NAME

sudo, **sudoedit** - execute a command as another user

SYNOPSIS

sudo -h | **-K** | **-k** | **-V**
sudo -v [**-AknS**] [**-a** *auth_type*] [**-g** *group name* | *#gid*] [**-p** *prompt*] [**-u** *user name* | *#uid*]
sudo -l[*l*] [**-AknS**] [**-a** *auth_type*] [**-g** *group name* | *#gid*] [**-p** *prompt*] [**-U** *user name*]
[**-u** *user name* | *#uid*] [*command*]
sudo [**-AbEHnPS**] [**-a** *auth_type*] [**-C** *fd*] [**-c** *class* | -] [**-g** *group name* | *#gid*] [**-p** *prompt*] [**-r** *role*]
[**-t** *type*] [**-u** *user name* | *#uid*] [**VAR**=*value*] **-i** | **-s** [*command*]
sudoedit [**-AnS**] [**-a** *auth_type*] [**-C** *fd*] [**-c** *class* | -] [**-g** *group name* | *#gid*] [**-p** *prompt*]
[**-u** *user name* | *#uid*] file ...

DESCRIPTION

sudo allows a permitted user to execute a *command* as the superuser or another user, as specified by the security policy.

sudo supports a plugin architecture for security policies and input/output logging. Third parties can develop and distribute their own policy and I/O logging plugins to work seamlessly with the **sudo** front end. The default security policy is *sudoers*, which is configured via the file */etc/sudoers*, or via LDAP. See the *PLUGINS* section for more information.

The security policy determines what privileges, if any, a user has to run **sudo**. The policy may require that users authenticate themselves with a password or another authentication mechanism. If authentication is required, **sudo** will exit if the user's password is not entered within a configurable time limit. This limit is policy-specific; the default password prompt timeout for the *sudoers* security policy is 5 minutes.

Security policies may support credential caching to allow the user to run **sudo** again for a period of time without requiring authentication. The *sudoers* policy caches credentials for 5 minutes, unless overridden in sudoers(5). By running **sudo** with the **-v** option, a user can update the cached credentials without running a *command*.

When invoked as **sudoedit**, the **-e** option (described below), is implied.

Security policies may log successful and failed attempts to use **sudo**. If an I/O plugin is configured, the running command's input and output may be logged as well.

The options are as follows:

-A Normally, if **sudo** requires a password, it will read it from the user's terminal. If the **-A** (*askpass*) option is specified, a (possibly graphical) helper program is executed to read the user's password and output the password to the standard output. If the SUDO_ASKPASS environment variable is set, it specifies the path to the helper program. Otherwise, if */etc/sudo.conf* contains a line specifying the askpass program, that value will be used. For example:

```
# Path to askpass helper program
Path askpass /usr/X11R6/bin/ssh-askpass
```

If no askpass program is available, **sudo** will exit with an error.

-a *type* The **-a** (*authentication type*) option causes **sudo** to use the specified authentication type when validating the user, as allowed by */etc/login.conf*. The system administrator may specify a list of sudo-specific authentication methods by adding an "auth-sudo" entry in */etc/login.conf*. This option is only available on systems that support BSD authentication.

-b The **-b** (*background*) option tells **sudo** to run the given command in the background. Note that if you use the **-b** option you cannot use shell job control to manipulate the process. Most interactive commands will fail to work properly in background mode.

-C *fd* Normally, **sudo** will close all open file descriptors other than standard input, standard output and standard error. The **-C** (*close from*) option allows the user to specify a starting point above the standard error (file descriptor three). Values less than three are not permitted. The security policy may restrict the user's ability to use the **-C** option. The *sudoers* policy only permits use of the **-C** option when the administrator has enabled the *closefrom_override* option.

-c *class* The **-c** (*class*) option causes **sudo** to run the specified command with resources limited by the specified login class. The *class* argument can be either a class name as defined in */etc/login.conf*, or a single '-' character. Specifying a *class* of - indicates that the command should be run restricted by the default login capabilities for the user the command is run as. If the *class* argument specifies an existing user class, the command must be run as root, or the **sudo** command must be run from a shell that is already root. This option is only available on systems with BSD login classes.

