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 *usr/local/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, see sudoers.ldap(5).

Configuring sudo.conf for sudoers

sudo consults the sudo.conf(5) file to determine which plugins to load. If no sudo.conf(5) file is present, or if it contains no *Plugin* lines, **sudoers** will be used for auditing, 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_audit sudoers.so
        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. Plugin arguments, if any, should be listed after the path to the plugin (i.e., after *sudoers.so*). The arguments are only effective for the plugin that opens (and parses) the *sudoers* file.

For **sudo** version 1.9.1 and higher, this is the *sudoers_audit* plugin. For older versions, it is the *sudoers_policy* plugin. Multiple arguments may be specified, separated by white space. For example:

        Plugin sudoers_audit sudoers.so sudoers_mode=0400 error_recovery=false

The following plugin arguments are supported:

error_recovery=bool
        The *error_recovery* argument can be used to control whether **sudoers** should attempt to recover from syntax errors in the *sudoers* file. If set to *true* (the default), **sudoers** will try to recover from a syntax error by discarding the portion of the line that contains the error until the end of the line. A value of *false* will disable error recovery. Prior to version 1.9.3, no error recovery was performed.

ignore_perms=bool
        The *ignore_perms* argument can be used to disable security checks when loading the *sudoers* file. If enabled, the *sudoers* file will be loaded regardless of the owner or file mode. This argument is intended to be used for testing purposes and should not be enabled on production systems.

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=user-ID
> 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=group-ID
> 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), 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*.

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**. By default, all attempts to run **sudo** (successful or not) are 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 user-ID that was used to authenticate, the terminal session ID, the start time of the session leader (or parent process) 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 *timestamp_timeout* option). By default, **sudoers** uses a separate record for each terminal, which means that a user's login sessions are authenticated separately. The *timestamp_type* option can be used to select the type of time stamp record **sudoers** will use.

Logging

By default, **sudoers** logs both successful and unsuccessful attempts (as well as errors). The *log_allowed* and *log_denied* flags can be used to control this behavior. Messages can be logged to syslog(3), a log file, or both. The default is to log to syslog(3) but this is configurable via the *syslog* and *logfile* settings. See *EVENT LOGGING* for a description of the log file format.

sudoers is also capable of running a command in a pseudo-terminal and logging input and/or output. The standard input, standard output, and standard error can be logged even when not associated with a terminal. For more information about I/O logging, see the *I/O LOGGING* section.

Starting with version 1.9, the *log_servers* setting may be used to send event and I/O log data to a remote server running **sudo_logsrvd** or another service that implements the protocol described by sudo_logsrv.proto(5).

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* flag 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* flag is enabled, the environment is initialized based on the *path* and *setenv* settings in */etc/login.conf*. The HOME, MAIL, SHELL, LOGNAME and USER environment variables are initialized based on the target user and the SUDO_* variables are set based on the invoking user. Additional variables, such as DISPLAY, PATH and TERM, are preserved from the invoking user's environment if permitted by the *env_check*, or *env_keep* options. A few environment variables are treated specially. If the PATH and TERM variables

are not preserved from the user's environment, they will be set to default values. The LOGNAME and USER are handled as a single entity. If one of them is preserved (or removed) from the user's environment, the other will be as well. If LOGNAME and USER are to be preserved but only one of them is present in the user's environment, the other will be set to the same value. This avoids an inconsistent environment where one of the variables describing the user name is set to the invoking user and one is set to the target user. 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 may be interpreted as functions by the **bash** shell. Prior to version 1.8.11, such variables were always removed.

If, however, the *env_reset* flag is disabled, any variables not explicitly denied by the *env_check* and *env_delete* options are allowed and their values are inherited from the invoking process. Prior to version 1.8.21, environment variables with a value beginning with '()' were always removed. Beginning with version 1.8.21, a pattern in *env_delete* is used to match **bash** shell functions instead. Since it is not possible to block all potentially dangerous environment variables, use of the default *env_reset* behavior is encouraged.

Environment variables specified by *env_check*, *env_delete*, or *env_keep* may include one or more '*' characters which will match zero or more characters. No other wildcard characters are supported.

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, a **bash** shell function could be matched as follows:

    env_keep += "BASH_FUNC_my_func%%=()*"

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

The complete list of environment variables that are preserved or removed, as modified by global Defaults parameters in *sudoers*, is displayed when **sudo** is run by **root** with the **-V** option. The list of environment variables to remove varies based on the operating system **sudo** is running on.

Other settings may influence the command environment:

⊕ **sudoers** options such as *always_set_home*, *secure_path*, *set_logname*, *set_home*, and *setenv*.

⊕ Command tags, such as SETENV and NOSETENV. Note that SETENV is implied if the command matched is **ALL**.

⊕ **sudo** options, such as **-E** and **-i**.

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.

The dynamic linker on most operating systems will remove variables that can control dynamic linking from the environment of set-user-ID 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 the **-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 unless permitted by *env_keep* or *env_check*, described above.

