NAME

sudoers - default sudo security policy plugin

DESCRIPTION

The **sudoers** policy plugin determines a user's **sudo** privileges. It is the default **sudo** policy plugin. The policy is driven by the */etc/sudoers* file or, optionally in LDAP. The policy format is described in detail in the *SUDOERS FILE FORMAT* section. For information on storing **sudoers** policy information in LDAP, please see sudoers.ldap(5).

Configuring sudo.conf for sudoers

sudo consults the sudo.conf(5) file to determine which policy and and I/O logging plugins to load. If no sudo.conf(5) file is present, or if it contains no Plugin lines, **sudoers** will be used for policy decisions and I/O logging. To explicitly configure sudo.conf(5) to use the **sudoers** plugin, the following configuration can be used.

```
Plugin sudoers_policy sudoers.so
Plugin sudoers_io sudoers.so
```

Starting with **sudo** 1.8.5, it is possible to specify optional arguments to the **sudoers** plugin in the sudo.conf(5) file. These arguments, if present, should be listed after the path to the plugin (i.e. after *sudoers.so*). Multiple arguments may be specified, separated by white space. For example:

```
Plugin sudoers_policy sudoers.so sudoers_mode=0400
```

The following plugin arguments are supported:

ldap_conf=pathname
: The *ldap_conf* argument can be used to override the default path to the *ldap.conf* file.

ldap_secret=pathname
: The *ldap_secret* argument can be used to override the default path to the *ldap.secret* file.

sudoers_file=pathname
: The *sudoers_file* argument can be used to override the default path to the *sudoers* file.

sudoers_uid=uid
: The *sudoers_uid* argument can be used to override the default owner of the sudoers file. It should be specified as a numeric user ID.

sudoers_gid=gid

The *sudoers_gid* argument can be used to override the default group of the sudoers file. It must be specified as a numeric group ID (not a group name).

sudoers_mode=mode

The *sudoers_mode* argument can be used to override the default file mode for the sudoers file. It should be specified as an octal value.

For more information on configuring sudo.conf(5), please refer to its manual.

User Authentication

The **sudoers** security policy requires that most users authenticate themselves before they can use **sudo**. A password is not required if the invoking user is root, if the target user is the same as the invoking user, or if the policy has disabled authentication for the user or command. Unlike su(1), when **sudoers** requires authentication, it validates the invoking user's credentials, not the target user's (or root's) credentials. This can be changed via the *rootpw*, *targetpw* and *runaspw* flags, described later.

If a user who is not listed in the policy tries to run a command via **sudo**, mail is sent to the proper authorities. The address used for such mail is configurable via the *mailto* Defaults entry (described later) and defaults to root.

Note that no mail will be sent if an unauthorized user tries to run **sudo** with the **-l** or **-v** option unless there is an authentication error and either the *mail_always* or *mail_badpass* flags are enabled. This allows users to determine for themselves whether or not they are allowed to use **sudo**. All attempts to run **sudo** (successful or not) will be logged, regardless of whether or not mail is sent.

If **sudo** is run by root and the SUDO_USER environment variable is set, the **sudoers** policy will use this value to determine who the actual user is. This can be used by a user to log commands through sudo even when a root shell has been invoked. It also allows the **-e** option to remain useful even when invoked via a sudo-run script or program. Note, however, that the *sudoers* file lookup is still done for root, not the user specified by SUDO_USER.

sudoers uses per-user time stamp files for credential caching. Once a user has been authenticated, a record is written containing the uid that was used to authenticate, the terminal session ID, and a time stamp (using a monotonic clock if one is available). The user may then use **sudo** without a password for a short period of time (5 minutes unless overridden by the *timeout* option). By default, **sudoers** uses a separate record for each tty, which means that a user's login sessions are authenticated separately. The *tty_tickets* option can be disabled to force the use of a single time stamp for all of a user's sessions.

Logging

sudoers can log both successful and unsuccessful attempts (as well as errors) to syslog(3), a log file, or

both. By default, **sudoers** will log via syslog(3) but this is changeable via the *syslog* and *logfile* Defaults settings. See *LOG FORMAT* for a description of the log file format.

sudoers is also capable of running a command in a pseudo-tty and logging all input and/or output. The standard input, standard output and standard error can be logged even when not associated with a terminal. I/O logging is not on by default but can be enabled using the *log_input* and *log_output* options as well as the LOG_INPUT and LOG_OUTPUT command tags. See *I/O LOG FILES* for details on how I/O log files are stored.

Command environment

Since environment variables can influence program behavior, **sudoers** provides a means to restrict which variables from the user's environment are inherited by the command to be run. There are two distinct ways **sudoers** can deal with environment variables.

By default, the *env_reset* option is enabled. This causes commands to be executed with a new, minimal environment. On AIX (and Linux systems without PAM), the environment is initialized with the contents of the */etc/environment* file. On BSD systems, if the *use_loginclass* option is enabled, the environment is initialized based on the *path* and *setenv* settings in */etc/login.conf*. The new environment contains the TERM, PATH, HOME, MAIL, SHELL, LOGNAME, USER, USERNAME and SUDO_* variables in addition to variables from the invoking process permitted by the *env_check* and *env_keep* options. This is effectively a whitelist for environment variables. Environment variables with a value beginning with () are removed unless both the name and value parts are matched by *env_keep* or *env_check*, as they will be interpreted as functions by older versions of the **bash** shell. Prior to version 1.8.11, such variables were always removed.

If, however, the *env_reset* option is disabled, any variables not explicitly denied by the *env_check* and *env_delete* options are inherited from the invoking process. In this case, *env_check* and *env_delete* behave like a blacklist. Environment variables with a value beginning with () are always removed, even if they do not match one of the blacklists. Since it is not possible to blacklist all potentially dangerous environment variables, use of the default *env_reset* behavior is encouraged.

By default, environment variables are matched by name. However, if the pattern includes an equal sign ('='), both the variables name and value must match. For example, an old-style (pre-shellshock) **bash** shell function could be matched as follows:

```
env_keep += "my_func=()*"
```

Without the ''=()*'' suffix, this would not match, as old-style **bash** shell functions are not preserved by default.

The complete list of environment variables that **sudo** allows or denies is contained in the output of ‘‘sudo -V’’ when run as root. Please note that this list varies based on the operating system **sudo** is running on.

On systems that support PAM where the **pam_env** module is enabled for **sudo**, variables in the PAM environment may be merged in to the environment. If a variable in the PAM environment is already present in the user’s environment, the value will only be overridden if the variable was not preserved by **sudoers**. When *env_reset* is enabled, variables preserved from the invoking user’s environment by the *env_keep* list take precedence over those in the PAM environment. When *env_reset* is disabled, variables present the invoking user’s environment take precedence over those in the PAM environment unless they match a pattern in the *env_delete* list.

Note that the dynamic linker on most operating systems will remove variables that can control dynamic linking from the environment of setuid executables, including **sudo**. Depending on the operating system this may include _RLD*, DYLD_*, LD_*, LDR_*, LIBPATH, SHLIB_PATH, and others. These type of variables are removed from the environment before **sudo** even begins execution and, as such, it is not possible for **sudo** to preserve them.

As a special case, if **sudo**’s **-i** option (initial login) is specified, **sudoers** will initialize the environment regardless of the value of *env_reset*. The DISPLAY, PATH and TERM variables remain unchanged; HOME, MAIL, SHELL, USER, and LOGNAME are set based on the target user. On AIX (and Linux systems without PAM), the contents of */etc/environment* are also included. On BSD systems, if the *use_loginclass* flag is enabled, the *path* and *setenv* variables in */etc/login.conf* are also applied. All other environment variables are removed.

Finally, if the *env_file* option is defined, any variables present in that file will be set to their specified values as long as they would not conflict with an existing environment variable.

SUDOERS FILE FORMAT

The *sudoers* file is composed of two types of entries: aliases (basically variables) and user specifications (which specify who may run what).

When multiple entries match for a user, they are applied in order. Where there are multiple matches, the last match is used (which is not necessarily the most specific match).

The *sudoers* file grammar will be described below in Extended Backus-Naur Form (EBNF). Don’t despair if you are unfamiliar with EBNF; it is fairly simple, and the definitions below are annotated.

Quick guide to EBNF

EBNF is a concise and exact way of describing the grammar of a language. Each EBNF definition is

made up of *production rules*. E.g.,

symbol ::= definition | alternate1 | alternate2 ...

Each *production rule* references others and thus makes up a grammar for the language. EBNF also contains the following operators, which many readers will recognize from regular expressions. Do not, however, confuse them with ‘‘wildcard’’ characters, which have different meanings.

? Means that the preceding symbol (or group of symbols) is optional. That is, it may appear once or not at all.

* Means that the preceding symbol (or group of symbols) may appear zero or more times.

+ Means that the preceding symbol (or group of symbols) may appear one or more times.

Parentheses may be used to group symbols together. For clarity, we will use single quotes (’’) to designate what is a verbatim character string (as opposed to a symbol name).

Aliases

There are four kinds of aliases: User_Alias, Runas_Alias, Host_Alias and Cmnd_Alias.

```
Alias ::= 'User_Alias'  User_Alias (':' User_Alias)* |
          'Runas_Alias' Runas_Alias (':' Runas_Alias)* |
          'Host_Alias'  Host_Alias (':' Host_Alias)* |
          'Cmnd_Alias'  Cmnd_Alias (':' Cmnd_Alias)*

User_Alias ::= NAME '=' User_List

Runas_Alias ::= NAME '=' Runas_List

Host_Alias ::= NAME '=' Host_List

Cmnd_Alias ::= NAME '=' Cmnd_List

NAME ::= [A-Z]([A-Z][0-9]_)*
```

Each *alias* definition is of the form

Alias_Type NAME = item1, item2, ...

where *Alias_Type* is one of User_Alias, Runas_Alias, Host_Alias, or Cmnd_Alias. A NAME is a string of uppercase letters, numbers, and underscore characters (‘_’). A NAME **must** start with an uppercase letter. It is possible to put several alias definitions of the same type on a single line, joined by a colon (‘:’). E.g.,

Alias_Type NAME = item1, item2, item3 : NAME = item4, item5

It is a syntax error to redefine an existing *alias*. It is possible to use the same name for *aliases* of different types, but this is not recommended.

The definitions of what constitutes a valid *alias* member follow.

```
User_List ::= User |
        User ’,’ User_List

User ::= ’!’* user name |
      ’!’* #uid |
      ’!’* %group |
      ’!’* %#gid |
      ’!’* +netgroup |
      ’!’* %:nonunix_group |
      ’!’* %:#nonunix_gid |
      ’!’* User_Alias
```

A User_List is made up of one or more user names, user IDs (prefixed with ‘#’), system group names and IDs (prefixed with ‘%’ and ‘%#’ respectively), netgroups (prefixed with ‘+’), non-Unix group names and IDs (prefixed with ‘%:’ and ‘%:#’ respectively) and User_Aliases. Each list item may be prefixed with zero or more ‘!’ operators. An odd number of ‘!’ operators negate the value of the item; an even number just cancel each other out. User netgroups are matched using the user and domain members only; the host member is not used when matching.

A user name, uid, group, gid, netgroup, nonunix_group or nonunix_gid may be enclosed in double quotes to avoid the need for escaping special characters. Alternately, special characters may be specified in escaped hex mode, e.g. \x20 for space. When using double quotes, any prefix characters must be included inside the quotes.