-E The **-E** (*preserve environment*) option indicates to the security policy that the user wishes to preserve their existing environment variables. The security policy may return an error if the **-E** option is specified and the user does not have permission to preserve the environment.

-e The **-e** (*edit*) option indicates that, instead of running a command, the user wishes to edit one or more files. In lieu of a command, the string "sudoedit" is used when consulting the security policy. If the user is authorized by the policy, the following steps are taken:

1. Temporary copies are made of the files to be edited with the owner set to the invoking user.

2. The editor specified by the policy is run to edit the temporary files. The *sudoers* policy uses the SUDO_EDITOR, VISUAL and EDITOR environment variables (in that order). If none of SUDO_EDITOR, VISUAL or EDITOR are set, the first program listed in the *editor* sudoers(5) option is used.

3. If they have been modified, the temporary files are copied back to their original location and the temporary versions are removed.

If the specified file does not exist, it will be created. Note that unlike most commands run by *sudo*, the editor is run with the invoking user's environment unmodified. If, for some reason, **sudo** is unable to update a file with its edited version, the user will receive a warning and the edited copy will remain in a temporary file.

-g *group* Normally, **sudo** runs a command with the primary group set to the one specified by the password database for the user the command is being run as (by default, root). The **-g** (*group*) option causes **sudo** to run the command with the primary group set to *group* instead. To specify a *gid* instead of a *group name*, use *#gid*. When running commands as a *gid*, many shells require that the '#' be escaped with a backslash ('\'). If no **-u** option is specified, the command will be run as the invoking user (not root). In either case, the primary group will be set to *group*.

-H The **-H** (*HOME*) option requests that the security policy set the HOME environment variable to the home directory of the target user (root by default) as specified by the password database. Depending on the policy, this may be the default behavior.

-h The **-h** (*help*) option causes **sudo** to print a short help message to the standard output and exit.

-i [*command*]

The **-i** (*simulate initial login*) option runs the shell specified by the password database entry of the target user as a login shell. This means that login-specific resource files such as *.profile* or *.login* will be read by the shell. If a command is specified, it is passed to the shell for execution via the shell's **-c** option. If no command is specified, an interactive

shell is executed. **sudo** attempts to change to that user's home directory before running the shell. The security policy shall initialize the environment to a minimal set of variables, similar to what is present when a user logs in. The *Command Environment* section in the sudoers(5) manual documents how the **-i** option affects the environment in which a command is run when the *sudoers* policy is in use.

-K The **-K** (sure *kill*) option is like **-k** except that it removes the user's cached credentials entirely and may not be used in conjunction with a command or other option. This option does not require a password. Not all security policies support credential caching.

-k [*command*]

When used alone, the **-k** (*kill*) option to **sudo** invalidates the user's cached credentials. The next time **sudo** is run a password will be required. This option does not require a password and was added to allow a user to revoke **sudo** permissions from a *.logout* file. Not all security policies support credential caching.

When used in conjunction with a command or an option that may require a password, the **-k** option will cause **sudo** to ignore the user's cached credentials. As a result, **sudo** will prompt for a password (if one is required by the security policy) and will not update the user's cached credentials.

-l[l] [*command*]

If no *command* is specified, the **-l** (*list*) option will list the allowed (and forbidden) commands for the invoking user (or the user specified by the **-U** option) on the current host. If a *command* is specified and is permitted by the security policy, the fully-qualified path to the command is displayed along with any command line arguments. If *command* is specified but not allowed, **sudo** will exit with a status value of 1. If the **-l** option is specified with an *l* argument (i.e. **-ll**), or if **-l** is specified multiple times, a longer list format is used.

-n The **-n** (*non-interactive*) option prevents **sudo** from prompting the user for a password. If a password is required for the command to run, **sudo** will display an error message and exit.