Finally, the *restricted_env_file* and *env_file* files are applied, if present. The variables in *restricted_env_file* are applied first and are subject to the same restrictions as the invoking user's environment, as detailed above. The variables in *env_file* are applied last and are not subject to these restrictions. In both cases, variables present in the files will only be set to their specified values if 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.

Resource limits
By default, **sudoers** uses the operating system's native method of setting resource limits for the target user. On Linux systems, resource limits are usually set by the *pam_limits.so* PAM module. On some

BSD systems, the */etc/login.conf* file specifies resource limits for the user. On AIX systems, resource limits are configured in the */etc/security/limits* file. If there is no system mechanism to set per-user resource limits, the command will run with the same limits as the invoking user. The one exception to this is the core dump file size, which is set by **sudoers** to 0 by default. Disabling core dumps by default makes it possible to avoid potential security problems where the core file is treated as trusted input.

Resource limits may also be set in the *sudoers* file itself, in which case they override those set by the system. See the *rlimit_as, rlimit_core, rlimit_cpu, rlimit_data, rlimit_fsize, rlimit_locks, rlimit_memlock, rlimit_nofile, rlimit_nproc, rlimit_rss, rlimit_stack* options described below. Resource limits in **sudoers** may be specified in one of the following formats:

"value" Both the soft and hard resource limits are set to the same value. The special value "infinity" can be used to indicate that the value is unlimited.

"soft,hard"
        Two comma-separated values. The soft limit is set to the first value and the hard limit is set to the second. Both values must either be enclosed in a set of double quotes, or the comma must be escaped with a backslash ('\'). The special value "infinity" may be used in place of either value.

"default"
        The default resource limit for the user will be used. This may be a user-specific value (see above) or the value of the resource limit when **sudo** was invoked for systems that don't support per-user limits.

"user"   The invoking user's resource limits will be preserved when running the command.

For example, to restore the historic core dump file size behavior, a line like the following may be used.

        Defaults rlimit_core=default

Resource limits in **sudoers** are only supported by version 1.8.7 or higher.

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*. For example:

        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*. Beginning with **sudo** 1.9.0, *Cmd_Alias* may be used in place of *Cmnd_Alias* if desired.

```
Alias ::= 'User_Alias'  User_Alias_Spec (':' User_Alias_Spec)* |
      'Runas_Alias' Runas_Alias_Spec (':' Runas_Alias_Spec)* |
      'Host_Alias'  Host_Alias_Spec (':' Host_Alias_Spec)* |
      'Cmnd_Alias'  Cmnd_Alias_Spec (':' Cmnd_Alias_Spec)* |
      'Cmd_Alias'   Cmnd_Alias_Spec (':' Cmnd_Alias_Spec)*
```

```
User_Alias ::= NAME
```

```
User_Alias_Spec ::= User_Alias '=' User_List
```

```
Runas_Alias ::= NAME
```

```
Runas_Alias_Spec ::= Runas_Alias '=' Runas_List
```

```
Host_Alias ::= NAME
```

```
Host_Alias_Spec ::= Host_Alias '=' Host_List
```

```
Cmnd_Alias ::= NAME
```

```
Cmnd_Alias_Spec ::= Cmnd_Alias '=' 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
(':'). For example:

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 |
       '!'* #user-ID |
       '!'* %group |
       '!'* %#group-ID |
       '!'* +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_Alias*es. 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*, *user-ID*, *group*, *group-ID*, *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.

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 |
          '!'* #user-ID |
          '!'* %group |
          '!'* %#group-ID |
          '!'* %:nonunix_group |
          '!'* %:#nonunix_gid |
          '!'* +netgroup |
          '!'* Runas_Alias |
          '!'* ALL
```

A *Runas_List* is similar to a *User_List* except that instead of *User_Alias*es it can contain *Runas_Alias*es. User names and groups are matched as strings. In other words, two users (groups) with the same user (group) ID are considered to be distinct. If you wish to match all user names with the same user-ID (e.g., **root** and **toor**), you can use a user-ID instead of a name (#0 in the example given). The user-ID or group-ID specified in a *Runas_Member* need not be listed in the password or group database.

```
Host_List ::= Host |
          Host ',' Host_List

Host ::= '!'* host name |
       '!'* ip_addr |
       '!'* network(/netmask)? |
```

'!'* +netgroup |
'!'* Host_Alias |
'!'* ALL

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 *hostname* command on your machine returns the fully qualified host name, you'll need to use the *fqdn* flag for wildcards to be useful. **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

Digest_List ::= Digest_Spec |
        Digest_Spec ',' Digest_List

Cmnd_List ::= Cmnd |
       Cmnd ',' Cmnd_List

command name ::= regex |
         file name

command ::= command name |
      command name args |
      command name regex |
      command name '""' |
      ALL

Edit_Spec ::= "sudoedit" file name+ |
          "sudoedit" regex |
          "sudoedit"