The actual nonunix_group and nonunix_gid syntax depends on the underlying group provider plugin. For instance, the QAS AD plugin supports the following formats:

- Group in the same domain: "%:Group Name"

- Group in any domain: "%:Group Name@FULLY.QUALIFIED.DOMAIN"

- Group SID: "%:S-1-2-34-5678901234-5678901234-5678901234-567"

See *GROUP PROVIDER PLUGINS* for more information.

Note that quotes around group names are optional. Unquoted strings must use a backslash (‘\’) to escape spaces and special characters. See *Other special characters and reserved words* for a list of characters that need to be escaped.

```
Runas_List ::= Runas_Member |
               Runas_Member ’,’ Runas_List

Runas_Member ::= ’!’* user name |
                 ’!’* #uid |
                 ’!’* %group |
                 ’!’* %#gid |
                 ’!’* %:nonunix_group |
                 ’!’* %:#nonunix_gid |
                 ’!’* +netgroup |
                 ’!’* Runas_Alias
```

A Runas_List is similar to a User_List except that instead of User_Aliases it can contain Runas_Aliases. Note that user names and groups are matched as strings. In other words, two users (groups) with the same uid (gid) are considered to be distinct. If you wish to match all user names with the same uid (e.g. root and toor), you can use a uid instead (#0 in the example given).

```
Host_List ::= Host |
              Host ’,’ Host_List

Host ::= ’!’* host name |
         ’!’* ip_addr |
         ’!’* network(/netmask)? |
         ’!’* +netgroup |
         ’!’* Host_Alias
```

A Host_List is made up of one or more host names, IP addresses, network numbers, netgroups (prefixed with ‘+’) and other aliases. Again, the value of an item may be negated with the ‘!’ operator. Host netgroups are matched using the host (both qualified and unqualified) and domain members only; the user member is not used when matching. If you specify a network number without a netmask, **sudo** will

query each of the local host's network interfaces and, if the network number corresponds to one of the hosts's network interfaces, will use the netmask of that interface. The netmask may be specified either in standard IP address notation (e.g. 255.255.255.0 or ffff:ffff:ffff:ffff::), or CIDR notation (number of bits, e.g. 24 or 64). A host name may include shell-style wildcards (see the *Wildcards* section below), but unless the host name command on your machine returns the fully qualified host name, you'll need to use the *fqdn* option for wildcards to be useful. Note that **sudo** only inspects actual network interfaces; this means that IP address 127.0.0.1 (localhost) will never match. Also, the host name ''localhost'' will only match if that is the actual host name, which is usually only the case for non-networked systems.

```
digest ::= [A-Fa-f0-9]+ |
        [[A-Za-z0-9+/=]+

Digest_Spec ::= "sha224" ':' digest |
             "sha256" ':' digest |
             "sha384" ':' digest |
             "sha512" ':' digest

Cmnd_List ::= Cmnd |
           Cmnd ',' Cmnd_List

command name ::= file name |
              file name args |
              file name '""'

Cmnd ::= Digest_Spec? '!'* command name |
       '!'* directory |
       '!'* "sudoedit" |
       '!'* Cmnd_Alias
```

A Cmnd_List is a list of one or more command names, directories, and other aliases. A command name is a fully qualified file name which may include shell-style wildcards (see the *Wildcards* section below). A simple file name allows the user to run the command with any arguments he/she wishes. However, you may also specify command line arguments (including wildcards). Alternately, you can specify "" to indicate that the command may only be run **without** command line arguments. A directory is a fully qualified path name ending in a '/'. When you specify a directory in a Cmnd_List, the user will be able to run any file within that directory (but not in any sub-directories therein).

If a Cmnd has associated command line arguments, then the arguments in the Cmnd must match exactly those given by the user on the command line (or match the wildcards if there are any). Note that the following characters must be escaped with a '\' if they are used in command arguments: ',', ':', '=', '\'.

The built-in command ‘‘sudoedit’’ is used to permit a user to run **sudo** with the **-e** option (or as **sudoedit**). It may take command line arguments just as a normal command does. Note that ‘‘sudoedit’’ is a command built into **sudo** itself and must be specified in the *sudoers* file without a leading path.

If a command name is prefixed with a Digest_Spec, the command will only match successfully if it can be verified using the specified SHA-2 digest. The following digest formats are supported: sha224, sha256, sha384 and sha512. The string may be specified in either hex or base64 format (base64 is more compact). There are several utilities capable of generating SHA-2 digests in hex format such as openssl, shasum, sha224sum, sha256sum, sha384sum, sha512sum.

For example, using openssl:

```
$ openssl dgst -sha224 /bin/ls
SHA224(/bin/ls)= 118187da8364d490b4a7debbf483004e8f3e053ec954309de2c41a25
```

It is also possible to use openssl to generate base64 output:

```
$ openssl dgst -binary -sha224 /bin/ls | openssl base64
EYGH2oNk1JC0p9679IMATo8+BT7JVDCd4sQaJQ==
```

Warning, if the user has write access to the command itself (directly or via a **sudo** command), it may be possible for the user to replace the command after the digest check has been performed but before the command is executed. A similar race condition exists on systems that lack the fexecve(2) system call when the directory in which the command is located is writable by the user.

Command digests are only supported by version 1.8.7 or higher.

Defaults

Certain configuration options may be changed from their default values at run-time via one or more Default_Entry lines. These may affect all users on any host, all users on a specific host, a specific user, a specific command, or commands being run as a specific user. Note that per-command entries may not include command line arguments. If you need to specify arguments, define a Cmnd_Alias and reference that instead.

```
Default_Type ::= ’Defaults’ |
                 ’Defaults’ ’@’ Host_List |
                 ’Defaults’ ’:’ User_List |
                 ’Defaults’ ’!’ Cmnd_List |
                 ’Defaults’ ’>’ Runas_List
```

Default_Entry ::= Default_Type Parameter_List

Parameter_List ::= Parameter |
Parameter ’,’ Parameter_List

Parameter ::= Parameter ’=’ Value |
Parameter ’+=’ Value |
Parameter ’-=’ Value |
’!’* Parameter

Parameters may be **flags**, **integer** values, **strings**, or **lists**. Flags are implicitly boolean and can be turned off via the ‘!’ operator. Some integer, string and list parameters may also be used in a boolean context to disable them. Values may be enclosed in double quotes ("") when they contain multiple words. Special characters may be escaped with a backslash (‘\’).

Lists have two additional assignment operators, += and -=. These operators are used to add to and delete from a list respectively. It is not an error to use the -= operator to remove an element that does not exist in a list.

Defaults entries are parsed in the following order: generic, host, user and runas Defaults first, then command defaults. If there are multiple Defaults settings of the same type, the last matching setting is used. The following Defaults settings are parsed before all others since they may affect subsequent entries: *fqdn*, *group_plugin*, *runas_default*, *sudoers_locale*.

See *SUDOERS OPTIONS* for a list of supported Defaults parameters.

User specification

User_Spec ::= User_List Host_List ’=’ Cmnd_Spec_List \
(’:’ Host_List ’=’ Cmnd_Spec_List)*

Cmnd_Spec_List ::= Cmnd_Spec |
Cmnd_Spec ’,’ Cmnd_Spec_List

Cmnd_Spec ::= Runas_Spec? SELinux_Spec? Solaris_Priv_Spec? Tag_Spec* Cmnd

Runas_Spec ::= ’(’ Runas_List? (’:’ Runas_List)? ’)’

SELinux_Spec ::= (’ROLE=role’ | ’TYPE=type’)

Solaris_Priv_Spec ::= (’PRIVS=privset’ | ’LIMITPRIVS=privset’)

Tag_Spec ::= (’EXEC:’ | ’NOEXEC:’ | ’FOLLOW:’ | ’NOFOLLOW’ |
’LOG_INPUT:’ | ’NOLOG_INPUT:’ | ’LOG_OUTPUT:’ |
’NOLOG_OUTPUT:’ | ’MAIL:’ | ’NOMAIL:’ | ’PASSWD:’ |
’NOPASSWD:’ | ’SETENV:’ | ’NOSETENV:’)

A **user specification** determines which commands a user may run (and as what user) on specified hosts. By default, commands are run as **root**, but this can be changed on a per-command basis.

The basic structure of a user specification is ‘‘who where = (as_whom) what’’. Let’s break that down into its constituent parts:

Runas_Spec

A Runas_Spec determines the user and/or the group that a command may be run as. A fully-specified Runas_Spec consists of two Runas_Lists (as defined above) separated by a colon (‘:’) and enclosed in a set of parentheses. The first Runas_List indicates which users the command may be run as via **sudo**’s **-u** option. The second defines a list of groups that can be specified via **sudo**’s **-g** option. If both Runas_Lists are specified, the command may be run with any combination of users and groups listed in their respective Runas_Lists. If only the first is specified, the command may be run as any user in the list but no **-g** option may be specified. If the first Runas_List is empty but the second is specified, the command may be run as the invoking user with the group set to any listed in the Runas_List. If both Runas_Lists are empty, the command may only be run as the invoking user. If no Runas_Spec is specified the command may be run as **root** and no group may be specified.

A Runas_Spec sets the default for the commands that follow it. What this means is that for the entry:

dgb boulder = (operator) /bin/ls, /bin/kill, /usr/bin/lprm

The user **dgb** may run */bin/ls*, */bin/kill*, and */usr/bin/lprm*--but only as **operator**. E.g.,

$ sudo -u operator /bin/ls

It is also possible to override a Runas_Spec later on in an entry. If we modify the entry like so:

dgb boulder = (operator) /bin/ls, (root) /bin/kill, /usr/bin/lprm

Then user **dgb** is now allowed to run */bin/ls* as **operator**, but */bin/kill* and */usr/bin/lprm* as **root**.

We can extend this to allow **dgb** to run /bin/ls with either the user or group set to **operator**:

dgb boulder = (operator : operator) /bin/ls, (root) /bin/kill,\

```
    /usr/bin/lprm
```

Note that while the group portion of the Runas_Spec permits the user to run as command with that group, it does not force the user to do so. If no group is specified on the command line, the command will run with the group listed in the target user's password database entry. The following would all be permitted by the sudoers entry above:

```
$ sudo -u operator /bin/ls
$ sudo -u operator -g operator /bin/ls
$ sudo -g operator /bin/ls
```

In the following example, user **tcm** may run commands that access a modem device file with the dialer group.

```
tcm  boulder = (:dialer) /usr/bin/tip, /usr/bin/cu,\
    /usr/local/bin/minicom
```

Note that in this example only the group will be set, the command still runs as user **tcm**. E.g.

```
$ sudo -g dialer /usr/bin/cu
```

Multiple users and groups may be present in a Runas_Spec, in which case the user may select any combination of users and groups via the **-u** and **-g** options. In this example:

```
alan  ALL = (root, bin : operator, system) ALL
```

user **alan** may run any command as either user root or bin, optionally setting the group to operator or system.

SELinux_Spec

On systems with SELinux support, *sudoers* file entries may optionally have an SELinux role and/or type associated with a command. If a role or type is specified with the command it will override any default values specified in *sudoers*. A role or type specified on the command line, however, will supersede the values in *sudoers*.

Solaris_Priv_Spec

On Solaris systems, *sudoers* file entries may optionally specify Solaris privilege set and/or limit privilege set associated with a command. If privileges or limit privileges are specified with the command it will override any default values specified in *sudoers*.

A privilege set is a comma-separated list of privilege names. The ppriv(1) command can be used to list all privileges known to the system. For example:

$ ppriv -l

In addition, there are several ‘‘special’’ privilege strings:

none the empty set

all the set of all privileges

zone the set of all privileges available in the current zone

basic the default set of privileges normal users are granted at login time

Privileges can be excluded from a set by prefixing the privilege name with either an ‘!’ or ‘-’ character.

