# Controlled Privilege Escalation in Linux/UNIX Environments

by David Bank CCNA, CNE, CCSE, CNA
v1.50 (2007-Jul-18)


## Why is there a need for controlled privilege escalation?

The Linux/UNIX environment tends to classify users in two ways:

* *root* - infinite, cosmic power
* Everyone else - very little control of anything they don't own

Many system tasks, such as starting a process that binds to a port number of 1024 or less, or stopping a process running under a different UID, require *root* privilege. Typically, these tasks can only be done by someone able to login as *root*, or someone who knows the *root* password and invokes the **su** utility to escalate their privilege.

The trouble with those methods is that the person performing those tasks must know the *root* password. The escalation of their privilege level is complete and unrestricted, which means that they can do anything, not just specific tasks. It's also impossible to effectively log or audit actions taken.

A better solution is a privilege escalation technique that controls both the escalation and to what commands/programs it applies. An excellent tool to do this is GratiSoft's **sudo**[1].

Using **sudo**, a system administrator could allow certain users to invoke processes that require privilege - for example, a trusted user might be allowed to reconfigure and restart an Apache webserver, or a "guest" admin might be empowered to restart the entire system. But neither one could perform the task allowed the other. Perhaps most importantly, such operations can be securely logged.

### **sudo** *vs.* Solaris RBAC

In Solaris 8, Sun introduced Role-Based Access Control, and it provides much of the same functionality as **sudo**[2]. So, if you have Solaris in your environment, which tool should you use?

The chief advantages of Solaris RBAC are tight integration with NIS/NIS+, and arguably better support for Solaris Zones and Process Contracts (**sudo** added support for Solaris Process Contracts in v1.6.9). However, Solaris RBAC tends to be more-complex to configure and manage than **sudo**.

What is the answer to the question? In the heterogeneous environment, **sudo** shines; in a Solaris-dominant environment, however, RBAC may be as good as or even better a secure privilege escalation tool as **sudo**. In any environment having a substantial Solaris host population, RBAC should be seriously considered.

## A sudo How-To

**sudo** is a Free/Open-Source Software (FOSS) package. Many modern Linux/UNIX distributions include a pre-built **sudo** or have sites where you can download installable packages. **sudo** is a package that I prefer to build from source, so this paper covers those steps.

### Why Not Packages?

Common practice in the Linux/UNIX world is to install pre-built packages, usually provided by the platform vendor or distribution maintainer, rather than to build from source. Where a package is not part of the "official" offerring, it is often available from a 3rd party (*e.g.* the package author, a site like SunFreeware[3], or other independent repository). Practically every modern Linux/UNIX distributions include a pre-built **sudo**.

Why, then, does this paper take the build from source approach?

When using a pre-built package, one must accept whatever compile-time configuration decisions were made by the package creator. These parameters may or may not be adjustable at run-time, and compile-time options selected by the package creator may or may not be appropriate to a specific environment. Building from source allows tailoring the software. As with any other system administration decision, weigh the factors in the environment and choose a course of action.

If you decide to use a pre-built package, skip past the Compilation and Installation sections - start reading at the Configuration section. You should follow the package installation documentation, and reference this paper for configuration tips. The current version, as of this writing, is **sudo v1.6.9**.

### Standards and Assumptions

This paper's assumptions, for those wishing to build from source, include the use of a modern Linux or UNIX (or UNIX-like) operating system, the ability to run shell scripts, compile C code (with GNU **gcc** or an equivalent), copy and/or link files, and set file modes and ownership. While some of these tasks require *root* privilege, you should only invoke that when you specifically need it (for example, most compilation steps can probably be accomplished as an unprivileged user).

Every system admin has their own way of doing things, and their own sense of where to put files. This paper is written from my perspective on these issues, which may be different from the maintainers of the pre-built packages. Select whatever file location scheme for this tool that is appropriate to your environment, and if necessary translate my paths and locations into the scheme you select.

My personal environment is usually either SLES v9 (or later) with **gcc v3.3.3** (or later), RHEL v4 (or later) or **gcc v3.4.6** (or later) or Sun Solaris v8 (or later) with **gcc v3.3.2** (or later). Other tools

common to both include **make v3.8x** and **nano v2.0.x**. I tend to use the stock development tools such as **ld**, **ar**, **yacc** and **lex**. I also tend to create a sub-directory structure specifically for building add-on tools, generally */work*. Make sure the partition where this is located has adequate room.