List_Spec ::= "list"

Cmnd ::= Digest_List? '!'* command |
      '!'* directory |
      '!'* Edit_Spec |
      '!'* List_Spec |
      '!'* Cmnd_Alias

A *Cmnd_List* is a list of one or more commands, directories, or aliases. A command is a fully qualified file name, which may include shell-style wildcards (see the *Wildcards* section below), or a regular expression that starts with '^' and ends with '$' (see the *Regular expressions* section below). A directory is a fully qualified path name ending in a '/'. When you specify a directory in a *Cmnd_List*, the user will be able to run any file within that directory (but not in any sub-directories therein). If no command line arguments are specified, the user may run the command with any arguments they choose. Command line arguments can include wildcards or be a regular expression that starts with '^' and ends with '$'. If the command line arguments consist of '""', the command may only be run with *no* arguments.

If a *Cmnd* has associated command line arguments, the arguments in the *Cmnd* must match those given by the user on the command line. If the arguments in a *Cmnd* begin with the '^' character, they will be interpreted as a regular expression and matched accordingly. Otherwise, shell-style wildcards are used when matching. Unless a regular expression is specified, the following characters must be escaped with a '\' if they are used in command arguments: ',', ':', '=', '\'. To prevent arguments in a *Cmnd* that begin with a '^' character from being interpreted as a regular expression, the '^' must be escaped with a '\'.

There are two commands built into **sudo** itself: "list" and "sudoedit". Unlike other commands, these two must be specified in the *sudoers* file *without* a leading path.

The "list" built-in can be used to permit a user to list another user's privileges with **sudo**'s **-U** option. For example, "sudo -l -U otheruser". A user with the "list" privilege is able to list another user's privileges even if they don't have permission to run commands as that user. By default, only root or a user with the ability to run any command as either root or the specified *user* on the current host may use the **-U** option. No command line arguments may be specified with the "list" built-in.

The "sudoedit" built-in 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. Unlike other commands, "sudoedit" is

built into **sudo** itself and must be specified in the *sudoers* file *without* a leading path. If a leading path is present, for example */usr/bin/sudoedit*, the path name will be silently converted to "sudoedit". A fully-qualified path for **sudoedit** is treated as an error by **visudo**.

A *command* may be preceded by a *Digest_List*, a comma-separated list of one or more *Digest_Spec* entries. If a *Digest_List* is present, the command will only match successfully if it can be verified using one of the SHA-2 digests in the list. Starting with version 1.9.0, the **ALL** reserved word can be used in conjunction with a *Digest_List*. 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. See the description of the *fdexec* setting for more information on how **sudo** executes commands that have an associated digest.

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 ('Defaults'), all users on a specific host ('Defaults@host'), a specific user ('Defaults:user'), a specific command ('Defaults!cmnd'), or commands being run as a specific user ('Defaults>runasuser').

White space is not permitted between 'Defaults' and the '@', ':', '!', or '>' characters. While a comma-separated list may be used in place of a single value after the '@', ':', '!', or '>' character, using an alias instead of a list is often improve readability. 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 ('\').

To include a literal backslash character in a command line argument you must escape the backslash twice. For example, to match '\n' as part of a command line argument, you must use '\\\\n' in the *sudoers* file. This is due to there being two levels of escaping, one in the *sudoers* parser itself and another when command line arguments are matched by the fnmatch(3) or regexec(3) function.

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: global, 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? Option_Spec* (Tag_Spec ’:’)* Cmnd

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

Option_Spec ::= (Date_Spec | Timeout_Spec | Chdir_Spec | Chroot_Spec)

Date_Spec ::= (’NOTBEFORE=timestamp’ | ’NOTAFTER=timestamp’)

Timeout_Spec ::= ’TIMEOUT=timeout’

Chdir_Spec ::= ’CWD=directory’

Chroot_Spec ::= ’CHROOT=directory’

Tag_Spec ::= (’EXEC’ | ’NOEXEC’ | ’FOLLOW’ | ’NOFOLLOW’ |
        ’LOG_INPUT’ | ’NOLOG_INPUT’ | ’LOG_OUTPUT’ |
        ’NOLOG_OUTPUT’ | ’MAIL’ | ’NOMAIL’ | ’INTERCEPT’ |
        ’NOINTERCEPT’ | ’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** (unless *runas_default* has been set to a different value) but this can also 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_List*s (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 the **-u** option. The second defines a list of groups that may be specified via the **-g** option (in addition to any of the target user's groups). If both *Runas_List*s are specified, the command may be run with any combination of users and groups listed in their respective *Runas_List*s. If only the first is specified, the command may be run as any user in the list and, optionally, with any group the target user belongs to. 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_List*s are empty, the command may

only be run as the invoking user and the group, if specified, must be one that the invoking user is a member of. If no *Runas_Spec* is specified, the command may only be run as the *runas_default* user (**root** by default) and the group, if specified, must be one that the *runas_default* user is a member of.