Tag_Spec

A command may have zero or more tags associated with it. There are ten possible tag values: EXEC, NOEXEC, FOLLOW, NOFOLLOW, LOG_INPUT, NOLOG_INPUT, LOG_OUTPUT, NOLOG_OUTPUT, MAIL, NOMAIL, PASSWD, NOPASSWD, SETENV, and NOSETENV. Once a tag is set on a Cmnd, subsequent Cmnds in the Cmnd_Spec_List, inherit the tag unless it is overridden by the opposite tag (in other words, PASSWD overrides NOPASSWD and NOEXEC overrides EXEC).

EXEC and *NOEXEC*

If **sudo** has been compiled with *noexec* support and the underlying operating system supports it, the NOEXEC tag can be used to prevent a dynamically-linked executable from running further commands itself.

In the following example, user **aaron** may run */usr/bin/more* and */usr/bin/vi* but shell escapes will be disabled.

aaron shanty = NOEXEC: /usr/bin/more, /usr/bin/vi

See the *Preventing shell escapes* section below for more details on how NOEXEC works and whether or not it will work on your system.

FOLLOW and *NOFOLLOW* Starting with version 1.8.15, **sudoedit** will not open a file that is a symbolic link unless the *sudoedit_follow* option is enabled. The *FOLLOW* and *NOFOLLOW* tags

override the value of *sudoedit_follow* and can be used to permit (or deny) the editing of symbolic links on a per-command basis. These tags are only effective for the *sudoedit* command and are ignored for all other commands.

LOG_INPUT and *NOLOG_INPUT*

These tags override the value of the *log_input* option on a per-command basis. For more information, see the description of *log_input* in the *SUDOERS OPTIONS* section below.

LOG_OUTPUT and *NOLOG_OUTPUT*

These tags override the value of the *log_output* option on a per-command basis. For more information, see the description of *log_output* in the *SUDOERS OPTIONS* section below.

MAIL and *NOMAIL*

These tags provide fine-grained control over whether mail will be sent when a user runs a command by overriding the value of the *mail_all_cmnds* option on a per-command basis. They have no effect when **sudo** is run with the **-l** or **-v** options. A *NOMAIL* tag will also override the *mail_always* and *mail_no_perms* options. For more information, see the descriptions of *mail_all_cmnds*, *mail_always*, and *mail_no_perms* in the *SUDOERS OPTIONS* section below.

PASSWD and *NOPASSWD*

By default, **sudo** requires that a user authenticate him or herself before running a command. This behavior can be modified via the NOPASSWD tag. Like a Runas_Spec, the NOPASSWD tag sets a default for the commands that follow it in the Cmnd_Spec_List. Conversely, the PASSWD tag can be used to reverse things. For example:

ray rushmore = NOPASSWD: /bin/kill, /bin/ls, /usr/bin/lprm

would allow the user **ray** to run */bin/kill*, */bin/ls*, and */usr/bin/lprm* as **root** on the machine rushmore without authenticating himself. If we only want **ray** to be able to run */bin/kill* without a password the entry would be:

ray rushmore = NOPASSWD: /bin/kill, PASSWD: /bin/ls, /usr/bin/lprm

Note, however, that the PASSWD tag has no effect on users who are in the group specified by the *exempt_group* option.

By default, if the NOPASSWD tag is applied to any of the entries for a user on the current host, he or she will be able to run ‘‘sudo -l’’ without a password. Additionally, a user may only run ‘‘sudo -v’’ without a password if the NOPASSWD tag is present for all a user’s entries that pertain to the current host. This behavior may be overridden via the *verifypw* and *listpw* options.

SETENV and *NOSETENV*

These tags override the value of the *setenv* option on a per-command basis. Note that if SETENV has been set for a command, the user may disable the *env_reset* option from the command line via the **-E** option. Additionally, environment variables set on the command line are not subject to the restrictions imposed by *env_check*, *env_delete*, or *env_keep*. As such, only trusted users should be allowed to set variables in this manner. If the command matched is **ALL**, the SETENV tag is implied for that command; this default may be overridden by use of the NOSETENV tag.

Wildcards

sudo allows shell-style *wildcards* (aka meta or glob characters) to be used in host names, path names and command line arguments in the *sudoers* file. Wildcard matching is done via the glob(3) and fnmatch(3) functions as specified by IEEE Std 1003.1 (‘‘POSIX.1’’).

*	Matches any set of zero or more characters (including white space).
?	Matches any single character (including white space).
[...]	Matches any character in the specified range.
[!...]	Matches any character *not* in the specified range.
\x	For any character ‘x’, evaluates to ‘x’. This is used to escape special characters such as: ‘*’, ‘?’, ‘[’, and ‘]’.

Note that these are not regular expressions. Unlike a regular expression there is no way to match one or more characters within a range.

Character classes may be used if your system’s glob(3) and fnmatch(3) functions support them. However, because the ‘:’ character has special meaning in *sudoers*, it must be escaped. For example:

```
/bin/ls [[\:alpha\:]]*
```

Would match any file name beginning with a letter.

Note that a forward slash (‘/’) will *not* be matched by wildcards used in the file name portion of the command. This is to make a path like:

```
/usr/bin/*
```

match */usr/bin/who* but not */usr/bin/X11/xterm*.

When matching the command line arguments, however, a slash *does* get matched by wildcards since command line arguments may contain arbitrary strings and not just path names.

Wildcards in command line arguments should be used with care.
Command line arguments are matched as a single, concatenated string. This mean a wildcard character such as ‘?’ or ‘*’ will match across word boundaries, which may be unexpected. For example, while a sudoers entry like:

```
%operator ALL = /bin/cat /var/log/messages*
```

will allow command like:

```
$ sudo cat /var/log/messages.1
```

It will also allow:

```
$ sudo cat /var/log/messages /etc/shadow
```

which is probably not what was intended. In most cases it is better to do command line processing outside of the *sudoers* file in a scripting language.

Exceptions to wildcard rules

The following exceptions apply to the above rules:

"" If the empty string "" is the only command line argument in the *sudoers* file entry it means that command is not allowed to be run with *any* arguments.

sudoedit Command line arguments to the *sudoedit* built-in command should always be path names, so a forward slash (‘/’) will not be matched by a wildcard.

Including other files from within sudoers

It is possible to include other *sudoers* files from within the *sudoers* file currently being parsed using the #include and #includedir directives.

This can be used, for example, to keep a site-wide *sudoers* file in addition to a local, per-machine file. For the sake of this example the site-wide *sudoers* file will be */etc/sudoers* and the per-machine one will be */etc/sudoers.local*. To include */etc/sudoers.local* from within */etc/sudoers* we would use the following line in */etc/sudoers*:

```
#include /etc/sudoers.local
```

When **sudo** reaches this line it will suspend processing of the current file (*/etc/sudoers*) and switch to */etc/sudoers.local*. Upon reaching the end of */etc/sudoers.local*, the rest of */etc/sudoers* will be processed. Files that are included may themselves include other files. A hard limit of 128 nested include files is enforced to prevent include file loops.

If the path to the include file is not fully-qualified (does not begin with a '/', it must be located in the same directory as the sudoers file it was included from. For example, if */etc/sudoers* contains the line:

```
#include sudoers.local
```

the file that will be included is */etc/sudoers.local*.

The file name may also include the %h escape, signifying the short form of the host name. In other words, if the machine's host name is ''xerxes'', then

```
#include /etc/sudoers.%h
```

will cause **sudo** to include the file */etc/sudoers.xerxes*.

The #includedir directive can be used to create a *sudoers.d* directory that the system package manager can drop *sudoers* file rules into as part of package installation. For example, given:

```
#includedir /etc/sudoers.d
```

sudo will read each file in */etc/sudoers.d*, skipping file names that end in '~' or contain a '.' character to avoid causing problems with package manager or editor temporary/backup files. Files are parsed in sorted lexical order. That is, */etc/sudoers.d/01_first* will be parsed before */etc/sudoers.d/10_second*. Be aware that because the sorting is lexical, not numeric, */etc/sudoers.d/1_whoops* would be loaded *after* */etc/sudoers.d/10_second*. Using a consistent number of leading zeroes in the file names can be used to avoid such problems.

Note that unlike files included via #include, **visudo** will not edit the files in a #includedir directory unless one of them contains a syntax error. It is still possible to run **visudo** with the **-f** flag to edit the

files directly, but this will not catch the redefinition of an *alias* that is also present in a different file.

Other special characters and reserved words

The pound sign ('#') is used to indicate a comment (unless it is part of a #include directive or unless it occurs in the context of a user name and is followed by one or more digits, in which case it is treated as a uid). Both the comment character and any text after it, up to the end of the line, are ignored.

The reserved word **ALL** is a built-in *alias* that always causes a match to succeed. It can be used wherever one might otherwise use a Cmnd_Alias, User_Alias, Runas_Alias, or Host_Alias. You should not try to define your own *alias* called **ALL** as the built-in alias will be used in preference to your own. Please note that using **ALL** can be dangerous since in a command context, it allows the user to run *any* command on the system.

An exclamation point ('!') can be used as a logical *not* operator in a list or *alias* as well as in front of a Cmnd. This allows one to exclude certain values. For the '!' operator to be effective, there must be something for it to exclude. For example, to match all users except for root one would use:

```
ALL,!root
```

If the **ALL**, is omitted, as in:

```
!root
```

it would explicitly deny root but not match any other users. This is different from a true ''negation'' operator.

Note, however, that using a '!' in conjunction with the built-in **ALL** alias to allow a user to run ''all but a few'' commands rarely works as intended (see *SECURITY NOTES* below).

Long lines can be continued with a backslash ('\') as the last character on the line.

White space between elements in a list as well as special syntactic characters in a *User Specification* ('=', ':', '(', ')') is optional.

The following characters must be escaped with a backslash ('\') when used as part of a word (e.g. a user name or host name): '!', '=', ':', ',', '(', ')', '\'.

SUDOERS OPTIONS

sudo's behavior can be modified by Default_Entry lines, as explained earlier. A list of all supported Defaults parameters, grouped by type, are listed below.

Boolean Flags:

always_query_group_plugin
: If a *group_plugin* is configured, use it to resolve groups of the form %group as long as there is not also a system group of the same name. Normally, only groups of the form %:group are passed to the *group_plugin*. This flag is *off* by default.

always_set_home
: If enabled, **sudo** will set the HOME environment variable to the home directory of the target user (which is root unless the **-u** option is used). This effectively means that the **-H** option is always implied. Note that by default, HOME will be set to the home directory of the target user when the *env_reset* option is enabled, so *always_set_home* only has an effect for configurations where either *env_reset* is disabled or HOME is present in the *env_keep* list. This flag is *off* by default.

authenticate
: If set, users must authenticate themselves via a password (or other means of authentication) before they may run commands. This default may be overridden via the PASSWD and NOPASSWD tags. This flag is *on* by default.

closefrom_override
: If set, the user may use **sudo**'s **-C** option which overrides the default starting point at which **sudo** begins closing open file descriptors. This flag is *off* by default.

compress_io
: If set, and **sudo** is configured to log a command's input or output, the I/O logs will be compressed using **zlib**. This flag is *on* by default when **sudo** is compiled with **zlib** support.

exec_background
: By default, **sudo** runs a command as the foreground process as long as **sudo** itself is running in the foreground. When the *exec_background* flag is enabled and the command is being run in a pty (due to I/O logging or the *use_pty* flag), the command will be run as a background process. Attempts to read from the controlling terminal (or to change terminal settings) will result in the command being suspended with the SIGTTIN signal (or SIGTTOU in the case of terminal settings). If this happens when **sudo** is a foreground process, the command will be granted the controlling terminal and resumed in the foreground with no user intervention required. The advantage of initially running the command in the background is that **sudo** need not read from the terminal unless the command explicitly requests it. Otherwise, any terminal input must be passed to the command, whether it has required it or not (the kernel buffers terminals so it is not possible to tell whether the command really wants the input). This is different from historic *sudo* behavior or when the command is not being run in a pty.