-P The **-P** (*preserve group vector*) option causes **sudo** to preserve the invoking user's group vector unaltered. By default, the *sudoers* policy will initialize the group vector to the list of groups the target user is in. The real and effective group IDs, however, are still set to match the target user.

-p *prompt* The **-p** (*prompt*) option allows you to override the default password prompt and use a custom one. The following percent ('%') escapes are supported by the *sudoers* policy:

%H
expanded to the host name including the domain name (on if the machine's host name is fully qualified or the *fqdn* option is set in sudoers(5))

%h
expanded to the local host name without the domain name

%p
expanded to the name of the user whose password is being requested (respects the *rootpw*, *targetpw*, and *runaspw* flags in sudoers(5))

%U
expanded to the login name of the user the command will be run as (defaults to root unless the **-u** option is also specified)

%u
expanded to the invoking user's login name

%%
two consecutive '%' characters are collapsed into a single '%' character

The prompt specified by the **-p** option will override the system password prompt on systems that support PAM unless the *passprompt_override* flag is disabled in *sudoers*.

-r *role* The **-r** (*role*) option causes the new (SELinux) security context to have the role specified by *role*.

-S The **-S** (*stdin*) option causes **sudo** to read the password from the standard input instead of the terminal device. The password must be followed by a newline character.

-s [*command*]
The **-s** (*shell*) option runs the shell specified by the SHELL environment variable if it is set or the shell as specified in the password database. If a command is specified, it is passed to the shell for execution via the shell's **-c** option. If no command is specified, an interactive shell is executed.

-t *type* The **-t** (*type*) option causes the new (SELinux) security context to have the type specified by *type*. If no type is specified, the default type is derived from the specified role.

-U *user* The **-U** (*other user*) option is used in conjunction with the **-l** option to specify the user

whose privileges should be listed. The security policy may restrict listing other users' privileges. The *sudoers* policy only allows root or a user with the ALL privilege on the current host to use this option.

-u *user* The **-u** (*user*) option causes **sudo** to run the specified command as a user other than *root*. To specify a *uid* instead of a *user name*, *#uid*. When running commands as a *uid*, many shells require that the '#' be escaped with a backslash ('\'). Security policies may restrict *uid*s to those listed in the password database. The *sudoers* policy allows *uid*s that are not in the password database as long as the *targetpw* option is not set. Other security policies may not support this.

-V The **-V** (*version*) option causes **sudo** to print its version string and the version string of the security policy plugin and any I/O plugins. If the invoking user is already root the **-V** option will display the arguments passed to configure when **sudo** was built and plugins may display more verbose information such as default options.

-v When given the **-v** (*validate*) option, **sudo** will update the user's cached credentials, authenticating the user's password if necessary. For the *sudoers* plugin, this extends the **sudo** timeout for another 5 minutes (or whatever the timeout is set to by the security policy) but does not run a command. Not all security policies support cached credentials.

-- The **--** option indicates that **sudo** should stop processing command line arguments.

Environment variables to be set for the command may also be passed on the command line in the form of **VAR**=*value*, e.g. **LD_LIBRARY_PATH**=*/usr/local/pkg/lib*. Variables passed on the command line are subject to the same restrictions as normal environment variables with one important exception. If the *setenv* option is set in *sudoers*, the command to be run has the SETENV tag set or the command matched is ALL, the user may set variables that would otherwise be forbidden. See sudoers(5) for more information.

COMMAND EXECUTION

When **sudo** executes a command, the security policy specifies the execution envionment for the command. Typically, the real and effective uid and gid are set to match those of the target user, as specified in the password database, and the group vector is initialized based on the group database (unless the **-P** option was specified).