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* on the host boulder--but only as **operator**. For example:

$ 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

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

In this example only the group will be set, the command still runs as user **tcm**. For example:

$ 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.

Option_Spec

A *Cmnd* may have zero or more options associated with it. Options may consist of start and/or end dates and command timeouts. Once an option is set for a *Cmnd*, subsequent *Cmnd*s in the *Cmnd_Spec_List*, inherit that option unless it is overridden by another option. Option names are reserved words in *sudoers*. This means that none of the valid option names (see below) can be used when declaring an alias.

Date_Spec

sudoers rules can be specified with a start and end date via the NOTBEFORE and NOTAFTER settings. The time stamp must be specified in "Generalized Time" as defined by RFC 4517. The format is effectively 'yyyymmddHHMMSSZ' where the minutes and seconds are optional. The 'Z' suffix indicates that the time stamp is in Coordinated Universal Time (UTC). It is also possible to specify a timezone offset from UTC in hours and minutes instead of a 'Z'. For example, '-0500' would correspond to Eastern Standard time in the US. As an extension, if no 'Z' or timezone offset is specified, local time will be used.

The following are all valid time stamps:

    20170214083000Z
    2017021408Z
    20160315220000-0500
    20151201235900

Timeout_Spec

A command may have a timeout associated with it. If the timeout expires before the command has exited, the command will be terminated. The timeout may be specified in combinations of days, hours, minutes, and seconds with a single-letter case-insensitive suffix that indicates the unit of time. For example, a timeout of 7 days, 8 hours, 30 minutes, and 10 seconds would be written as '7d8h30m10s'. If a number is specified without a unit, seconds are assumed. Any of the days, minutes, hours, or seconds may be omitted. The order must be from largest to smallest unit and a unit may not be specified

more than once.

The following are all *valid* timeout values: '7d8h30m10s', '14d', '8h30m', '600s', '3600'. The following are *invalid* timeout values: '12m2w1d', '30s10m4h', '1d2d3h'.

This setting is only supported by version 1.8.20 or higher.

Chdir_Spec

The working directory that the command will be run in can be specified using the CWD setting. The *directory* must be a fully-qualified path name beginning with a '/' or '~' character, or the special value "*". A value of "*" indicates that the user may specify the working directory by running **sudo** with the **-D** option. By default, commands are run from the invoking user's current working directory, unless the **-i** option is given. Path names of the form *~user/path/name* are interpreted as being relative to the named user's home directory. If the user name is omitted, the path will be relative to the runas user's home directory.

This setting is only supported by version 1.9.3 or higher.

Chroot_Spec

The root directory that the command will be run in can be specified using the CHROOT setting. The *directory* must be a fully-qualified path name beginning with a '/' or '~' character, or the special value "*". A value of "*" indicates that the user may specify the root directory by running **sudo** with the **-R** option. This setting can be used to run the command in a chroot(2) "sandbox" similar to the chroot(8) utility. Path names of the form *~user/path/name* are interpreted as being relative to the named user's home directory. If the user name is omitted, the path will be relative to the runas user's home directory.

This setting is only supported by version 1.9.3 or higher.

Tag_Spec

A command may have zero or more tags associated with it. The following tag values are supported: EXEC, NOEXEC, FOLLOW, NOFOLLOW, LOG_INPUT, NOLOG_INPUT, LOG_OUTPUT, NOLOG_OUTPUT, MAIL, NOMAIL, INTERCEPT, NOINTERCEPT, PASSWD, NOPASSWD, SETENV, and NOSETENV. Once a tag is set on a *Cmnd*, subsequent *Cmnd*s 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* flag 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* flag on a per-command basis. For more information, see *I/O LOGGING*.

LOG_OUTPUT and NOLOG_OUTPUT

These tags override the value of the *log_output* flag on a per-command basis. For more information, see *I/O LOGGING*.

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* flag 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 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* setting.

By default, if the NOPASSWD tag is applied to any of a user's entries for the current host, the user will be able to run 'sudo -l' without a password. Additionally, a user may only run 'sudo -v' without a password if all of the user's entries for the current host have the NOPASSWD tag. This behavior may be overridden via the *verifypw* and *listpw* options.

 SETENV and NOSETENV

These tags override the value of the *setenv* flag on a per-command basis. If SETENV has been set for a command, the user may disable the *env_reset* flag 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.

 INTERCEPT and NOINTERCEPT

If **sudo** has been compiled with *intercept* support and the underlying operating system supports it, the INTERCEPT tag can be used to cause programs spawned by a command to be validated against *sudoers* and logged just like they would be if run through **sudo** directly. This is useful in conjunction with commands that allow shell escapes such as editors, shells, and paginators. There is additional overhead due to the policy check that may add latency when running commands such as shell scripts that execute a large number of sub-commands. For interactive commands, such as a shell or editor, the overhead is not usually noticeable.

In the following example, user **chuck** may run any command on the machine "research" in intercept mode.

chuck   research = INTERCEPT: ALL

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

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 ']'.