For this to work seamlessly, the operating system must support the automatic restarting of system calls. Unfortunately, not all operating systems do this by default, and even those that do may have bugs. For example, Mac OS X fails to restart the **tcgetattr**() and **tcsetattr**() system calls (this is a bug in Mac OS X). Furthermore, because this behavior depends on the command stopping with the SIGTTIN or SIGTTOU signals, programs that catch these signals and suspend themselves with a different signal (usually SIGTOP) will not be automatically foregrounded. Some versions of the linux su(1) command behave this way. This flag is *off* by default.

This setting is only supported by version 1.8.7 or higher. It has no effect unless I/O logging is enabled or the *use_pty* flag is enabled.

env_editor If set, **visudo** will use the value of the EDITOR or VISUAL environment variables before falling back on the default editor list. Note that this may create a security hole as it allows the user to run any arbitrary command as root without logging. A safer alternative is to place a colon-separated list of editors in the editor variable. **visudo** will then only use the EDITOR or VISUAL if they match a value specified in editor. If the *env_reset* flag is enabled, the EDITOR and/or VISUAL environment variables must be present in the *env_keep* list for the *env_editor* flag to function when **visudo** is invoked via **sudo**. This flag is *off* by default.

env_reset If set, **sudo** will run the command in a minimal environment containing the TERM, PATH, HOME, MAIL, SHELL, LOGNAME, USER, USERNAME and SUDO_* variables. Any variables in the caller's environment that match the env_keep and env_check lists are then added, followed by any variables present in the file specified by the *env_file* option (if any). The contents of the env_keep and env_check lists, as modified by global Defaults parameters in *sudoers*, are displayed when **sudo** is run by root with the **-V** option. If the *secure_path* option is set, its value will be used for the PATH environment variable. This flag is *on* by default.

fast_glob Normally, **sudo** uses the glob(3) function to do shell-style globbing when matching path names. However, since it accesses the file system, glob(3) can take a long time to complete for some patterns, especially when the pattern references a network file system that is mounted on demand (auto mounted). The *fast_glob* option causes **sudo** to use the fnmatch(3) function, which does not access the file system to do its matching. The disadvantage of *fast_glob* is that it is unable to match relative path names such as *./ls* or *../bin/ls*. This has security implications when path names that include globbing characters are used with the negation

operator, ‘!’, as such rules can be trivially bypassed. As such, this option should not be used when the *sudoers* file contains rules that contain negated path names which include globbing characters. This flag is *off* by default.

fqdn Set this flag if you want to put fully qualified host names in the *sudoers* file when the local host name (as returned by the hostname command) does not contain the domain name. In other words, instead of myhost you would use myhost.mydomain.edu. You may still use the short form if you wish (and even mix the two). This option is only effective when the ‘‘canonical’’ host name, as returned by the **getaddrinfo**() or **gethostbyname**() function, is a fully-qualified domain name. This is usually the case when the system is configured to use DNS for host name resolution.

If the system is configured to use the */etc/hosts* file in preference to DNS, the ‘‘canonical’’ host name may not be fully-qualified. The order that sources are queried for host name resolution is usually specified in the */etc/nsswitch.conf*, */etc/netsvc.conf*, */etc/host.conf*, or, in some cases, */etc/resolv.conf* file. In the */etc/hosts* file, the first host name of the entry is considered to be the ‘‘canonical’’ name; subsequent names are aliases that are not used by **sudoers**. For example, the following hosts file line for the machine ‘‘xyzzy’’ has the fully-qualified domain name as the ‘‘canonical’’ host name, and the short version as an alias.

```
192.168.1.1 xyzzy.sudo.ws xyzzy
```

If the machine’s hosts file entry is not formatted properly, the *fqdn* option will not be effective if it is queried before DNS.

Beware that when using DNS for host name resolution, turning on *fqdn* requires **sudoers** to make DNS lookups which renders **sudo** unusable if DNS stops working (for example if the machine is disconnected from the network). Also note that just like with the hosts file, you must use the ‘‘canonical’’ name as DNS knows it. That is, you may not use a host alias (CNAME entry) due to performance issues and the fact that there is no way to get all aliases from DNS.

This flag is *off* by default.

ignore_audit_errors Allow commands to be run even if **sudoers** cannot write to the audit log. If enabled, an audit log write failure is not treated as a fatal error. If disabled, a command may only be run after the audit event is successfully written. This flag is only effective on systems for which **sudoers** supports audit logging, including

FreeBSD, Linux, Mac OS X and Solaris. This flag is *on* by default.

ignore_dot
: If set, **sudo** will ignore "." or "" (both denoting current directory) in the PATH environment variable; the PATH itself is not modified. This flag is *off* by default.

ignore_iolog_errors
: Allow commands to be run even if **sudoers** cannot write to the I/O log. If enabled, an I/O log write failure is not treated as a fatal error. If disabled, the command will be terminated if the I/O log cannot be written to. This flag is *off* by default.

ignore_logfile_errors
: Allow commands to be run even if **sudoers** cannot write to the log file. If enabled, a log file write failure is not treated as a fatal error. If disabled, a command may only be run after the log file entry is successfully written. This flag only has an effect when **sudoers** is configured to use file-based logging via the *logfile* option. This flag is *on* by default.

ignore_local_sudoers
: If set via LDAP, parsing of */etc/sudoers* will be skipped. This is intended for Enterprises that wish to prevent the usage of local sudoers files so that only LDAP is used. This thwarts the efforts of rogue operators who would attempt to add roles to */etc/sudoers*. When this option is present, */etc/sudoers* does not even need to exist. Since this option tells **sudo** how to behave when no specific LDAP entries have been matched, this sudoOption is only meaningful for the cn=defaults section. This flag is *off* by default.

insults
: If set, **sudo** will insult users when they enter an incorrect password. This flag is *off* by default.

log_host
: If set, the host name will be logged in the (non-syslog) **sudo** log file. This flag is *off* by default.

log_input
: If set, **sudo** will run the command in a pseudo-tty and log all user input. If the standard input is not connected to the user's tty, due to I/O redirection or because the command is part of a pipeline, that input is also captured and stored in a separate log file. For more information, see the *I/O LOG FILES* section. This flag is *off* by default.

log_output
: If set, **sudo** will run the command in a pseudo-tty and log all output that is sent to the screen, similar to the script(1) command. For more information, see the *I/O LOG FILES* section. This flag is *off* by default.

log_year
: If set, the four-digit year will be logged in the (non-syslog) **sudo** log file. This flag is *off* by default.

long_otp_prompt
: When validating with a One Time Password (OTP) scheme such as **S/Key** or **OPIE**, a two-line prompt is used to make it easier to cut and paste the challenge to a local window. It's not as pretty as the default but some people find it more convenient. This flag is *off* by default.

mail_all_cmnds
: Send mail to the *mailto* user every time a user attempts to run a command via **sudo** (this includes **sudoedit**). No mail will be sent if the user runs **sudo** with the **-l** or **-v** option unless there is an authentication error and the *mail_badpass* flag is also set. This flag is *off* by default.

mail_always
: Send mail to the *mailto* user every time a user runs **sudo**. This flag is *off* by default.

mail_badpass
: Send mail to the *mailto* user if the user running **sudo** does not enter the correct password. If the command the user is attempting to run is not permitted by **sudoers** and one of the *mail_all_cmnds*, *mail_always*, *mail_no_host*, *mail_no_perms* or *mail_no_user* flags are set, this flag will have no effect. This flag is *off* by default.

mail_no_host
: If set, mail will be sent to the *mailto* user if the invoking user exists in the *sudoers* file, but is not allowed to run commands on the current host. This flag is *off* by default.

mail_no_perms
: If set, mail will be sent to the *mailto* user if the invoking user is allowed to use **sudo** but the command they are trying is not listed in their *sudoers* file entry or is explicitly denied. This flag is *off* by default.

mail_no_user
: If set, mail will be sent to the *mailto* user if the invoking user is not in the *sudoers* file. This flag is *on* by default.

match_group_by_gid
: By default, when matching groups, **sudoers** will first resolve all the user's group IDs to group names and then compare those group names to any group names listed in the *sudoers* file. This works well on systems where the number of groups listed in the *sudoers* file is larger than the number of groups a typical user belongs to. On systems where group lookups are slow, where users may belong to a large number of groups, and where the number of groups listed in the *sudoers* file is relatively small, it may be prohibitively expensive and running commands via **sudo** may take

longer than normal. On such systems it may be faster to use the *match_group_by_gid* flag to avoid resolving the user's group IDs to group names and instead resolve all group names listed in the *sudoers* file, matching by group ID instead of by group name. The *match_group_by_gid* flag has no effect when *sudoers* data is stored in LDAP. This flag is *off* by default.

This setting is only supported by version 1.8.18 or higher.

netgroup_tuple
: If set, netgroup lookups will be performed using the full netgroup tuple: host name, user name and domain (if one is set). Historically, **sudo** only matched the user name and domain for netgroups used in a User_List and only matched the host name and domain for netgroups used in a Host_List. This flag is *off* by default.

noexec
: If set, all commands run via **sudo** will behave as if the NOEXEC tag has been set, unless overridden by an EXEC tag. See the description of *EXEC and NOEXEC* above as well as the *Preventing shell escapes* section at the end of this manual. This flag is *off* by default.

pam_session
: On systems that use PAM for authentication, **sudo** will create a new PAM session for the command to be run in. Disabling *pam_session* may be needed on older PAM implementations or on operating systems where opening a PAM session changes the utmp or wtmp files. If PAM session support is disabled, resource limits may not be updated for the command being run. If *pam_session*, *pam_setcred*, and *use_pty* are disabled and I/O logging has not been configured, **sudo** will execute the command directly instead of running it as a child process. This flag is *on* by default.

This setting is only supported by version 1.8.7 or higher.

pam_setcred
: On systems that use PAM for authentication, **sudo** will attempt to establish credentials for the target user by default, if supported by the underlying authentication system. One example of a credential is a Kerberos ticket. If *pam_session*, *pam_setcred*, and *use_pty* are disabled and I/O logging has not been configured, **sudo** will execute the command directly instead of running it as a child process. This flag is *on* by default.