The following parameters may be specified by security policy:

- real and effective user ID

- real and effective group ID
- supplementary group IDs
- the environment list
- current working directory
- file creation mode mask (umask)
- SELinux role and type
- Solaris project
- Solaris privileges
- BSD login class
- scheduling priority (aka nice value)

Process model

When **sudo** runs a command, it calls fork(2), sets up the execution environment as described above, and calls the execve system call in the child process. The main **sudo** process waits until the command has completed, then passes the command's exit status to the security policy's close method and exits. If an I/O logging plugin is configured, a new pseudo-terminal ("pty") is created and a second **sudo** process is used to relay job control signals between the user's existing pty and the new pty the command is being run in. This extra process makes it possible to, for example, suspend and resume the command. Without it, the command would be in what POSIX terms an "orphaned process group" and it would not receive any job control signals.

Signal handling

Because the command is run as a child of the **sudo** process, **sudo** will relay signals it receives to the command. Unless the command is being run in a new pty, the SIGHUP, SIGINT and SIGQUIT signals are not relayed unless they are sent by a user process, not the kernel. Otherwise, the command would receive SIGINT twice every time the user entered control-C. Some signals, such as SIGSTOP and SIGKILL, cannot be caught and thus will not be relayed to the command. As a general rule, SIGTSTP should be used instead of SIGSTOP when you wish to suspend a command being run by **sudo**.

As a special case, **sudo** will not relay signals that were sent by the command it is running. This prevents the command from accidentally killing itself. On some systems, the reboot(8) command sends

SIGTERM to all non-system processes other than itself before rebooting the systyem. This prevents **sudo** from relaying the SIGTERM signal it received back to reboot(8), which might then exit before the system was actually rebooted, leaving it in a half-dead state similar to single user mode. Note, however, that this check only applies to the command run by **sudo** and not any other processes that the command may create. As a result, running a script that calls reboot(8) or shutdown(8) via **sudo** may cause the system to end up in this undefined state unless the reboot(8) or shutdown(8) are run using the **exec**() family of functions instead of **system**() (which interposes a shell between the command and the calling process).

PLUGINS

Plugins are dynamically loaded based on the contents of the */etc/sudo.conf* file. If no */etc/sudo.conf* file is present, or it contains no Plugin lines, **sudo** will use the traditional *sudoers* security policy and I/O logging, which corresponds to the following */etc/sudo.conf* file.

```
#
# Default /etc/sudo.conf file
#
# Format:
#   Plugin plugin_name plugin_path plugin_options ...
#   Path askpass /path/to/askpass
#   Path noexec /path/to/sudo_noexec.so
#   Debug sudo /var/log/sudo_debug all@warn
#   Set disable_coredump true
#
# The plugin_path is relative to /usr/local/libexec unless
#   fully qualified.
# The plugin_name corresponds to a global symbol in the plugin
#   that contains the plugin interface structure.
# The plugin_options are optional.
#
Plugin policy_plugin sudoers.so
Plugin io_plugin sudoers.so
```

A Plugin line consists of the Plugin keyword, followed by the *symbol_name* and the *path* to the shared object containing the plugin. The *symbol_name* is the name of the struct policy_plugin or struct io_plugin in the plugin shared object. The *path* may be fully qualified or relative. If not fully qualified it is relative to the */usr/local/libexec* directory. Any additional parameters after the *path* are passed as arguments to the plugin's *open* function. Lines that don't begin with Plugin, Path, Debug, or Set are silently ignored.

For more information, see the sudo_plugin(8) manual.

PATHS

A Path line consists of the Path keyword, followed by the name of the path to set and its value. E.g.

```
Path noexec /usr/local/libexec/sudo_noexec.so
Path askpass /usr/X11R6/bin/ssh-askpass
```

The following plugin-agnostic paths may be set in the */etc/sudo.conf* file:

askpass The fully qualified path to a helper program used to read the user's password when no terminal is available. This may be the case when **sudo** is executed from a graphical (as opposed to text-based) application. The program specified by *askpass* should display the argument passed to it as the prompt and write the user's password to the standard output. The value of *askpass* may be overridden by the SUDO_ASKPASS environment variable.

noexec The fully-qualified path to a shared library containing dummy versions of the **execv**(), **execve**() and **fexecve**() library functions that just return an error. This is used to implement the *noexec* functionality on systems that support LD_PRELOAD or its equivalent. Defaults to */usr/local/libexec/sudo_noexec.so*.