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.

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.
Wildcards can match any character, including white space. In most cases, it is safer to use a regular expression to match command line arguments. For more information, see *Wildcards in command arguments* below.

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.

Regular expressions
Starting with version 1.9.10, it is possible to use regular expressions for path names and command line arguments. Regular expressions are more expressive than shell-style *wildcards* and are usually safer because they provide a greater degree of control when matching. The type of regular expressions supported by **sudoers** are POSIX extended regular expressions, similar to those used by the egrep(1) utility. They are usually documented in the regex(7) or re_format(7) manual, depending on the system. As an extension, if the regular expression begins with "(?i)", it will be matched in a case-insensitive manner.

In *sudoers*, regular expressions must start with a '^' character and end with a '$'. This makes it explicit what is, or is not, a regular expression. Either the path name, the command line arguments or both may be regular expressions. Because the path name and arguments are matched separately, it is even possible to use wildcards for the path name and regular expressions for the arguments. It is not possible to use a single regular expression to match both the command and its arguments. Regular expressions in *sudoers* are limited to 1024 characters.

There is no need to escape *sudoers* special characters in a regular expression other than the pound sign ('#').

In the following example, user **john** can run the passwd(1) command as **root** on any host but is not allowed to change **root**'s password. This kind of rule is impossible to express safely using wildcards.

        john   ALL = /usr/bin/passwd ^[a-zA-Z0-9_]+$,\
                    !/usr/bin/passwd root

It is also possible to use a regular expression in conjunction with **sudoedit** rules. The following rule

would give user bob the ability to edit the */etc/motd*, */etc/issue*, and */etc/hosts* files only.

    bob   ALL = sudoedit ^/etc/(motd|issue|hosts)$

Regular expressions may also be used to match the command itself. In this example, a regular expression is used to allow user **sid** to run the */usr/sbin/groupadd*, */usr/sbin/groupmod*, */usr/sbin/groupdel*, */usr/sbin/useradd*, */usr/sbin/usermod*, and */usr/sbin/userdel* commands as **root**.

    sid   ALL = ^/usr/sbin/(group|user)(add|mod|del)$

One disadvantage of using a regular expression to match the command name is that it is not possible to match relative paths such as *./useradd* or *../sbin/useradd*. This has security implications when a regular expression is used for the command name in conjunction with the negation operator, '!', as such rules can be trivially bypassed. Because of this, using a negated regular expression for the command name is **strongly discouraged**. This does not apply to negated commands that only use a regular expression to match the command arguments. See *Regular expressions in command names* below for more information.

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. For compatibility with sudo versions prior to 1.9.1, #*include* and #*includedir* are also accepted.

An include file 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* one 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.

Starting with version 1.9.1, the path to the include file may contain white space if it is escaped with a backslash ('\'). Alternately, the entire path may be enclosed in double quotes (""), in which case no escaping is necessary. To include a literal backslash in the path, '\\' should be used.

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*. Any path name separator characters ('/') present in the host name will be replaced with an underbar ('_') during expansion.

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 suspend processing of the current file and 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. After parsing the files in the directory, control returns to the file that contained the @*includedir* directive.

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 user-ID). 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*. Attempting

to define an *alias* named **ALL** will result in a syntax error. Using **ALL** can be dangerous since in a command context, it allows the user to run *any* command on the system.

The following option names permitted in an *Option_Spec* are also considered reserved words: CHROOT, TIMEOUT, CWD, NOTBEFORE and NOTAFTER. Attempting to define an *alias* with the same name as one of the options will result in a syntax error.

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 the *runas_default* user unless the **-u** option is used). This
                       flag is largely obsolete and has no effect unless the *env_reset* flag has been
                       disabled or HOME is present in the *env_keep* list, both of which are strongly
                       discouraged. 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.

case_insensitive_group
                       If enabled, group names in *sudoers* will be matched in a case insensitive manner.
                       This may be necessary when users are stored in LDAP or AD. This flag is *on* by
                       default.

case_insensitive_user
                       If enabled, user names in *sudoers* will be matched in a case insensitive manner.
                       This may be necessary when groups are stored in LDAP or AD. This flag is *on* by
                       default.

closefrom_override If set, the user may use the **-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 pseudo-terminal (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 pseudo-terminal.