This setting is only supported by version 1.8.8 or higher.

passprompt_override
: The password prompt specified by *passprompt* will normally only be used if the

password prompt provided by systems such as PAM matches the string ‘‘Password:’’. If *passprompt_override* is set, *passprompt* will always be used. This flag is *off* by default.

path_info — Normally, **sudo** will tell the user when a command could not be found in their PATH environment variable. Some sites may wish to disable this as it could be used to gather information on the location of executables that the normal user does not have access to. The disadvantage is that if the executable is simply not in the user’s PATH, **sudo** will tell the user that they are not allowed to run it, which can be confusing. This flag is *on* by default.

preserve_groups — By default, **sudo** will initialize the group vector to the list of groups the target user is in. When *preserve_groups* is set, the user’s existing group vector is left unaltered. The real and effective group IDs, however, are still set to match the target user. This flag is *off* by default.

pwfeedback — By default, **sudo** reads the password like most other Unix programs, by turning off echo until the user hits the return (or enter) key. Some users become confused by this as it appears to them that **sudo** has hung at this point. When *pwfeedback* is set, **sudo** will provide visual feedback when the user presses a key. Note that this does have a security impact as an onlooker may be able to determine the length of the password being entered. This flag is *off* by default.

requiretty — If set, **sudo** will only run when the user is logged in to a real tty. When this flag is set, **sudo** can only be run from a login session and not via other means such as cron(8) or cgi-bin scripts. This flag is *off* by default.

root_sudo — If set, root is allowed to run **sudo** too. Disabling this prevents users from ‘‘chaining’’ **sudo** commands to get a root shell by doing something like ‘‘sudo sudo /bin/sh’’. Note, however, that turning off *root_sudo* will also prevent root from running **sudoedit**. Disabling *root_sudo* provides no real additional security; it exists purely for historical reasons. This flag is *on* by default.

rootpw — If set, **sudo** will prompt for the root password instead of the password of the invoking user when running a command or editing a file. This flag is *off* by default.

runaspw — If set, **sudo** will prompt for the password of the user defined by the *runas_default* option (defaults to root) instead of the password of the invoking user when running a command or editing a file. This flag is *off* by default.

set_home
: If enabled and **sudo** is invoked with the **-s** option the HOME environment variable will be set to the home directory of the target user (which is root unless the **-u** option is used). This effectively makes the **-s** option imply **-H**. Note that HOME is already set when the *env_reset* option is enabled, so *set_home* is only effective for configurations where either *env_reset* is disabled or HOME is present in the *env_keep* list. This flag is *off* by default.

set_logname
: Normally, **sudo** will set the LOGNAME, USER and USERNAME environment variables to the name of the target user (usually root unless the **-u** option is given). However, since some programs (including the RCS revision control system) use LOGNAME to determine the real identity of the user, it may be desirable to change this behavior. This can be done by negating the set_logname option. Note that *set_logname* will have no effect if the *env_reset* option has not been disabled and the *env_keep* list contains LOGNAME, USER or USERNAME. This flag is *on* by default.

set_utmp
: When enabled, **sudo** will create an entry in the utmp (or utmpx) file when a pseudo-tty is allocated. A pseudo-tty is allocated by **sudo** when the *log_input*, *log_output* or *use_pty* flags are enabled. By default, the new entry will be a copy of the user's existing utmp entry (if any), with the tty, time, type and pid fields updated. This flag is *on* by default.

setenv
: Allow the user to disable the *env_reset* option from the command line via the **-E** option. Additionally, environment variables set via the command line are not subject to the restrictions imposed by *env_check*, *env_delete*, or *env_keep*. As such, only trusted users should be allowed to set variables in this manner. This flag is *off* by default.

shell_noargs
: If set and **sudo** is invoked with no arguments it acts as if the **-s** option had been given. That is, it runs a shell as root (the shell is determined by the SHELL environment variable if it is set, falling back on the shell listed in the invoking user's /etc/passwd entry if not). This flag is *off* by default.

stay_setuid
: Normally, when **sudo** executes a command the real and effective UIDs are set to the target user (root by default). This option changes that behavior such that the real UID is left as the invoking user's UID. In other words, this makes **sudo** act as a setuid wrapper. This can be useful on systems that disable some potentially dangerous functionality when a program is run setuid. This option is only effective on systems that support either the setreuid(2) or setresuid(2) system call. This flag is *off* by default.

sudoedit_checkdir
: If set, **sudoedit** will check all directory components of the path to be edited for writability by the invoking user. Symbolic links will not be followed in writable directories and **sudoedit** will refuse to edit a file located in a writable directory. These restrictions are not enforced when **sudoedit** is run by root. On some systems, if all directory components of the path to be edited are not readable by the target user, **sudoedit** will be unable to edit the file. This flag is *on* by default.

  This setting was first introduced in version 1.8.15 but initially suffered from a race condition. The check for symbolic links in writable intermediate directories was added in version 1.8.16.

sudoedit_follow
: By default, **sudoedit** will not follow symbolic links when opening files. The *sudoedit_follow* option can be enabled to allow **sudoedit** to open symbolic links. It may be overridden on a per-command basis by the *FOLLOW* and *NOFOLLOW* tags. This flag is *off* by default.

  This setting is only supported by version 1.8.15 or higher.

targetpw
: If set, **sudo** will prompt for the password of the user specified by the **-u** option (defaults to root) instead of the password of the invoking user when running a command or editing a file. Note that this flag precludes the use of a uid not listed in the passwd database as an argument to the **-u** option. This flag is *off* by default.

tty_tickets
: If set, users must authenticate on a per-tty basis. With this flag enabled, **sudo** will use a separate record in the time stamp file for each tty. If disabled, a single record is used for all login sessions. This flag is *on* by default.

umask_override
: If set, **sudo** will set the umask as specified in the *sudoers* file without modification. This makes it possible to specify a umask in the *sudoers* file that is more permissive than the user's own umask and matches historical behavior. If *umask_override* is not set, **sudo** will set the umask to be the union of the user's umask and what is specified in *sudoers*. This flag is *off* by default.

use_loginclass
: If set, **sudo** will apply the defaults specified for the target user's login class if one exists. Only available if **sudo** is configured with the --with-logincap option. This flag is *off* by default.

use_netgroups
: If set, netgroups (prefixed with '+'), may be used in place of a user or host. For LDAP-based sudoers, netgroup support requires an expensive substring match on the server unless the **NETGROUP_BASE** directive is present in the */etc/ldap.conf*

file. If netgroups are not needed, this option can be disabled to reduce the load on the LDAP server. This flag is *on* by default.

use_pty
: If set, **sudo** will run the command in a pseudo-pty even if no I/O logging is being gone. A malicious program run under **sudo** could conceivably fork a background process that retains to the user's terminal device after the main program has finished executing. Use of this option will make that impossible. This flag is *off* by default.

utmp_runas
: If set, **sudo** will store the name of the runas user when updating the utmp (or utmpx) file. By default, **sudo** stores the name of the invoking user. This flag is *off* by default.

visiblepw
: By default, **sudo** will refuse to run if the user must enter a password but it is not possible to disable echo on the terminal. If the *visiblepw* flag is set, **sudo** will prompt for a password even when it would be visible on the screen. This makes it possible to run things like ''ssh somehost sudo ls'' since by default, ssh(1) does not allocate a tty when running a command. This flag is *off* by default.

Integers:

closefrom
: Before it executes a command, **sudo** will close all open file descriptors other than standard input, standard output and standard error (ie: file descriptors 0-2). The *closefrom* option can be used to specify a different file descriptor at which to start closing. The default is 3.

maxseq
: The maximum sequence number that will be substituted for the ''%{seq}'' escape in the I/O log file (see the *iolog_dir* description above for more information). While the value substituted for ''%{seq}'' is in base 36, *maxseq* itself should be expressed in decimal. Values larger than 2176782336 (which corresponds to the base 36 sequence number ''ZZZZZZ'') will be silently truncated to 2176782336. The default value is 2176782336.

  Once the local sequence number reaches the value of *maxseq*, it will ''roll over'' to zero, after which **sudoers** will truncate and re-use any existing I/O log path names.

  This setting is only supported by version 1.8.7 or higher.

passwd_tries
: The number of tries a user gets to enter his/her password before **sudo** logs the failure and exits. The default is 3.

Integers that can be used in a boolean context:

loglinelen
: Number of characters per line for the file log. This value is used to decide when to wrap lines for nicer log files. This has no effect on the syslog log file, only the file log. The default is 80 (use 0 or negate the option to disable word wrap).

passwd_timeout
: Number of minutes before the **sudo** password prompt times out, or 0 for no timeout. The timeout may include a fractional component if minute granularity is insufficient, for example 2.5. The default is 5.

timestamp_timeout
: Number of minutes that can elapse before **sudo** will ask for a passwd again. The timeout may include a fractional component if minute granularity is insufficient, for example 2.5. The default is 5. Set this to 0 to always prompt for a password. If set to a value less than 0 the user's time stamp will not expire until the system is rebooted. This can be used to allow users to create or delete their own time stamps via ‘‘sudo -v’’ and ‘‘sudo -k’’ respectively.

umask
: Umask to use when running the command. Negate this option or set it to 0777 to preserve the user's umask. The actual umask that is used will be the union of the user's umask and the value of the *umask* option, which defaults to 0022. This guarantees that **sudo** never lowers the umask when running a command. Note: on systems that use PAM, the default PAM configuration may specify its own umask which will override the value set in *sudoers*.

Strings:

badpass_message
: Message that is displayed if a user enters an incorrect password. The default is Sorry, try again. unless insults are enabled.

editor
: A colon (‘:’) separated list of editors allowed to be used with **visudo**. **visudo** will choose the editor that matches the user's EDITOR or VISUAL environment variable if possible, or the first editor in the list that exists and is executable. Note that the EDITOR and VISUAL environment variables are not preserved by default when the *env_reset* option is enabled. The default is *vi*.

iolog_dir
: The top-level directory to use when constructing the path name for the input/output log directory. Only used if the *log_input* or *log_output* options are enabled or when the LOG_INPUT or LOG_OUTPUT tags are present for a command. The session sequence number, if any, is stored in the directory. The default is */var/log/sudo-io*.

The following percent (‘%’) escape sequences are supported:

%{seq}
: expanded to a monotonically increasing base-36 sequence number, such as 0100A5, where every two digits are used to form a new directory, e.g. *01/00/A5*

%{user}
: expanded to the invoking user’s login name

%{group}
: expanded to the name of the invoking user’s real group ID

%{runas_user}
: expanded to the login name of the user the command will be run as (e.g. root)

%{runas_group}
: expanded to the group name of the user the command will be run as (e.g. wheel)

%{hostname}
: expanded to the local host name without the domain name

%{command}
: expanded to the base name of the command being run

In addition, any escape sequences supported by the system’s strftime(3) function will be expanded.

To include a literal ‘%’ character, the string ‘%%’ should be used.

iolog_file
: The path name, relative to *iolog_dir*, in which to store input/output logs when the *log_input* or *log_output* options are enabled or when the LOG_INPUT or LOG_OUTPUT tags are present for a command. Note that *iolog_file* may contain directory components. The default is ‘‘%{seq}’’.

  See the *iolog_dir* option above for a list of supported percent (‘%’) escape sequences.

In addition to the escape sequences, path names that end in six or more Xs will have the Xs replaced with a unique combination of digits and letters, similar to the mktemp(3) function.

If the path created by concatenating *iolog_dir* and *iolog_file* already exists, the existing I/O log file will be truncated and overwritten unless *iolog_file* ends in six or more Xs.

lecture_status_dir The directory in which **sudo** stores per-user lecture status files. Once a user has received the lecture, a zero-length file is created in this directory so that **sudo** will not lecture the user again. This directory should *not* be cleared when the system reboots. The default is */var/adm/sudo/lectured*.

limitprivs The default Solaris limit privileges to use when constructing a new privilege set for a command. This bounds all privileges of the executing process. The default limit privileges may be overridden on a per-command basis in *sudoers*. This option is only available if **sudoers** is built on Solaris 10 or higher.

mailsub Subject of the mail sent to the *mailto* user. The escape %h will expand to the host name of the machine. Default is ‘‘*** SECURITY information for %h ***’’.

noexec_file As of **sudo** version 1.8.1 this option is no longer supported. The path to the noexec file should now be set in the sudo.conf(5) file.

pam_login_service On systems that use PAM for authentication, this is the service name used when the **-i** option is specified. The default value is ‘‘sudo’’. See the description of *pam_service* for more information.