DEBUG FLAGS

sudo versions 1.8.4 and higher support a flexible debugging framework that can help track down what **sudo** is doing internally if there is a problem.

A Debug line consists of the Debug keyword, followed by the name of the program to debug (**sudo**, **visudo**, **sudoreplay**), the debug file name and a comma-separated list of debug flags. The debug flag syntax used by **sudo** and the *sudoers* plugin is *subsystem@priority* but the plugin is free to use a different format so long as it does not include a comma (',').

For instance:

```
Debug sudo /var/log/sudo_debug all@warn,plugin@info
```

would log all debugging statements at the *warn* level and higher in addition to those at the *info* level for the plugin subsystem.

Currently, only one Debug entry per program is supported. The **sudo** Debug entry is shared by the **sudo** front end, **sudoedit** and the plugins. A future release may add support for per-plugin Debug lines and/or support for multiple debugging files for a single program.

The priorities used by the **sudo** front end, in order of decreasing severity, are: *crit*, *err*, *warn*, *notice*, *diag*, *info*, *trace* and *debug*. Each priority, when specified, also includes all priorities higher than it. For example, a priority of *notice* would include debug messages logged at *notice* and higher.

The following subsystems are used by the **sudo** front-end:

all	matches every subsystem
args	command line argument processing
conv	user conversation
edit	sudoedit
exec	command execution
main	**sudo** main function
netif	network interface handling
pcomm	communication with the plugin
plugin	plugin configuration
pty	pseudo-tty related code
selinux	SELinux-specific handling
util	utility functions
utmp	utmp handling

EXIT VALUE

Upon successful execution of a program, the exit status from *sudo* will simply be the exit status of the program that was executed.

Otherwise, **sudo** exits with a value of 1 if there is a configuration/permission problem or if **sudo** cannot execute the given command. In the latter case the error string is printed to the standard error. If **sudo** cannot stat(2) one or more entries in the user's PATH, an error is printed on stderr. (If the directory does not exist or if it is not really a directory, the entry is ignored and no error is printed.) This should

not happen under normal circumstances. The most common reason for stat(2) to return "permission denied" is if you are running an automounter and one of the directories in your PATH is on a machine that is currently unreachable.

SECURITY NOTES

sudo tries to be safe when executing external commands.

To prevent command spoofing, **sudo** checks "." and "" (both denoting current directory) last when searching for a command in the user's PATH (if one or both are in the PATH). Note, however, that the actual PATH environment variable is *not* modified and is passed unchanged to the program that **sudo** executes.

Please note that **sudo** will normally only log the command it explicitly runs. If a user runs a command such as sudo su or sudo sh, subsequent commands run from that shell are not subject to **sudo**'s security policy. The same is true for commands that offer shell escapes (including most editors). If I/O logging is enabled, subsequent commands will have their input and/or output logged, but there will not be traditional logs for those commands. Because of this, care must be taken when giving users access to commands via **sudo** to verify that the command does not inadvertently give the user an effective root shell. For more information, please see the *PREVENTING SHELL ESCAPES* section in sudoers(5).

To prevent the disclosure of potentially sensitive information, **sudo** disables core dumps by default while it is executing (they are re-enabled for the command that is run). To aid in debugging **sudo** crashes, you may wish to re-enable core dumps by setting "disable_coredump" to false in the */etc/sudo.conf* file as follows:

```
Set disable_coredump false
```

Note that by default, most operating systems disable core dumps from setuid programs, which includes **sudo**. To actually get a **sudo** core file you may need to enable core dumps for setuid processes. On BSD and Linux systems this is accomplished via the sysctl command, on Solaris the coreadm command can be used.

ENVIRONMENT

sudo utilizes the following environment variables. The security policy has control over the actual content of the command's environment.

EDITOR Default editor to use in **-e** (sudoedit) mode if neither SUDO_EDITOR nor VISUAL is set.