I usually install add-on tools, like **sudo**, in a sub-directory off of */opt*, and then use symbolic links in */usr/bin* or wherever else might be needed. This allows me to control access better than if everything is dumped in */usr/local*. I prefer symbolic links because I frequently make */opt* its own partition, and I can "snap-in" a newer version of a tool with a few **mv** commands, since the link is merely a pointer to a path and file name (a hard link is another inode entry and can't cross partition boundaries).

What About Deployment Tool/Technique X?

Administrators accustomed to working in a homogeneous environment may wonder why this paper does not mention or advocate the use of specific tools for deployment beyond a single machine, or uses techniques that might seem problematic when viewed from a specific environment perspective. A reader might find themselves thinking “Why not just use <*insert tool name here*>?” or “That suggestion doesn't make sense in <*insert specific environment name here*>!”

This paper is deliberately written for a generic audience, where a reader may be interested in applying the information presented in diverse environments, perhaps other than a typical Linux distribution or common UNIX variant. As a consequence, it offers ideas culled from a number of environments. The reader is encouraged to consider those ideas, techniques and tools that are applicable to their situation, and to ignore those that are not.

Download and unpack the source

However might be appropriate for your environment, download the latest **sudo** source package. I download mine into */work/sudo*:

```
me@host /work 2 $ dir
drwx------ 2 me wheel    512 Jul 17 23:32 sudo/
me@host /work 3 $ cd sudo
me@host /work/sudo 4 $ dir
-rw------- 1 me wheel 557692 Jul 17 19:47 sudo-1.6.9.tar.gz
```

If your **tar** program includes the ability to decompress **gzip**-ed files, then you can use it directly, or you can call **gzip** as a separate step on the way to un-**tar**ing the file. Here, I'm showing the latter method, and writing three commands separated by semi-colons:

```
me@host /work/sudo 5 $ gzip -dv *.gz ; tar -xvf*.tar ; gzip -v9 *.tar
```

The commands will decompress the file, unpack the tarfile, then recompress the tarfile using the best compression offered by **gzip**. No sense wasting disk space leaving the uncompressed tarfile around. If you have it handy, substitute **bzip2** for **gzip** – the compression results tend to be better[4].

The source files now reside in a directory named very similar to the file from which everything was extracted. If you, like me, use the autocompletion capability of your shell, this can be annoying. So I usually change the directory name to something short:

```
me@host /work/sudo 9 $ mv sudo-1.6.9 V1.6.9
me@host /work/sudo 10 $ dir
-rw------- 1 me wheel   557692 Jul 17 19:47 sudo-1.6.9.tar.gz
-rwx------ 1 me wheel      512 Jul 17 19:47 V1.6.9/
```

Now autocompletion won't beep at me. I can get ready for the next step with:

```
me@host /work/sudo 11 $ cd V*
me@host /work/sudo/V1.6.9 12 $
```

Compilation

Helpfully, **sudo** uses the ubiquitous GNU *autoconf* configuration tool. The script **configure** will automatically examine your system, check dependencies, and prepare **sudo** to be compiled with a built-in set of defaults.

However, an advantage of building from source is the opportunity to tweak things a bit. For example, the default location of the **sudo** binary is */usr/local*, and as I noted above, I prefer */opt/sudo*. You can see all the possible configuration options, and their defaults, with the "--help" parameter:

```
me@host /work/sudo/V1.6.9 13 $ ./configure --help
```

This will show you a lot of information, but won't actually configure or compile anything. What you might change from the defaults is largely dependent on your preferences and standards in your environment.

I like to change a number of the defaults, which I can do from the command-line when I invoke the **configure** script:

```
me@host /work/sudo/V1.6.9 14 $ ./configure --prefix=/opt/sudo \
      --sysconfdir=/opt/sudo/conf --with-umask=077 \
      --with-mailto=< my standard address > --disable-root-sudo
      --with-editor=/usr/bin/rnano:/sbin/vi --with-env-editor \
      --with_logging=syslog --with-logfac=local0 \
      --with-goodpri=notice --with-badpri=alert --with-ignore-dot
```

Why these particular options and settings? Glad you asked that:

**`--prefix=/opt/sudo`**
See my previous discussion about file locations.

**`--sysconfdir=/opt/sudo/conf`**
Without this, the location of the *sudoers* configuration file would be in */opt/sudo/etc*. As time-honored as */etc* is, I prefer the more-descriptive */conf*.