This setting is only supported by version 1.8.8 or higher.

pam_service On systems that use PAM for authentication, the service name specifies the PAM policy to apply. This usually corresponds to an entry in the *pam.conf* file or a file in the */etc/pam.d* directory. The default value is ‘‘sudo’’.

This setting is only supported by version 1.8.8 or higher.

passprompt The default prompt to use when asking for a password; can be overridden via the **-p** option or the SUDO_PROMPT environment variable. The following percent (‘%’) escape sequences are supported:

%H expanded to the local host name including the domain name (only if the machine's host name is fully qualified or the *fqdn* option is set)

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

%p expanded to the user whose password is being asked for (respects the *rootpw*, *targetpw* and *runaspw* flags in *sudoers*)

%U expanded to the login name of the user the command will be run as (defaults to root)

%u expanded to the invoking user's login name

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

The default value is ''Password:''.

privs The default Solaris privileges to use when constructing a new privilege set for a command. This is passed to the executing process via the inherited privilege set, but is bounded by the limit privileges. If the *privs* option is specified but the *limitprivs* option is not, the limit privileges of the executing process is set to *privs*. The default privileges may be overridden on a per-command basis in *sudoers*. This option is only available if **sudoers** is built on Solaris 10 or higher.

role The default SELinux role to use when constructing a new security context to run the command. The default role may be overridden on a per-command basis in the *sudoers* file or via command line options. This option is only available when **sudo** is built with SELinux support.

runas_default The default user to run commands as if the **-u** option is not specified on the command line. This defaults to root.

syslog_badpri Syslog priority to use when user authenticates unsuccessfully. Defaults to alert.

The following syslog priorities are supported: **alert**, **crit**, **debug**, **emerg**, **err**, **info**, **notice**, and **warning**.

syslog_goodpri Syslog priority to use when user authenticates successfully. Defaults to notice.

See *syslog_badpri* for the list of supported syslog priorities.

sudoers_locale Locale to use when parsing the sudoers file, logging commands, and sending email. Note that changing the locale may affect how sudoers is interpreted. Defaults to ‘‘C’’.

timestampdir The directory in which **sudo** stores its time stamp files. This directory should be cleared when the system reboots. The default is */var/run/sudo/ts*.

timestampowner The owner of the lecture status directory, time stamp directory and all files stored therein. The default is root.

type The default SELinux type to use when constructing a new security context to run the command. The default type may be overridden on a per-command basis in the *sudoers* file or via command line options. This option is only available when **sudo** is built with SELinux support.

Strings that can be used in a boolean context:

env_file The *env_file* option specifies the fully qualified path to a file containing variables to be set in the environment of the program being run. Entries in this file should either be of the form ‘‘VARIABLE=value’’ or ‘‘export VARIABLE=value’’. The value may optionally be surrounded by single or double quotes. Variables in this file are subject to other **sudo** environment settings such as *env_keep* and *env_check*.

exempt_group Users in this group are exempt from password and PATH requirements. The group name specified should not include a % prefix. This is not set by default.

group_plugin A string containing a **sudoers** group plugin with optional arguments. The string should consist of the plugin path, either fully-qualified or relative to the */usr/local/libexec/sudo* directory, followed by any configuration arguments the plugin requires. These arguments (if any) will be passed to the plugin’s initialization function. If arguments are present, the string must be enclosed in double quotes (""). For more information see GROUP PROVIDER PLUGINS.

lecture This option controls when a short lecture will be printed along with the password prompt. It has the following possible values:

always Always lecture the user.

never Never lecture the user.

once Only lecture the user the first time they run **sudo**.

If no value is specified, a value of *once* is implied. Negating the option results in a value of *never* being used. The default value is *once*.

lecture_file Path to a file containing an alternate **sudo** lecture that will be used in place of the standard lecture if the named file exists. By default, **sudo** uses a built-in lecture.

listpw This option controls when a password will be required when a user runs **sudo** with the **-l** option. It has the following possible values:

all All the user's *sudoers* file entries for the current host must have the NOPASSWD flag set to avoid entering a password.

always The user must always enter a password to use the **-l** option.

any At least one of the user's *sudoers* file entries for the current host must have the NOPASSWD flag set to avoid entering a password.

never The user need never enter a password to use the **-l** option.

If no value is specified, a value of *any* is implied. Negating the option results in a value of *never* being used. The default value is *any*.

logfile Path to the **sudo** log file (not the syslog log file). Setting a path turns on logging to a file; negating this option turns it off. By default, **sudo** logs via syslog.

mailerflags Flags to use when invoking mailer. Defaults to **-t**.

mailerpath Path to mail program used to send warning mail. Defaults to the path to sendmail found at configure time.

mailfrom Address to use for the ‘‘from’’ address when sending warning and error mail. The address should be enclosed in double quotes ("") to protect against **sudo** interpreting the @ sign. Defaults to the name of the user running **sudo**.

mailto Address to send warning and error mail to. The address should be enclosed in double quotes ("") to protect against **sudo** interpreting the @ sign. Defaults to root.

secure_path Path used for every command run from **sudo**. If you don't trust the people running **sudo**

to have a sane PATH environment variable you may want to use this. Another use is if you want to have the ‘‘root path’’ be separate from the ‘‘user path’’. Users in the group specified by the *exempt_group* option are not affected by *secure_path*. This option is not set by default.

syslog Syslog facility if syslog is being used for logging (negate to disable syslog logging). Defaults to auth.

The following syslog facilities are supported: **authpriv** (if your OS supports it), **auth**, **daemon**, **user**, **local0**, **local1**, **local2**, **local3**, **local4**, **local5**, **local6**, and **local7**.

verifypw This option controls when a password will be required when a user runs **sudo** with the **-v** option. It has the following possible values:

all All the user’s *sudoers* file entries for the current host must have the NOPASSWD flag set to avoid entering a password.

always The user must always enter a password to use the **-v** option.

any At least one of the user’s *sudoers* file entries for the current host must have the NOPASSWD flag set to avoid entering a password.

never The user need never enter a password to use the **-v** option.

If no value is specified, a value of *all* is implied. Negating the option results in a value of *never* being used. The default value is *all*.

Lists that can be used in a boolean context:

env_check Environment variables to be removed from the user’s environment unless they are considered ‘‘safe’’. For all variables except TZ, ‘‘safe’’ means that the variable’s value does not contain any ‘%’ or ‘/’ characters. This can be used to guard against printf-style format vulnerabilities in poorly-written programs. The TZ variable is considered unsafe if any of the following are true:

- It consists of a fully-qualified path name, optionally prefixed with a colon (‘:’), that does not match the location of the *zoneinfo* directory.
- It contains a .. path element.

- It contains white space or non-printable characters.
- It is longer than the value of PATH_MAX.

The argument may be a double-quoted, space-separated list or a single value without double-quotes. The list can be replaced, added to, deleted from, or disabled by using the =, +=, -=, and ! operators respectively. Regardless of whether the env_reset option is enabled or disabled, variables specified by env_check will be preserved in the environment if they pass the aforementioned check. The global list of environment variables to check is displayed when **sudo** is run by root with the **-V** option.

env_delete — Environment variables to be removed from the user's environment when the *env_reset* option is not in effect. The argument may be a double-quoted, space-separated list or a single value without double-quotes. The list can be replaced, added to, deleted from, or disabled by using the =, +=, -=, and ! operators respectively. The global list of environment variables to remove is displayed when **sudo** is run by root with the **-V** option. Note that many operating systems will remove potentially dangerous variables from the environment of any setuid process (such as **sudo**).

env_keep — Environment variables to be preserved in the user's environment when the *env_reset* option is in effect. This allows fine-grained control over the environment **sudo**-spawned processes will receive. The argument may be a double-quoted, space-separated list or a single value without double-quotes. The list can be replaced, added to, deleted from, or disabled by using the =, +=, -=, and ! operators respectively. The global list of variables to keep is displayed when **sudo** is run by root with the **-V** option.

GROUP PROVIDER PLUGINS

The **sudoers** plugin supports its own plugin interface to allow non-Unix group lookups which can query a group source other than the standard Unix group database. This can be used to implement support for the nonunix_group syntax described earlier.

Group provider plugins are specified via the *group_plugin* Defaults setting. The argument to *group_plugin* should consist of the plugin path, either fully-qualified or relative to the */usr/local/libexec/sudo* directory, followed by any configuration options the plugin requires. These options (if specified) will be passed to the plugin's initialization function. If options are present, the string must be enclosed in double quotes ("").

The following group provider plugins are installed by default:

group_file
: The *group_file* plugin supports an alternate group file that uses the same syntax as the */etc/group* file. The path to the group file should be specified as an option to the plugin. For example, if the group file to be used is */etc/sudo-group*:

  Defaults group_plugin="group_file.so /etc/sudo-group"

system_group
: The *system_group* plugin supports group lookups via the standard C library functions **getgrnam**() and **getgrid**(). This plugin can be used in instances where the user belongs to groups not present in the user's supplemental group vector. This plugin takes no options:

  Defaults group_plugin=system_group.so

The group provider plugin API is described in detail in sudo_plugin(8).

LOG FORMAT

sudoers can log events using either syslog(3) or a simple log file. The log format is almost identical in both cases.

Accepted command log entries

Commands that sudo runs are logged using the following format (split into multiple lines for readability):

```
date hostname progname: username : TTY=ttyname ; PWD=cwd ; \
  USER=runasuser ; GROUP=runasgroup ; TSID=logid ; \
  ENV=env_vars COMMAND=command
```

Where the fields are as follows:

date
: The date the command was run. Typically, this is in the format ‘‘MMM, DD, HH:MM:SS’’. If logging via syslog(3), the actual date format is controlled by the syslog daemon. If logging to a file and the *log_year* option is enabled, the date will also include the year.

hostname
: The name of the host **sudo** was run on. This field is only present when logging via syslog(3).

progname The name of the program, usually *sudo* or *sudoedit*. This field is only present when logging via syslog(3).

username The login name of the user who ran **sudo**.

ttyname The short name of the terminal (e.g. ‘‘console’’, ‘‘tty01’’, or ‘‘pts/0’’) **sudo** was run on, or ‘‘unknown’’ if there was no terminal present.

cwd The current working directory that **sudo** was run in.

runasuser The user the command was run as.

runasgroup The group the command was run as if one was specified on the command line.

logid An I/O log identifier that can be used to replay the command’s output. This is only present when the *log_input* or *log_output* option is enabled.

env_vars A list of environment variables specified on the command line, if specified.

command The actual command that was executed.

Messages are logged using the locale specified by *sudoers_locale*, which defaults to the ‘‘C’’ locale.

Denied command log entries

If the user is not allowed to run the command, the reason for the denial will follow the user name. Possible reasons include:

user NOT in sudoers
The user is not listed in the *sudoers* file.

user NOT authorized on host
The user is listed in the *sudoers* file but is not allowed to run commands on the host.

command not allowed
The user is listed in the *sudoers* file for the host but they are not allowed to run the specified command.

3 incorrect password attempts
The user failed to enter their password after 3 tries. The actual number of tries will vary based on the number of failed attempts and the value of the *passwd_tries* option.

a password is required
sudo's **-n** option was specified but a password was required.

sorry, you are not allowed to set the following environment variables
The user specified environment variables on the command line that were not allowed by *sudoers*.