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, macOS fails to restart the tcgetattr(3) and tcsetattr(3) functions (this is a bug in macOS). 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 SUDO_EDITOR, VISUAL or EDITOR environment variables before falling back on the default editor list. **visudo** is typically run as **root** so this flag may allow a user with **visudo** privileges to run arbitrary commands as **root** without logging. An alternative is to place a colon-separated list of "safe" editors int the *editor* setting. **visudo** will then only use SUDO_EDITOR, VISUAL or EDITOR if they match a value specified in *editor*. If the *env_reset* flag is enabled, the SUDO_EDITOR, VISUAL and/or EDITOR 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 *on* by default.

env_reset   If set, **sudo** will run the command in a minimal environment containing the TERM, PATH, HOME, MAIL, SHELL, LOGNAME, USER and SUDO_* variables. Any variables in the caller's environment or in the file specified by the *restricted_env_file* setting 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* setting (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* setting is enabled, 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* flag

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 paths 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 flag 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.

log_passwords   Most programs that require a user's password will disable echo before reading the password to avoid displaying the plaintext password on the screen. However, if terminal input is being logged (see *I/O LOGGING*), the password will still be present in the I/O log. If the *log_passwords* option is disabled, **sudoers** will attempt to prevent passwords from being logged. It does this by using the regular expressions in *passprompt_regex* to match a password prompt in the terminal output buffer. When a match is found, input characters in the I/O log will be replaced with '*' until either a line feed or carriage return is found in the terminal input or a new terminal output buffer is received. If, however, a program displays characters as the user types (such as **sudo** when *pwfeedback* is set), only the first character of the password will be replaced in the I/O log. This option has no effect unless *log_input* or *log_ttyin* are also set. This flag is *on* by default.

                       This setting is only supported by version 1.9.10 or higher.

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 flag is only effective when the "canonical" host name, as returned by the getaddrinfo(3) or gethostbyname(3) 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* flag 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). 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, macOS, and Solaris. This flag is *on* by default.

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

ignore_iolog_errors   Allow commands to be run even if **sudoers** cannot write to the I/O log (local or remote). 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* setting. This flag is *on* by default.

ignore_local_sudoers

If set via LDAP, parsing of */usr/local/etc/sudoers* will be skipped. This is intended for sites 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 */usr/local/etc/sudoers*. When this flag is enabled, */usr/local/etc/sudoers* does not

even need to exist. Since this flag 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.

ignore_unknown_defaults

If set, **sudo** will not produce a warning if it encounters an unknown Defaults entry in the *sudoers* file or an unknown sudoOption in LDAP. 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_allowed   If set, **sudoers** will log commands allowed by the policy to the system audit log (where supported) as well as to syslog and/or a log file. This flag is *on* by default.

                   This setting is only supported by version 1.8.29 or higher.

log_denied   If set, **sudoers** will log commands denied by the policy to the system audit log (where supported) as well as to syslog and/or a log file. This flag is *on* by default.

                   This setting is only supported by version 1.8.29 or higher.

log_exit_status   If set, **sudoers** will log the exit value of commands that are run to syslog and/or a log file. If a command was terminated by a signal, the signal name is logged as well. This flag is *off* by default.

                   This setting is only supported by version 1.9.8 or higher.

log_host   If set, the host name will be included in log entries written to the file configured by the *logfile* setting. This flag is *off* by default.

log_input   If set, **sudo** will run the command in a pseudo-terminal (if **sudo** was run from a terminal) and log all user input. If the standard input is not connected to the user's terminal, due to I/O redirection or because the command is part of a pipeline, that input is also logged. For more information about I/O logging, see the *I/O LOGGING* section. This flag is *off* by default.

log_output   If set, **sudo** will run the command in a pseudo-terminal (if **sudo** was run from a terminal) and log all output that is sent to the user's terminal, the standard output or the standard error. If the standard output or standard error is not connected to the user's terminal, due to I/O redirection or because the command is part of a pipeline,

that output is also logged. For more information about I/O logging, see the *I/O LOGGING* section. This flag is *off* by default.

log_server_keepalive

If set, **sudo** will enable the TCP keepalive socket option on the connection to the log server. This enables the periodic transmission of keepalive messages to the server. If the server does not respond to a message, the connection will be closed and the running command will be terminated unless the *ignore_iolog_errors* flag (I/O logging enabled) or the *ignore_log_errors* flag (I/O logging disabled) is set. This flag is *on* by default.

This setting is only supported by version 1.9.0 or higher.

log_server_verify   If set, the server certificate received during the TLS handshake must be valid and it must contain either the server name (from *log_servers*) or its IP address. If either of these conditions is not met, the TLS handshake will fail. This flag is *on* by default.

This setting is only supported by version 1.9.0 or higher.

log_stderr   If set, **sudo** will log the standard error if it is not connected to the user's terminal. This can be used to log output to a pipe or redirected to a file. This flag is *off* by default but is enabled when either the *log_output* flag or the LOG_OUTPUT command tag is set.

log_stdin   If set, **sudo** will log the standard input if it is not connected to the user's terminal. This can be used to log input from a pipe or redirected from a file. This flag is *off* by default but is enabled when either the *log_input* flag or the LOG_INPUT command tag is set.

log_stdout   If set, **sudo** will log the standard output if it is not connected to the user's terminal. This can be used to log output to a pipe or redirected to a file. This flag is *off* by default but is enabled when either the *log_output* flag or the LOG_OUTPUT command tag is set.

log_subcmds   If set, **sudoers** will log when a command spawns a child process and executes a program using the execve(2), execl(3), execle(3), execlp(3), execv(3), execvp(3), execvpe(3), or system(3) library functions. For example, if a shell is run by **sudo**, the individual commands run via the shell will be logged. This flag is *off* by default.