**`--umask=077`**
Normally, the environment spawned by **sudo** can inherit the *UMASK* of the user environment in existence at the time **sudo** was invoked. And that *UMASK* could be anything. I prefer to force the *UMASK* of all **sudo**-spawned environments to the very restrictive *077*.

**`--with-mailto=< my standard address >`**
In its configuration, **sudo** can be set up to send an E-Mail when certain suspicious events occur, such as an unauthorized account trying to use **sudo**, or an authorized account trying to access an unauthorized command. If you use this option, substitute an appropriate E-Mail address for < *my standard address* >, and be sure to check that it routes properly.

**`--disable-root-sudo`**
I have a hard time imagining why *root* would need to run **sudo** in most environments. My general goal with **sudo** is to get as close as possible to obviating the *root* account.

**`--with-editor=/usr/bin/rnano:/sbin/vi`**
If you don't specify editors, then on most platforms you get stuck using **vi**. Yes, **vi** is the One True Editor, but it's also a pain. I like **nano** as a nice middle ground between the rich world of **emacs** and the terse obtuseness of **vi**. Helpfully, **nano** has a *restricted* invocation mode (**rnano**) that overrides the configuration files, disables suspension or shell escape, and sharply limits file operations[5]. **sudo** will use the first editor listed that exists, but this can be modified with the next option.

**`--with-env-editor`**
This option allows **sudo** to honor the *$EDITOR* and *$VISUAL* environment variables, subject to further configuration in the *sudoers* file. It is important to only use this option in conjunction with the previous one, as allowing **sudo** to honor any value in those variables can create a security hole.

**--with-logging=*syslog***
Using this option instructs **sudo** to log its events to the standard syslog interface. The next three options further configure this.

**--with-logfac=*local0***
With this, **sudo** will use *Facility* **LOCAL0** when writing to **syslogd**. Linux admins may prefer **AUTHPRIV**, although most Linux-based OpenSSH configurations for **sshd** log to **AUTHPRIV** and you may not want to mix them.

**--with-goodpri=*notice***
This option tells **sudo** to use *Priority* **NOTICE** when writing "routine" messages to **syslogd**. An example would be a notification that an authorized user has run an authorized command. Any valid *Priority* may be used.

**--with-badpri=*alert***
Opposite the previous option, this option instructs **sudo** to use *Priority* **ALERT** when writing to **syslogd** for "abnormal" events, such as an unauthorized user trying to use **sudo**, or an authorized user trying to run a command they are not allowed. Any valid *Priority* may be used.

**--with-ignore-dot**
A very important option, this tells **sudo** to ignore any **.** in *$PATH* variables. There shouldn't be any of those in *$PATH* variables, but if there are, **sudo** will ignore them.

At this point, the **configure** script should have been run, with any options you wanted. For the purposes of this paper, the assumption is made that it ran to completion without any errors. If it had problems, you need to fix them and re-run **configure**. Among other things, **configure** will generate *Makefile* in the same directory, and this is used for the rest of the process.

You're now ready to compile **sudo**:

```
me@host /work/sudo/V1.6.9 21 $ make
```

This starts the compilation process. Troubleshooting compilation issues is outside the scope of this paper, and again the assumption is made that **sudo** has been successfully compiled and you are ready to proceed to the next stage.

Installation and Configuration

If you've gotten to this point without any problems, the rest of the process should be equally trouble-free. To actually install the program files, you need to have *root* privilege because, among other

things, you'll be setting SUID bits in file modes, and usually only *root* can do that. So, invoke **su**:

```
me@host /work/sudo/V1.6.9 25 $ su
```

