EINVAL
            The structure pointed to by tp specifies an invalid time.

-     EPERM A user other than the privileged user attempted to set the
-           time or time zone.
+     EPERM {PRIV_SYS_TIME} was not asserted in the effective set of
+           the calling process.

      The gettimeofday() function will fail for 32-bit interfaces if:

@@ -70,7 +70,8 @@
     |_____________________________|_____________________________|

 SEE ALSO
-     adjtime(2), ctime(3C), TIMEZONE(4), attributes(5), standards(5)
+     adjtime(2), ctime(3C), TIMEZONE(4), attributes(5), privileges(5),
+     standards(5)

 modified 11 Apr 2002                              gettimeofday(3C)
```

```
--- timer_create.3rt    Fri Jan 24 12:26:52 2003
+++ timer_create.3rt.new        Fri Jan 24 12:36:48 2003
@@ -83,8 +83,10 @@
            The timer_create() function is not supported by the system.

      EPERM The specified clock ID, clock_id, is CLOCK_HIGHRES and the
-           effective user of the caller is not superuser.
+           {PRIV_PROC_CLOCK_HIGHRES} is not asserted in the effective
+           set of the calling process.

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

@@ -96,8 +98,8 @@

 SEE ALSO
      exec(2), fork(2), time(2), clock_settime(3RT), signal(3C),
-     timer_delete(3RT), timer_settime(3RT), attributes(5), stan-
```

```
-     dards(5)
+     timer_delete(3RT), timer_settime(3RT), attributes(5),
+     privileges(5), standards(5)

 modified 28 Jun 2002                              timer_create(3RT)
```

```
--- bind.3socket        Thu Jan 23 12:18:29 2003
+++ bind.3socket.new    Fri Jan 24 12:35:10 2003
@@ -25,8 +25,8 @@
      The bind() call will fail if:

       EACCES
-           The requested address is protected and the current user has
-           inadequate permission to access it.
+           The requested address is protected and {PRIV_NET_PRIVADDR}
+           is not asserted in the effective set of the current process.

      EADDRINUSE
           The specified address is already in use.
@@ -85,7 +85,8 @@
     |_____________________________|_____________________________|

 SEE ALSO
-     unlink(2), socket(3SOCKET), attributes(5), socket.h(3HEAD)
+     unlink(2), socket(3SOCKET), attributes(5), privileges(5),
+     socket.h(3HEAD)

 NOTES
```

```
--- bind.3xnet  Thu Jan 23 12:18:37 2003
+++ bind.3xnet.new      Fri Jan 24 12:35:14 2003
@@ -106,8 +106,8 @@
      The bind() function may fail if:

      EACCES
-           The specified address is protected and the current user
-           does not have permission to bind to it.
+           The specified address is protected and {PRIV_NET_PRIVADDR}
+           is not asserted in the effective set of the current process.

      EINVAL
           The address_len argument is not a valid length for the
```

```
@@ -137,7 +137,7 @@

 SEE ALSO
      connect(3XNET), getsockname(3XNET), listen(3XNET), socket(3XNET),
-     attributes(5), standards(5)
+     attributes(5), privileges(5), standards(5)

 modified 10 Jun 2002                                    bind(3XNET)
```

```
--- exec_attr.4 Fri Nov 15 16:02:30 2002
+++ exec_attr.4.new     Fri Jan 24 12:35:23 2003
@@ -35,7 +35,7 @@

      policy
            The policy that is associated with the profile entry. The
-           only valid policy is suser.
+           valid policies are suser and solaris.

      type  The type of object defined in the profile. The only valid
            type is cmd.
@@ -75,6 +75,14 @@
            real and effective GIDs. Setting gid may be more appropri-
            ate than setting guid on privileged shell scripts.

+           privs contains a privilege set which will be added to the
+           inheritable set prior to running the command.
+
+           limitprivs contains a privilege set which will be assigned to
+           the limit set prior to running the command.
+
+           privs and limitprivs are only valid for the "solaris" policy.
+
 EXAMPLES
      Example 1: Using effective user and group IDs

@@ -110,7 +118,7 @@
      auths(1), profiles(1), roles(1), makedbm(1M),
      getauthattr(3SECDB), getauusernam(3BSM), getexecattr(3SECDB),
      getprofattr(3SECDB), getuserattr(3SECDB), kva_match(3SECDB),
-     auth_attr(4), prof_attr(4), user_attr(4)
+     auth_attr(4), prof_attr(4), user_attr(4), privileges(5)

 modified 26 Oct 1999                                    exec_attr(4)
```

```
--- proc.4      Sun Oct  6 16:13:11 2002
+++ proc.4.new  Mon Jan  6 12:35:09 2003
@@ -633,6 +633,27 @@
      groups. pr_ngroups indicates the number of supplementary groups.
      (See also the PCSCRED control operation.)