The *log_subcmds* flag uses the same underlying mechanism as the *intercept* setting. Some commands may not work properly when *log_subcmds* is enabled, due to the way it intercepts sub-commands. See *Preventing shell escapes* for more information on what systems support this option and its limitations. This setting is only supported by version 1.9.8 or higher and is incompatible with SELinux RBAC support unless the system supports seccomp(2) filter mode.

log_ttyin   If set, **sudo** will run the command in a pseudo-terminal and log user keystrokes sent to the user's terminal, if one is present. This flag is *off* by default but is enabled when either the *log_input* flag or the LOG_INPUT command tag is set. If no terminal is present, for example when running a remote command using ssh(1), this flag will have no effect.

log_ttyout   If set, **sudo** will run the command in a pseudo-terminal and log all output displayed on the user's terminal, if one is present. This flag is *off* by default but is enabled when either the *log_output* flag or the LOG_OUTPUT command tag is set. If no terminal is present, for example when running a remote command using ssh(1), this flag will have no effect.

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 *on* 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, **sudoers** will look up each group the user is a member of by group-ID to
                 determine the group name (this is only done once). The resulting list of the user's
                 group names is used when matching groups 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, or 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. In this case, **sudoers** must
                 look up any group name listed in the *sudoers* file and use the group-ID instead of
                 the group name when determining whether the user is a member of the group.

                 If *match_group_by_gid* is enabled, group database lookups performed by **sudoers**
                 will be keyed by group name as opposed to group-ID. On systems where there are
                 multiple sources for the group database, it is possible to have conflicting group
                 names or group-IDs in the local */etc/group* file and the remote group database. On
                 such systems, enabling or disabling *match_group_by_gid* can be used to choose
                 whether group database queries are performed by name (enabled) or ID (disabled),
                 which may aid in working around group entry conflicts.

                 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.

intercept   If set, all commands run via **sudo** will behave as if the INTERCEPT tag has been
                 set, unless overridden by an NOINTERCEPT tag. Some commands may not work

properly when *intercept* is enabled, due to the way it intercept sub-commands. See the description of INTERCEPT and NOINTERCEPT above as well as the *Preventing shell escapes* section at the end of this manual. This flag is *off* by default.

This setting is only supported by version 1.9.8 or higher and is incompatible with SELinux RBAC support unless the system supports seccomp(2) filter mode.

intercept_allow_setid

On most systems, the dynamic loader will ignore LD_PRELOAD (or the equivalent) when running set-user-ID and set-group-ID programs, effectively disabling intercept mode. To prevent this from happening, **sudoers** will not permit a set-user-ID or set-group-ID program to be run in intercept mode unless *intercept_allow_setid* is enable. This flag has no effect unless the *intercept* flag is enabled or the INTERCEPT tag has been set for the command. This flag is *on* by default when the *intercept_type* option is set to *trace*, otherwise it default to *off*.

This setting is only supported by version 1.9.8 or higher.

intercept_authenticate

If set, commands run by an intercepted process must be authenticated when the user's time stamp is not current. For example, if a shell is run with *intercept* enabled, as soon as the invoking user's time stamp is out of date, subsequent commands will need to be authenticated. This flag has no effect unless the *intercept* flag is enabled or the INTERCEPT tag has been set for the command. This flag is *off* by default.

This setting is only supported by version 1.9.8 or higher.

intercept_verify   If set, **sudo** will attempt to verify that a command run in intercept mode has the expected path name, command line arguments and environment.

The process will be stopped after execve(2) has completed but before the new command has had a chance to run. To verify the command, **sudo** will read the command's path from */proc/PID/exe*, the command line arguments and environment from the process's memory, and compare them against the arguments that were passed to execve(2). In the event of a mismatch, the command will be sent a SIGKILL signal and terminated.

This can help prevent a time of check versus time of use issue with intercept mode

where the execve(2) arguments could be altered after the **sudoers** policy check. The checks can only be performed if the proc(5) file system is available. This flag has no effect unless the *intercept* flag is enabled or the INTERCEPT tag has been set for the command and the *intercept_type* option is set to *trace*.

This setting is incompatible with programs that change their root directory via chroot(2). If a program changes its root directory, path names will no longer match those seen by the **sudo** parent process and sub-commands will be terminated before they have a chance to run. This flag is *on* by default.

This setting is only supported by version 1.9.12 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.

noninteractive_auth   If set, authentication will be attempted even in non-interactive mode (when **sudo**'s **-n** option is specified). This allows authentication methods that don't require user interaction to succeed. Authentication methods that require input from the user's terminal will still fail. If disabled, authentication will not be attempted in non-interactive mode. This flag is *off* by default.