Now that you are privileged, install the program:

```
# make install
```

Again, troubleshooting installation error messages are outside the scope of this paper. You'll just have to look at the message(s) and figure it out for yourself. If there are no errors, it's a good idea to check the results, which should look like this:

```
# ls -la /opt
drwxr-xr-x 7 root other   512 Jul 17 21:04 sudo/
# ls -la /opt/sudo/bin
---s--x--x 2 root root 110488 Jul 17 21:04 sudo*
---s--x--x 2 root root 110488 Jul 17 21:04 sudoedit*
# ls -la /opt/sudo/sbin
---x--x--x 1 root root  75816 Jul 17 21:04 visudo*
# ls -la /opt/sudo/conf
-r--r----- 1 root root   4193 Jul 17 21:04 sudoers
# ls -la /opt/sudo/libexec
-rw-r--r-- 1 root other  2768 Jul 17 21:04 sudo_noexec.a
-rw------- 1 root other   804 Jul 17 21:04 sudo_noexec.la
-rwx------ 1 root other  8088 Jul 17 21:04 sudo_noexec.so*
# ls -la /opt/sudo/man
drwxr-xr-x 2 root other   512 Jul 17 21:04 man1m/
drwxr-xr-x 2 root other   512 Jul 17 21:04 man4/
# ls -la /opt/sudo/man/man1m
-r--r--r-- 2 root root  25830 Jul 17 21:04 sudo.1m
-r--r--r-- 2 root root  25830 Jul 17 21:04 sudoedit.1m
-r--r--r-- 1 root root  12099 Jul 17 21:04 visudo.1m
# ls -la /opt/sudo/man4
-r--r--r-- 1 root root  57562 Jul 17 21:04 sudoers.4
```

Your primary concern should be checking the ownership and modes. Most important are the SUID bits on the **sudo** and **sudoedit** executables.

If everything is correct, and you've installed **sudo** into a location not normally in your *$PATH* (like */opt/sudo*), then you'll probably want to link the files to */usr/bin* or wherever is appropriate, like so:

```
# ln -s /opt/sudo/bin/sudo /usr/bin/sudo
# ln -s /opt/sudo/bin/sudoedit /usr/bin/sudoedit
# ln -s /opt/sudo/sbin/visudo /usr/sbin/visudo
# ln -s /opt/sudo/man/man1m/sudo.1m /usr/share/man/man1m/sudo.1m
# ln -s /opt/sudo/man/man1m/sudoedit.1m /usr/share/man/man1m/sudoedit.1m
# ln -s /opt/sudo/man/man1m/visudo.1m /usr/share/man/man1m/visudo.1m
# ln -s /opt/sudo/man/man4/sudoers.4 /usr/share/man/man4/sudoers.4
```

The appropriate links/paths for your environment may be different.

Finally, you want to edit to the *sudoers* configuration file. The file installed by default is quite sparse, and also not well documented. There are probably a number of things you want to do, but at the very least, give your usual unprivileged account the ability to edit *sudoers*. You're still privileged, so simply:

```
# nano /opt/sudo/conf/sudoers
```

The **Reference** section has a sample *sudoers* file that is a little better documented.

Exit the *root* shell:

```
# exit
me@host /work/sudo/V1.6.9 26 $ which sudo
/usr/bin/sudo
```

Congratulations, **sudo** is now installed.

## Tips for using sudo effectively

1) Keep **sudo** updated: Common advice for sysadmins is to keep your systems patched and your add-on programs updated. However, that advice is especially applicable to a tool like **sudo**. Bookmark the **sudo** website and check it regularly for updates and bug notifications, or subscribe to the **sudo** announcements mailing list[6]. If you use a pre-built package, then still check the **sudo** site for updates so you know when to check with the package maintainer. The point is not to be passive – proactively keep up with changes to **sudo**.

2) Don't be deceived by simplicity: **sudo** is powerful, but when you get down to it, it's a fairly simple tool. Don't let its simplicity deceive you into being careless in its configuration.

3) Locate **sudo** appropriately: If you put **sudo** somewhere other than */usr/local*, in a subdirectory structure to which you can control access (I'll stick with my example of */opt/sudo*), then you can easily restrict filesystem access to **sudo** to those accounts that can legitimately use it. For example, let's say that you create the group *cansudo* in */etc/group*, and put everyone who can legitimately use **sudo** in that group (this is independent of whether or not you leverage the group in your *sudoers* file). You can set the ownership of */opt/sudo* to **root:cansudo** with mode *750*. Anyone not in *cansudo* is now blocked from the entire subdirectory structure. They can't even read the *man* pages. A side effect is that even if someone is listed in the *sudoers* file as a legitimate user of **sudo**, they won't be able to actually use it without being in *cansudo*.[7]

4) Leverage the simplicity: The heart and soul of **sudo** is the *sudoers* file. If you take the time to write the file well, then a single "standard" file can be distributed across multiple systems, even multiple OSes/architectures. Subsequently, changes across multiple hosts can be accomplished with tools like **sed**, the standard content of the *sudoers* file enabling you to script changes.