+priv
+     Contains a description of the privileges associated with the
+     process:
+
+     typedef struct prpriv {
+          uint32_t        pr_nsets;      /* number of privilege set */
+          uint32_t        pr_setsize;    /* size of privilege set */
+          uint32_t        pr_infosize;   /* size of supplementary data */
+          priv_chunk_t    pr_sets[1];    /* array of sets */
+     } prpriv_t;
+
+     The actual dimension of the pr_sets[] field is
+
+          pr_sets[pr_nsets][pr_setsize]
+
+     which is followed by additional information about the process
+     state pr_infosize bytes in size.
+
+     The full size of the structure can be computed using
+     PRIV_PRPRIV_SIZE(prpriv_t *).
+
 sigact
      Contains an array of sigaction structures describing the current
      dispositions of all signals associated with the traced process
@@ -734,8 +755,8 @@
 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 super-user. It has the
-     same semantics as /proc/pid/cwd.
+     of its ancestors executed chroot(2) with appropriate privileges.
+     It has the same semantics as /proc/pid/cwd.

 fd
      A directory containing references to the open files of the pro-
@@ -1510,7 +1531,8 @@

 PCNICE
```

```
      The traced process's nice(2) value is incremented by the amount
-     in the operand long. Only the super-user may better a process's
+     in the operand long. Only a process with the {PRIV_PROC_PRIOCNTL}
+     privilege asserted in its effective set 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.

@@ -1520,20 +1542,44 @@
      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 the super-user may perform this operation; for all
-     others it fails with EPERM.
+     ignored. Only privileged processes may perform this operation; for
+     all others it fails with EPERM.
+
+PCSPRIV
+     Set the target process privilege to the values contained in the
+     prpriv_t operand (see /proc/pid/priv).  The effective, permitted,
+     inheritable and limit sets are all changed; privilege flags can
+     also be set.  The process is made privilege aware (see
+     privileges(5)) unless it can relinquish privilege awareness.
+
+     The limit set of the target process cannot be grown; the other
+     privilege sets must be subsets of the intersection of effective set
+     of the calling process with the new limit set of the target process
+     or subsets of the original values of the sets in the target process.

+     If any of the above restrictions are not met, EPERM is returned.
+     If the structure written is improperly formatted, EINVAL is returned.
+
 PROGRAMMING NOTES
      For security reasons, except for the psinfo, usage, lpsinfo,
      lusage, lwpsinfo, and lwpusage files, which are world-readable,
-     and except for the super-user, an open of a /proc file fails
+     and except for privileged processes, 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 the super-user.
+     only by the appropriate privileged processes.
+
+     A process that has {PRIV_PROC_OWNER} asserted in its effective
+     set can open any file for reading; in order to manipulate or
```

```
+     control a process, the controlling process must have at least
+     as many privileges in its effective set as the target process
+     has in its effective, inheritable and permitted sets.  The
+     limit set of the controlling process must be a superset of the
+     limit set of the target process.  Additional restrictions apply
+     if any of the uids of the target process are 0 (see privileges(5)).

-     Even if held by the super-user, an open process or lwp file
+     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
@@ -1603,6 +1649,9 @@
      /proc/pid/cred
            process credentials

+     /proc/pid/priv
+           process privileges
+
      /proc/pid/sigact
            process signal actions

@@ -1684,7 +1733,7 @@
      sigaction(2), sigaltstack(2), vfork(2), wait(2), write(2), wri-
      tev(2), _stack_grow(3C), readdir(3C), siginfo.h(3HEAD),
      signal.h(3HEAD), types32.h(3HEAD), ucontext.h(3HEAD), lfcom-
-     pile(5)
+     pile(5), privileges(5)

 DIAGNOSTICS
      Errors that can occur in addition to the errors normally associ-
@@ -1706,9 +1755,26 @@
            was applied to a process that was not fully stopped or that
            already had an agent lwp.

-     EPERM Someone other than the super-user issued the PCSCRED opera-
-           tion; someone other than the super-user attempted to better
-           a process's priority by applying PCNICE.
+     EPERM The process that issued the PCSCRED operation did not
+           have the {PRIV_PROC_SETID} privilege asserted in its
+           effective set.  The process that issued the PCNICE operation
+           did not have the {PRIV_PROC_PRIOCNTL} in its effective set.
+
+           An attempt was made to control a process of which the
+           E, P and I privilege sets were not a subset of the
```

```
+           effective set of the controlling process or the limit
+           set of the controlling process is not a superset of limit
+           set of the controlled process.
+           Any of the uids of the target process are 0 or an attempt
+           was made to change any of the uids to 0 using PCSCRED and
+           the security policy imposed additional restrictions (See
+           privileges(5)).
+
+     EACCES
+           An attempt was made to examine a process that ran under
+           a different uid than the controlling process and
+           {PRIV_PROC_OWNER} was not asserted in the effective set.
+

      ENOSYS
            An attempt was made to perform an unsupported operation
@@ -1765,10 +1831,10 @@
      Because the old ioctl(2)-based version of /proc is currently sup-
      ported 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 the super-user. Support for the old
-     ioctl(2)-based version of /proc will be dropped in a future
+     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 an appropriately 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.
```

```
--- prof_attr.4 Fri Nov 15 15:59:40 2002
+++ prof_attr.4.new     Fri Jan 24 12:35:59 2003
@@ -57,6 +57,10 @@
            profs specifies a comma-separated list of profile names
            chosen from those names defined in the prof_attr database.