Error log entries

If an error occurs, **sudoers** will log a message and, in most cases, send a message to the administrator via email. Possible errors include:

parse error in /etc/sudoers near line N
sudoers encountered an error when parsing the specified file. In some cases, the actual error may be one line above or below the line number listed, depending on the type of error.

problem with defaults entries
The *sudoers* file contains one or more unknown Defaults settings. This does not prevent **sudo** from running, but the *sudoers* file should be checked using **visudo**.

timestamp owner (username): No such user
The time stamp directory owner, as specified by the *timestampowner* setting, could not be found in the password database.

unable to open/read /etc/sudoers
The *sudoers* file could not be opened for reading. This can happen when the *sudoers* file is located on a remote file system that maps user ID 0 to a different value. Normally, **sudoers** tries to open the *sudoers* file using group permissions to avoid this problem. Consider either changing the ownership of */etc/sudoers* or adding an argument like ''sudoers_uid=N'' (where 'N' is the user ID that owns the *sudoers* file) to the end of the **sudoers** Plugin line in the sudo.conf(5) file.

unable to stat /etc/sudoers
The */etc/sudoers* file is missing.

/etc/sudoers is not a regular file
The */etc/sudoers* file exists but is not a regular file or symbolic link.

/etc/sudoers is owned by uid N, should be 0
The *sudoers* file has the wrong owner. If you wish to change the *sudoers* file owner, please add ''sudoers_uid=N'' (where 'N' is the user ID that owns the *sudoers* file) to the **sudoers** Plugin line in the sudo.conf(5) file.

/etc/sudoers is world writable
The permissions on the *sudoers* file allow all users to write to it. The *sudoers* file must not be world-writable, the default file mode is 0440 (readable by owner and group, writable by none). The default mode may be changed via the ‘‘sudoers_mode’’ option to the **sudoers** Plugin line in the sudo.conf(5) file.

/etc/sudoers is owned by gid N, should be 1
The *sudoers* file has the wrong group ownership. If you wish to change the *sudoers* file group ownership, please add ‘‘sudoers_gid=N’’ (where ‘N’ is the group ID that owns the *sudoers* file) to the **sudoers** Plugin line in the sudo.conf(5) file.

unable to open /var/run/sudo/ts/username
sudoers was unable to read or create the user’s time stamp file. This can happen when *timestampowner* is set to a user other than root and the mode on */var/run/sudo* is not searchable by group or other. The default mode for */var/run/sudo* is 0711.

unable to write to /var/run/sudo/ts/username
sudoers was unable to write to the user’s time stamp file.

/var/run/sudo/ts is owned by uid X, should be Y
The time stamp directory is owned by a user other than *timestampowner*. This can occur when the value of *timestampowner* has been changed. **sudoers** will ignore the time stamp directory until the owner is corrected.

/var/run/sudo/ts is group writable
The time stamp directory is group-writable; it should be writable only by *timestampowner*. The default mode for the time stamp directory is 0700. **sudoers** will ignore the time stamp directory until the mode is corrected.

Notes on logging via syslog

By default, **sudoers** logs messages via syslog(3). The *date*, *hostname*, and *progname* fields are added by the syslog daemon, not **sudoers** itself. As such, they may vary in format on different systems.

On most systems, syslog(3) has a relatively small log buffer. To prevent the command line arguments from being truncated, **sudoers** will split up log messages that are larger than 960 characters (not including the date, hostname, and the string ‘‘sudo’’). When a message is split, additional parts will include the string ‘‘(command continued)’’ after the user name and before the continued command line arguments.

Notes on logging to a file

If the *logfile* option is set, **sudoers** will log to a local file, such as */var/log/sudo*. When logging to a file, **sudoers** uses a format similar to syslog(3), with a few important differences:

1. The *progname* and *hostname* fields are not present.

2. If the *log_year* option is enabled, the date will also include the year.

3. Lines that are longer than *loglinelen* characters (80 by default) are word-wrapped and continued on the next line with a four character indent. This makes entries easier to read for a human being, but makes it more difficult to use grep(1) on the log files. If the *loglinelen* option is set to 0 (or negated with a '!'), word wrap will be disabled.

I/O LOG FILES

When I/O logging is enabled, **sudo** will run the command in a pseudo-tty and log all user input and/or output. I/O is logged to the directory specified by the *iolog_dir* option (*/var/log/sudo-io* by default) using a unique session ID that is included in the **sudo** log line, prefixed with ''TSID=''. The *iolog_file* option may be used to control the format of the session ID.

Each I/O log is stored in a separate directory that contains the following files:

log a text file containing the time the command was run, the name of the user who ran **sudo**, the name of the target user, the name of the target group (optional), the terminal that **sudo** was run from, the number of rows and columns of the terminal, the working directory the command was run from and the path name of the command itself (with arguments if present)

timing a log of the amount of time between, and the number of bytes in, each I/O log entry (used for session playback)

ttyin input from the user's tty (what the user types)

stdin input from a pipe or file

ttyout output from the pseudo-tty (what the command writes to the screen)

stdout standard output to a pipe or redirected to a file

stderr standard error to a pipe or redirected to a file

All files other than *log* are compressed in gzip format unless the *compress_io* option has been disabled. Due to buffering, the I/O log data will not be complete until the **sudo** command has completed. The

output portion of an I/O log file can be viewed with the sudoreplay(8) utility, which can also be used to list or search the available logs.

Note that user input may contain sensitive information such as passwords (even if they are not echoed to the screen), which will be stored in the log file unencrypted. In most cases, logging the command output via *log_output* or LOG_OUTPUT is all that is required.

Since each session's I/O logs are stored in a separate directory, traditional log rotation utilities cannot be used to limit the number of I/O logs. The simplest way to limit the number of I/O is by setting the *maxseq* option to the maximum number of logs you wish to store. Once the I/O log sequence number reaches *maxseq*, it will be reset to zero and **sudoers** will truncate and re-use any existing I/O logs.

FILES

/etc/sudo.conf	Sudo front end configuration
/etc/sudoers	List of who can run what
/etc/group	Local groups file
/etc/netgroup	List of network groups
/var/log/sudo-io	I/O log files
/var/run/sudo/ts	Directory containing time stamps for the **sudoers** security policy
/var/adm/sudo/lectured	Directory containing lecture status files for the **sudoers** security policy
/etc/environment	Initial environment for **-i** mode on AIX and Linux systems

EXAMPLES

Below are example *sudoers* file entries. Admittedly, some of these are a bit contrived. First, we allow a few environment variables to pass and then define our *aliases*:

```
# Run X applications through sudo; HOME is used to find the
# .Xauthority file.  Note that other programs use HOME to find
# configuration files and this may lead to privilege escalation!
Defaults env_keep += "DISPLAY HOME"

# User alias specification
User_Alias      FULLTIMERS = millert, mikef, dowdy
```

```
User_Alias      PARTTIMERS = bostley, jwfox, crawl
User_Alias      WEBMASTERS = will, wendy, wim

# Runas alias specification
Runas_Alias     OP = root, operator
Runas_Alias     DB = oracle, sybase
Runas_Alias     ADMINGRP = adm, oper

# Host alias specification
Host_Alias      SPARC = bigtime, eclipse, moet, anchor :\
                SGI = grolsch, dandelion, black :\
                ALPHA = widget, thalamus, foobar :\
                HPPA = boa, nag, python
Host_Alias      CUNETS = 128.138.0.0/255.255.0.0
Host_Alias      CSNETS = 128.138.243.0, 128.138.204.0/24, 128.138.242.0
Host_Alias      SERVERS = master, mail, www, ns
Host_Alias      CDROM = orion, perseus, hercules

# Cmnd alias specification
Cmnd_Alias      DUMPS = /usr/bin/mt, /usr/sbin/dump, /usr/sbin/rdump,\
                /usr/sbin/restore, /usr/sbin/rrestore,\
                sha224:0GomF8mNN3wlDt1HD9XldjJ3SNgpFdbjO1+NsQ== \
                /home/operator/bin/start_backups
Cmnd_Alias      KILL = /usr/bin/kill
Cmnd_Alias      PRINTING = /usr/sbin/lpc, /usr/bin/lprm
Cmnd_Alias      SHUTDOWN = /usr/sbin/shutdown
Cmnd_Alias      HALT = /usr/sbin/halt
Cmnd_Alias      REBOOT = /usr/sbin/reboot
Cmnd_Alias      SHELLS = /usr/bin/sh, /usr/bin/csh, /usr/bin/ksh,\
                 /usr/local/bin/tcsh, /usr/bin/rsh,\
                 /usr/local/bin/zsh
Cmnd_Alias      SU = /usr/bin/su
Cmnd_Alias      PAGERS = /usr/bin/more, /usr/bin/pg, /usr/bin/less
```

Here we override some of the compiled in default values. We want **sudo** to log via syslog(3) using the *auth* facility in all cases. We don’t want to subject the full time staff to the **sudo** lecture, user **millert** need not give a password, and we don’t want to reset the LOGNAME, USER or USERNAME environment variables when running commands as root. Additionally, on the machines in the *SERVERS* Host_Alias, we keep an additional local log file and make sure we log the year in each log line since the log entries will be kept around for several years. Lastly, we disable shell escapes for the

commands in the PAGERS Cmnd_Alias (*/usr/bin/more*, */usr/bin/pg* and */usr/bin/less*). Note that this will not effectively constrain users with **sudo ALL** privileges.

```
# Override built-in defaults
Defaults        syslog=auth
Defaults>root       !set_logname
Defaults:FULLTIMERS !lecture
Defaults:millert    !authenticate
Defaults@SERVERS    log_year, logfile=/var/log/sudo.log
Defaults!PAGERS     noexec
```

The *User specification* is the part that actually determines who may run what.

```
root        ALL = (ALL) ALL
%wheel          ALL = (ALL) ALL
```

We let **root** and any user in group **wheel** run any command on any host as any user.

```
FULLTIMERS      ALL = NOPASSWD: ALL
```

Full time sysadmins (**millert**, **mikef**, and **dowdy**) may run any command on any host without authenticating themselves.

```
PARTTIMERS      ALL = ALL
```

Part time sysadmins **bostley**, **jwfox**, and **crawl**) may run any command on any host but they must authenticate themselves first (since the entry lacks the NOPASSWD tag).

```
jack        CSNETS = ALL
```

The user **jack** may run any command on the machines in the *CSNETS* alias (the networks 128.138.243.0, 128.138.204.0, and 128.138.242.0). Of those networks, only 128.138.204.0 has an explicit netmask (in CIDR notation) indicating it is a class C network. For the other networks in *CSNETS*, the local machine's netmask will be used during matching.

```
lisa        CUNETS = ALL
```

The user **lisa** may run any command on any host in the *CUNETS* alias (the class B network 128.138.0.0).

```
operator   ALL = DUMPS, KILL, SHUTDOWN, HALT, REBOOT, PRINTING,\
           sudoedit /etc/printcap, /usr/oper/bin/
```

The **operator** user may run commands limited to simple maintenance. Here, those are commands related to backups, killing processes, the printing system, shutting down the system, and any commands in the directory */usr/oper/bin/*. Note that one command in the DUMPS Cmnd_Alias includes a sha224 digest, */home/operator/bin/start_backups*. This is because the directory containing the script is writable by the operator user. If the script is modified (resulting in a digest mismatch) it will no longer be possible to run it via **sudo**.

```
joe        ALL = /usr/bin/su operator
```

The user **joe** may only su(1) to operator.

```
pete       HPPA = /usr/bin/passwd [A-Za-z]*, !/usr/bin/passwd root
```

```
%opers          ALL = (: ADMINGRP) /usr/sbin/
```

Users in the **opers** group may run commands in */usr/sbin/* as themselves with any group in the *ADMINGRP* Runas_Alias (the **adm** and **oper** groups).