5) Use filesystem options: If your environment is such that you can make use of the *nosuid* option when mounting filesystems, then putting **sudo** in */opt* (or under */usr*) might not be the best idea. Consider having a partition specifically for SUID programs like **sudo**, and mounting */opt* (which I tend to use for other tools as well) with the *nosuid* option. This helps segregate programs that use the security-problematic SUID bit into their own filesystem, and allows you to mount other filesystems without SUID support. If you are able to make your filesystem environment this granular, then also consider mounting the partition where **sudo** is kept as read-only.

6) Use extra caution with scripts and **sudo**: Be very careful, if not downright paranoid, about using **sudo** in conjunction with shell scripts, such as start/stop scripts commonly found in */etc/init.d* or similar locations. Insure that such scripts are owned by *root* and are mode *755*, *750*, or more restrictive. The danger here is that if an otherwise unprivileged user can edit a script available for execution with **sudo**, then they can execute any command at the privilege level afforded by **sudo**. For example, consider an environment with an application, located in */opt/someapp*, where the app's daemon is controlled with the script */opt/someapp/bin/control.sh*, which is available to be run as *root* using **sudo**. User *bob* is the app "owner" with unprivileged shell access. If *bob* can edit the control script, then he can easily insert a statement into the script, and it will be executed as *root*. One pernicious act might be to code a small **sed** statement that changed Bob's UID to **0**, making the *bob* account a *root*-equivalent account.

7) *sudoers* can override compile-time options: Be aware that entries in *sudoers* can override any compile-time options you set, including the ones presented in this paper[8]. If you compile from source, then you are, of course, free to edit the source code so that the configuration file parser ignores entries you don't want overridden. More practically, this is simply something to keep in mind when constructing your *sudoers* file.

> Important!
>
> Never configure **sudo** to allow execution, as *root*, of a script (or other program file) that is editable/writable by an unprivileged user. Additionally, it is dangerous to allow execution of shell-escape-enabled programs, such as editors, through **sudo**. Only do so when necessary, and then only with strict limits (preferably further enforced by the program).

## Helpful reference materials

The sample *sudoers* file included in with the program is a bare-bones framework, and not all that well documented. Here is a documented template that is most-apropos for Solaris, but is easily adapted for other platforms (modify the various **`Cmnd_Alias`** entries):

```
##############################################################
# /opt/sudo/conf/sudoers:  sudo permissions file
#
# Change Log:
# Who  When        What
# ---- ----------  ---------------------------------------------
#
#
##############################
## User alias specification ##
##############################
# Defines shorthand names for users and groups using names
#   from /etc/passwd and /etc/group
#   Format: User_Alias <sudo alias> = <group in /etc/groups or user>
##
User_Alias      ADMINS = notroot

##############################
## Runas alias specification ##
##############################
# Defines shorthand names for IDs under which sudo'd programs
#     will run; uses names from /etc/passwd, and default is "root"
#     ("OVERLORD" is therefore redundant, but shown for the sake
#     explicitness)
#   Format: RunAs_Alias <sudo alias> = <user in /etc/passwd>
##
Runas_Alias     OVERLORD = root

##############################
## Host alias specification ##
##############################
# Defines a shorthand for the hosts on which commands can be
#      run. Running commands on other hosts would required rsh
#      functionality, which is often the first thing admins disable
#  Format: Host_Alias <sudo alias> = <host's fully-qualified domain name>
##
Host_Alias      HOST = host.domain.tld
```

```
##############################
## Cmnd alias specification ##
##############################
# Defines shorthand specs for various commands
#  Format: Cmnd_Alias <sudo alias> = <full path specification of program>

# Command necessary to make/restore backups and operate tape drive
Cmnd_Alias      TAPE = /usr/sbin/ufsdump, /usr/sbin/ufsrestore, \
                       /usr/bin/mt

# System shutdown command
Cmnd_Alias      SHUTDOWN = /usr/sbin/shutdown

# Process kill command
Cmnd_Alias      KILL = /usr/bin/kill

# Super-user command
Cmnd_Alias      SU = /usr/bin/su

# Editor for this file
Cmnd_Alias      SUEDIT = /sbin/visudo

# syslogd management
Cmnd_Alias      SYSLOG = /etc/init.d/syslog, /usr/sbin/syslogd

##############################
## Override builtin defaults ##
##############################
## Formats:
# Defaults<:optional list of users>  [whitespace]  <option>
##
# Turn off sudo's "lecture" for selected people
#Defaults:ADMINS    !lecture
Defaults:root      !lecture

# Enable visudo to honor the EDITOR environment variable
Defaults           env_editor

########################
## User specification ##
########################
# Definitions of who can do what
# Format:
#       User_Alias =  ( RunAs_Alias ) Cmnd_Alias [, Cmnd_Alias]
##
# Staff may run  "visudo"
ADMINS          HOST = (OVERLORD) SUEDIT

# Staff may make/restore backups (as root)
ADMINS          HOST = (OVERLORD) TAPE

# Staff may stop/start syslog (as root)
ADMINS          HOST = (OVERLORD) SYSLOG
```

```
# Staff may issue kill or shutdown commands
ADMINS          HOST = (OVERLORD) KILL, SHUTDOWN

# Staff may change any password except for that of root
ADMINS          HOST = /usr/bin/passwd [A-z]*, !/usr/bin/passwd root
####################################
#### End /opt/sudo/conf/sudoers ####
####################################
```