+           privs specifies a privilege set as described in
+           priv_str_to_set(3C).  Users with the specific profile
+           can obtain any listed privilege using pfexec(1).
+
 EXAMPLES
      Example 1: Allowing execution of all commands
```

```
@@ -106,7 +110,8 @@

 SEE ALSO
      auths(1), profiles(1), getauthattr(3SECDB), getprofattr(3SECDB),
-     getuserattr(3SECDB), auth_attr(4), exec_attr(4), user_attr(4)
+     getuserattr(3SECDB), priv_str_to_set(3C), auth_attr(4), exec_attr(4),
+     user_attr(4), privileges(5)

 modified 11 Feb 2000                                    prof_attr(4)
```

```
--- user_attr.4 Thu Oct  3 17:13:49 2002
+++ user_attr.4.new     Fri Nov 15 15:53:19 2002
@@ -73,6 +73,20 @@
                   place the user in at login time. For more informa-
                   tion, see getdefaultproj(3PROJECT).

+           defaultpriv
+                 The default set of privileges assigned to a user's
+                 inheritable set on login.
+
+           limitpriv
+                 The maximum set of privileges a user or any process
+                 started by the user whether through su(1m) or any
+                 other means can obtain. The system administrator
+                 must take extreme care when removing privileges
+                 from the limit set; removing any basic privilege
+                 has the ability of crippling all applications;
+                 removing any other privilege can cause many or
+                 all applications requiring privileges to malfunction.
+
 EXAMPLES
      Example 1: Assigning a Profile to Root
```

```
--- drv_priv.9f Tue Oct  8 17:17:39 2002
+++ drv_priv.9f.new     Sun Oct 20 16:57:31 2002
@@ -19,12 +19,14 @@
 DESCRIPTION
      drv_priv() provides a general interface to the system privilege
      policy.  It determines whether the credentials supplied by the
-     user credential structure pointed to by  cr identify a privileged
-     process.  This function should only be used when file access
-     modes and special minor device numbers are insufficient to pro-
```

```
-     vide protection for the requested driver function.  It is
-     intended to replace all calls to  suser() and any explicit checks
-     for effective user ID = 0 in driver code.
+     user credential structure pointed to by cr identify a process
+     that has the {PRIV_SYS_DEVICES} privilege asserted in its
+     effective set.  This function should only be used when file access
+     modes, special minor device numbers and the device policy (see
+     privileges(5), add_drv(1m)) are insufficient to provide protection
+     for the requested driver function.  It is intended to replace all
+     calls to suser() and any explicit checks for effective user ID = 0
+     in driver code.

 RETURN VALUES
      This routine returns  0 if it succeeds, EPERM if it fails.
@@ -33,6 +35,8 @@
      drv_priv() can be called from user or interrupt context.

 SEE ALSO
+     add_drv(1m), update_drv(1m), privileges(5)
+
      Writing Device Drivers

 modified 11 Apr 1991                                    drv_priv(9F)
```

```
--- ld.so.1.1   Thu Oct 24 17:06:06 2002
+++ ld.so.1.1.new       Thu Oct 24 17:08:03 2002
@@ -281,11 +281,8 @@
      of their dependencies and runpaths to prevent malicious depen-
      dency substitution or symbol interposition.

-     The runtime linker categorizes a process as secure if the user is
-     not a super-user, and either the real user and effective user
-     identifiers are not equal, or the real group and effective group
-     identifiers are not equal. See getuid(2), geteuid(2), getgid(2),
-     and getegid(2).
+     The runtime linker categorizes a process as secure if the
+     issetugid(2) system call returns true for the process.

      The default trusted directory known to the runtime linker is
      /usr/lib/secure for 32-bit objects or /usr/lib/secure/64 for 64-
@@ -387,9 +384,9 @@

 SEE ALSO
      crle(1), gprof(1), ld(1), ldd(1), exec(2), getegid(2),
-     geteuid(2), getuid(2), kill(2), mmap(2), profil(2), dladdr(3DL),
```

```
-     dlclose(3DL), dldump(3DL), dlerror(3DL), dlopen(3DL), dlsym(3DL),
-     proc(4), attributes(5)
+     geteuid(2), getuid(2), issetugid(2), kill(2), mmap(2), profil(2),
+     dladdr(3DL), dlclose(3DL), dldump(3DL), dlerror(3DL), dlopen(3DL),
+     dlsym(3DL), proc(4), attributes(5)

      Linker and Libraries Guide
```