MAIL In **-i** mode or when *env_reset* is enabled in *sudoers*, set to the mail spool of the target

user.

HOME Set to the home directory of the target user if **-i** or **-H** are specified, *env_reset* or *always_set_home* are set in *sudoers*, or when the **-s** option is specified and *set_home* is set in *sudoers*.

PATH May be overridden by the security policy.

SHELL Used to determine shell to run with **-s** option.

SUDO_ASKPASS
Specifies the path to a helper program used to read the password if no terminal is available or if the **-A** option is specified.

SUDO_COMMAND
Set to the command run by sudo.

SUDO_EDITOR Default editor to use in **-e** (sudoedit) mode.

SUDO_GID Set to the group ID of the user who invoked sudo.

SUDO_PROMPT Used as the default password prompt.

SUDO_PS1 If set, PS1 will be set to its value for the program being run.

SUDO_UID Set to the user ID of the user who invoked sudo.

SUDO_USER Set to the login name of the user who invoked sudo.

USER Set to the target user (root unless the **-u** option is specified).

VISUAL Default editor to use in **-e** (sudoedit) mode if SUDO_EDITOR is not set.

FILES

/etc/sudo.conf **sudo** front end configuration

EXAMPLES

Note: the following examples assume a properly configured security policy.

To get a file listing of an unreadable directory:

```
$ sudo ls /usr/local/protected
```

To list the home directory of user yaz on a machine where the file system holding ~yaz is not exported as root:

```
$ sudo -u yaz ls ~yaz
```

To edit the *index.html* file as user www:

```
$ sudo -u www vi ~www/htdocs/index.html
```

To view system logs only accessible to root and users in the adm group:

```
$ sudo -g adm view /var/log/syslog
```

To run an editor as jim with a different primary group:

```
$ sudo -u jim -g audio vi ~jim/sound.txt
```

To shut down a machine:

```
$ sudo shutdown -r +15 "quick reboot"
```

To make a usage listing of the directories in the /home partition. Note that this runs the commands in a sub-shell to make the cd and file redirection work.

```
$ sudo sh -c "cd /home ; du -s * | sort -rn > USAGE"
```

SEE ALSO

grep(1), su(1), stat(2), login_cap(3), passwd(5), sudoers(5), sudo_plugin(8), sudoreplay(8), visudo(8)

HISTORY

See the HISTORY file in the **sudo** distribution (http://www.sudo.ws/sudo/history.html) for a brief history of sudo.

AUTHORS

Many people have worked on **sudo** over the years; this version consists of code written primarily by:

Todd C. Miller

See the CONTRIBUTORS file in the **sudo** distribution (http://www.sudo.ws/sudo/contributors.html) for an exhaustive list of people who have contributed to **sudo**.

CAVEATS

There is no easy way to prevent a user from gaining a root shell if that user is allowed to run arbitrary commands via **sudo**. Also, many programs (such as editors) allow the user to run commands via shell escapes, thus avoiding **sudo**'s checks. However, on most systems it is possible to prevent shell escapes with the sudoers(5) plugin's *noexec* functionality.

It is not meaningful to run the cd command directly via sudo, e.g.,

```
$ sudo cd /usr/local/protected
```

since when the command exits the parent process (your shell) will still be the same. Please see the *EXAMPLES* section for more information.

Running shell scripts via **sudo** can expose the same kernel bugs that make setuid shell scripts unsafe on some operating systems (if your OS has a /dev/fd/ directory, setuid shell scripts are generally safe).

BUGS

If you feel you have found a bug in **sudo**, please submit a bug report at http://www.sudo.ws/sudo/bugs/

SUPPORT

Limited free support is available via the sudo-users mailing list, see http://www.sudo.ws/mailman/listinfo/sudo-users to subscribe or search the archives.

DISCLAIMER

sudo is provided "AS IS" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. See the LICENSE file distributed with **sudo** or http://www.sudo.ws/sudo/license.html for complete details.