The user **pete** is allowed to change anyone's password except for root on the *HPPA* machines. Because command line arguments are matched as a single, concatenated string, the '*' wildcard will match *multiple* words. This example assumes that passwd(1) does not take multiple user names on the command line. Note that on GNU systems, options to passwd(1) may be specified after the user argument. As a result, this rule will also allow:

```
    passwd username --expire
```

which may not be desirable.

```
bob        SPARC = (OP) ALL : SGI = (OP) ALL
```

The user **bob** may run anything on the *SPARC* and *SGI* machines as any user listed in the *OP* Runas_Alias (**root** and **operator**.)

```
jim        +biglab = ALL
```

The user **jim** may run any command on machines in the *biglab* netgroup. **sudo** knows that ''biglab'' is a netgroup due to the '+' prefix.

+secretaries ALL = PRINTING, /usr/bin/adduser, /usr/bin/rmuser

Users in the **secretaries** netgroup need to help manage the printers as well as add and remove users, so they are allowed to run those commands on all machines.

fred ALL = (DB) NOPASSWD: ALL

The user **fred** can run commands as any user in the *DB* Runas_Alias (**oracle** or **sybase**) without giving a password.

john ALPHA = /usr/bin/su [!-]*, !/usr/bin/su *root*

On the *ALPHA* machines, user **john** may su to anyone except root but he is not allowed to specify any options to the su(1) command.

jen ALL, !SERVERS = ALL

The user **jen** may run any command on any machine except for those in the *SERVERS* Host_Alias (master, mail, www and ns).

jill SERVERS = /usr/bin/, !SU, !SHELLS

For any machine in the *SERVERS* Host_Alias, **jill** may run any commands in the directory */usr/bin/* except for those commands belonging to the *SU* and *SHELLS* Cmnd_Aliases. While not specifically mentioned in the rule, the commands in the *PAGERS* Cmnd_Alias all reside in */usr/bin* and have the *noexec* option set.

steve CSNETS = (operator) /usr/local/op_commands/

The user **steve** may run any command in the directory /usr/local/op_commands/ but only as user operator.

matt valkyrie = KILL

On his personal workstation, valkyrie, **matt** needs to be able to kill hung processes.

WEBMASTERS www = (www) ALL, (root) /usr/bin/su www

On the host www, any user in the *WEBMASTERS* User_Alias (will, wendy, and wim), may run any command as user www (which owns the web pages) or simply su(1) to www.

```
ALL     CDROM = NOPASSWD: /sbin/umount /CDROM,\
        /sbin/mount -o nosuid\,nodev /dev/cd0a /CDROM
```

Any user may mount or unmount a CD-ROM on the machines in the CDROM Host_Alias (orion, perseus, hercules) without entering a password. This is a bit tedious for users to type, so it is a prime candidate for encapsulating in a shell script.

SECURITY NOTES

Limitations of the '!' operator

It is generally not effective to ''subtract'' commands from **ALL** using the '!' operator. A user can trivially circumvent this by copying the desired command to a different name and then executing that. For example:

```
bill   ALL = ALL, !SU, !SHELLS
```

Doesn't really prevent **bill** from running the commands listed in *SU* or *SHELLS* since he can simply copy those commands to a different name, or use a shell escape from an editor or other program. Therefore, these kind of restrictions should be considered advisory at best (and reinforced by policy).

In general, if a user has sudo **ALL** there is nothing to prevent them from creating their own program that gives them a root shell (or making their own copy of a shell) regardless of any '!' elements in the user specification.

Security implications of *fast_glob*

If the *fast_glob* option is in use, it is not possible to reliably negate commands where the path name includes globbing (aka wildcard) characters. This is because the C library's fnmatch(3) function cannot resolve relative paths. While this is typically only an inconvenience for rules that grant privileges, it can result in a security issue for rules that subtract or revoke privileges.

For example, given the following *sudoers* file entry:

```
john ALL = /usr/bin/passwd [a-zA-Z0-9]*, /usr/bin/chsh [a-zA-Z0-9]*,\
        /usr/bin/chfn [a-zA-Z0-9]*, !/usr/bin/* root
```

User **john** can still run /usr/bin/passwd root if *fast_glob* is enabled by changing to */usr/bin* and running ./passwd root instead.

Preventing shell escapes

Once **sudo** executes a program, that program is free to do whatever it pleases, including run other programs. This can be a security issue since it is not uncommon for a program to allow shell escapes,

which lets a user bypass **sudo**'s access control and logging. Common programs that permit shell escapes include shells (obviously), editors, paginators, mail and terminal programs.

There are two basic approaches to this problem:

restrict Avoid giving users access to commands that allow the user to run arbitrary commands. Many editors have a restricted mode where shell escapes are disabled, though **sudoedit** is a better solution to running editors via **sudo**. Due to the large number of programs that offer shell escapes, restricting users to the set of programs that do not is often unworkable.

noexec Many systems that support shared libraries have the ability to override default library functions by pointing an environment variable (usually LD_PRELOAD) to an alternate shared library. On such systems, **sudo**'s *noexec* functionality can be used to prevent a program run by **sudo** from executing any other programs. Note, however, that this applies only to native dynamically-linked executables. Statically-linked executables and foreign executables running under binary emulation are not affected.

The *noexec* feature is known to work on SunOS, Solaris, *BSD, Linux, IRIX, Tru64 UNIX, MacOS X, HP-UX 11.x and AIX 5.3 and above. It should be supported on most operating systems that support the LD_PRELOAD environment variable. Check your operating system's manual pages for the dynamic linker (usually ld.so, ld.so.1, dyld, dld.sl, rld, or loader) to see if LD_PRELOAD is supported.

On Solaris 10 and higher, *noexec* uses Solaris privileges instead of the LD_PRELOAD environment variable.

To enable *noexec* for a command, use the NOEXEC tag as documented in the User Specification section above. Here is that example again:

aaron shanty = NOEXEC: /usr/bin/more, /usr/bin/vi

This allows user **aaron** to run */usr/bin/more* and */usr/bin/vi* with *noexec* enabled. This will prevent those two commands from executing other commands (such as a shell). If you are unsure whether or not your system is capable of supporting *noexec* you can always just try it out and check whether shell escapes work when *noexec* is enabled.

Note that restricting shell escapes is not a panacea. Programs running as root are still capable of many potentially hazardous operations (such as changing or overwriting files) that could lead to unintended privilege escalation. In the specific case of an editor, a safer approach is to give the user permission to run **sudoedit** (see below).

Secure editing

The **sudoers** plugin includes **sudoedit** support which allows users to securely edit files with the editor of their choice. As **sudoedit** is a built-in command, it must be specified in the *sudoers* file without a leading path. However, it may take command line arguments just as a normal command does. Wildcards used in *sudoedit* command line arguments are expected to be path names, so a forward slash ('/') will not be matched by a wildcard.

Unlike other **sudo** commands, the editor is run with the permissions of the invoking user and with the environment unmodified. More information may be found in the description of the **-e** option in sudo(8).

For example, to allow user operator to edit the ''message of the day'' file:

```
operator   sudoedit /etc/motd
```

The operator user then runs **sudoedit** as follows:

```
$ sudoedit /etc/motd
```

The editor will run as the operator user, not root, on a temporary copy of */etc/motd*. After the file has been edited, */etc/motd* will be updated with the contents of the temporary copy.

Users should *never* be granted **sudoedit** permission to edit a file that resides in a directory the user has write access to, either directly or via a wildcard. If the user has write access to the directory it is possible to replace the legitimate file with a link to another file, allowing the editing of arbitrary files. To prevent this, starting with version 1.8.16, symbolic links will not be followed in writable directories and **sudoedit** will refuse to edit a file located in a writable directory unless the *sudoedit_checkdir* option has been disabled or the invoking user is root. Additionally, in version 1.8.15 and higher, **sudoedit** will refuse to open a symbolic link unless either the *sudoedit_follow* option is enabled or the *sudoedit* command is prefixed with the FOLLOW tag in the *sudoers* file.

Time stamp file checks

sudoers will check the ownership of its time stamp directory (*/var/run/sudo/ts* by default) and ignore the directory's contents if it is not owned by root or if it is writable by a user other than root. Older versions of **sudo** stored time stamp files in */tmp*; this is no longer recommended as it may be possible for a user to create the time stamp themselves on systems that allow unprivileged users to change the ownership of files they create.

While the time stamp directory *should* be cleared at reboot time, not all systems contain a */var/run* directory. To avoid potential problems, **sudoers** will ignore time stamp files that date from before the machine booted on systems where the boot time is available.

Some systems with graphical desktop environments allow unprivileged users to change the system clock. Since **sudoers** relies on the system clock for time stamp validation, it may be possible on such systems for a user to run **sudo** for longer than *timestamp_timeout* by setting the clock back. To combat this, **sudoers** uses a monotonic clock (which never moves backwards) for its time stamps if the system supports it.

sudoers will not honor time stamps set far in the future. Time stamps with a date greater than current_time + 2 * TIMEOUT will be ignored and **sudoers** will log and complain.

Since time stamp files live in the file system, they can outlive a user's login session. As a result, a user may be able to login, run a command with **sudo** after authenticating, logout, login again, and run **sudo** without authenticating so long as the record's time stamp is within 5 minutes (or whatever value the timeout is set to in the *sudoers* file). When the *tty_tickets* option is enabled, the time stamp record includes the device number of the terminal the user authenticated with. This provides per-tty granularity but time stamp records still may outlive the user's session. The time stamp record also includes the session ID of the process that last authenticated. This prevents processes in different terminal sessions from using the same time stamp record. It also helps reduce the chance that a user will be able to run **sudo** without entering a password when logging out and back in again on the same terminal.

DEBUGGING

Versions 1.8.4 and higher of the **sudoers** plugin support a flexible debugging framework that can help track down what the plugin is doing internally if there is a problem. This can be configured in the sudo.conf(5) file.

The **sudoers** plugin uses the same debug flag format as the **sudo** front-end: *subsystem@priority*.

The priorities used by **sudoers**, 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 **sudoers** plugin:

alias User_Alias, Runas_Alias, Host_Alias and Cmnd_Alias processing

all matches every subsystem

audit BSM and Linux audit code

auth user authentication

defaults *sudoers* file *Defaults* settings

env environment handling

ldap LDAP-based sudoers

logging logging support

match matching of users, groups, hosts and netgroups in the *sudoers* file

netif network interface handling

nss network service switch handling in **sudoers**

parser *sudoers* file parsing

perms permission setting

plugin The equivalent of *main* for the plugin.

pty pseudo-tty related code

rbtree redblack tree internals

sssd SSSD-based sudoers

util utility functions

For example:

Debug sudo /var/log/sudo_debug match@info,nss@info

For more information, see the sudo.conf(5) manual.

SEE ALSO

ssh(1), su(1), fnmatch(3), glob(3), mktemp(3), strftime(3), sudo.conf(5), sudoers.ldap(5), sudo(8), sudo_plugin(8), visudo(8)

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 (https://www.sudo.ws/contributors.html) for an exhaustive list of people who have contributed to **sudo**.

CAVEATS

The *sudoers* file should **always** be edited by the **visudo** command which locks the file and does grammatical checking. It is imperative that the *sudoers* file be free of syntax errors since **sudo** will not run with a syntactically incorrect *sudoers* file.

When using netgroups of machines (as opposed to users), if you store fully qualified host name in the netgroup (as is usually the case), you either need to have the machine's host name be fully qualified as returned by the hostname command or use the *fqdn* option in *sudoers*.

BUGS

If you feel you have found a bug in **sudo**, please submit a bug report at https://bugzilla.sudo.ws/

SUPPORT

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

DISCLAIMER