This simple shell script automates running the **configure** script with the options suggested in this paper:

```
#!/usr/bin/sh
#
./configure --prefix=/opt/sudo --sysconfdir=/opt/sudo/conf \
      --with-umask=077 --with-mailto=< my standard address > \
      --disable-root-sudo --with-editor=/usr/bin/rnano:/sbin/vi \
      --with-env-editor --with_logging=syslog --with-logfac=local0 \
      --with-goodpri=notice --with-badpri=alert --with-ignore-dot
```

## Footnotes

[1] `http://www.sudo.ws/sudo`
[2] See Ross Oliver's helpful SysAdmin Magazine article
`http://www.samag.com/documents/s=7667/sam0213c/0213c.htm`
[3] `http://www.sunfreeware.com`
[4] As of Solaris v9, Sun has included **bzip2** as well as **gzip**. Both are typically found in most modern Linux distributions as well.
[5] `http://www.nano-editor.org/`
[6] `http://www.sudo.ws/mailman/listinfo/sudo-announce`
[7] There are, of course, ways around such a limitation. In particular, a skilled attacker with local shell access may be able to gain the necessary group membership. However, this technique can foil remote attacks made without local shell access, and will certainly discourage the casual “noodler”.
[8] I personally disagree with this "feature" - if I'm going to bother setting a compile-time option, it shouldn't be so easily overridden.

## Change Log

| Version | Date | Change |
|---|---|---|
| 1.00 | 2006-Jan-19 | Initial creation |
| 1.05 | 2006-Mar-31 | Fixed typos; minor formatting edits; documented changes |
| 1.10 | 2006-Apr-11 | Added more tips; more minor formatting edits |
| 1.15 | 2006-May-09 | Added more tips; corrected minor typos and formatting |
| 1.20 | 2006-May-11 | Clarified references to the **nano** editor (version/mode) |
| 1.21 | 2006-May-17 | Clarified that the configuration file can override |

| | | |
|---|---|---|
| | | compile-time options |
| 1.22 | 2006-May-23 | Minor formatting changes |
| 1.25 | 2006-Jun-16 | Minor formatting changes |
| 1.30 | 2006-Jun-17 | Reconciled with HTML version; added Footnotes |
| 1.40 | 2007-Jan-09 | Minor updates and formatting changes |
| 1.41 | 2007-Jan-10 | Added Footnote for **nano**; clarified that the sample *sudoers* file is derived from a Solaris environment |
| 1.45 | 2007-Apr-25 | Added information about Solaris RBAC and inclusion of **bzip2** in Solaris 9; minor updates and formatting changes; revised Tip 4 to be more practical; updated **nano** version |
| 1.46 | 2007-May-03 | Minor text and formatting changes |
| 1.47 | 2007-May-05 | Updated development environment info; minor typo fixes |
| 1.48 | 2007-May-13 | Minor text changes |
| 1.50 | 2007-Jul-18 | Updated for **sudo** v1.6.9; fixed copyright statement; updated link to sudo website; added reference to announcements mailing list; re-ordered Footnotes; recomposed section on Solaris RBAC; updated screen capture texts |
| 1.70 | 2007-Dec-15 | Minor text changes; fixed Footnotes; sync version number |

**End of Document**