This setting is only supported by version 1.9.10 or higher.

pam_acct_mgmt   On systems that use PAM for authentication, **sudo** will perform PAM account validation for the invoking user by default. The actual checks performed depend on which PAM modules are configured. If enabled, account validation will be performed regardless of whether or not a password is required. This flag is *on* by default.

This setting is only supported by version 1.8.28 or higher.

pam_rhost   On systems that use PAM for authentication, **sudo** will set the PAM remote host value to the name of the local host when the *pam_rhost* flag is enabled. On Linux

systems, enabling *pam_rhost* may result in DNS lookups of the local host name
when PAM is initialized. On Solaris versions prior to Solaris 8, *pam_rhost* must be
enabled if *pam_ruser* is also enabled to avoid a crash in the Solaris PAM
implementation.

This flag is *off* by default on systems other than Solaris.

This setting is only supported by version 1.9.0 or higher.

pam_ruser   On systems that use PAM for authentication, **sudo** will set the PAM remote user
                        value to the name of the user that invoked sudo when the *pam_ruser* flag is
                        enabled. This flag is *on* by default.

                        This setting is only supported by version 1.9.0 or higher.

pam_session   On systems that use PAM for authentication, **sudo** will create a new PAM session
                        for the command to be run in. Unless **sudo** is given the **-i** or **-s** options, PAM
                        session modules are run with the "silent" flag enabled. This prevents last login
                        information from being displayed for every command on some systems. 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, *log_servers* has
                        not been set 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, *log_servers* has not been set
                        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
                        If set, the prompt specified by *passprompt* or the SUDO_PROMPT environment

variable will always be used and will replace the prompt provided by a PAM module or other authentication method. 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. 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.

runas_allow_unknown_id
If enabled, allow matching of runas user and group IDs that are not present in the password or group databases. In addition to explicitly matching unknown user or group IDs in a *Runas_List*, this option also allows the **ALL** alias to match unknown

IDs. This flag is *off* by default.

This setting is only supported by version 1.8.30 or higher. Older versions of **sudo** always allowed matching of unknown user and group IDs.

runas_check_shell   If enabled, **sudo** will only run commands as a user whose shell appears in the */etc/shells* file, even if the invoking user's *Runas_List* would otherwise permit it. If no */etc/shells* file is present, a system-dependent list of built-in default shells is used. On many operating systems, system users such as "bin", do not have a valid shell and this flag can be used to prevent commands from being run as those users. This flag is *off* by default.

This setting is only supported by version 1.8.30 or higher.

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 the *runas_default* user unless the **-u** option is used). This flag is largely obsolete and has no effect unless the *env_reset* flag has been disabled or HOME is present in the *env_keep* list, both of which are strongly discouraged. This flag is *off* by default.

set_logname   Normally, **sudo** will set the LOGNAME and USER environment variables to the name of the target user (the user specified by *runas_default* 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. The *set_logname* option will have no effect if the *env_reset* option has not been disabled and the *env_keep* list contains LOGNAME or USER. This flag is *on* by default.

set_utmp   When enabled, **sudo** will create an entry in the utmp (or utmpx) file when a pseudo-terminal is allocated. A pseudo-terminal is allocated by **sudo** when it is running in a terminal and one or more of the *log_input*, *log_output*, *log_stdin*, *log_stdout*, *log_stderr*, *log_ttyin*, *log_ttyout*, or *use_pty* flags is 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 user-IDs are set to
                  the target user (**root** by default). This option changes that behavior such that the
                  real user-ID is left as the invoking user's user-ID. In other words, this makes **sudo**
                  act as a set-user-ID wrapper. This can be useful on systems that disable some
                  potentially dangerous functionality when a program is run set-user-ID. 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.

syslog_pid   When logging via syslog(3), include the process ID in the log entry. This flag is
                  off by default.

This setting is only supported by version 1.8.21 or higher.

targetpw
If set, **sudo** will prompt for the password of the user specified by the **-u** option (defaults to the value of *runas_default*) instead of the password of the invoking user when running a command or editing a file. This flag precludes the use of a user-ID 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 terminal. If disabled, a single record is used for all login sessions.

This option has been superseded by the *timestamp_type* option.

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 sub-string 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, and **sudo** is running in a terminal, the command will be run in a new pseudo-terminal. If the **sudo** process is not attached to a terminal, *use_pty* has no effect.

A malicious program run under **sudo** may be capable of injecting commands into the user's terminal or running a background process that retains access to the user's terminal device even after the main program has finished executing. By running the command in a separate pseudo-terminal, this attack is no longer possible. This flag is *on* by default for **sudo** 1.9.14 and above.

user_command_timeouts

If set, the user may specify a timeout on the command line. If the timeout expires before the command has exited, the command will be terminated. If a timeout is specified both in the *sudoers* file and on the command line, the smaller of the two timeouts will be used. See the *Timeout_Spec* section for a description of the timeout syntax. This flag is *off* by default.

This setting is only supported by version 1.8.20 or higher.

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 (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.

command_timeout   The maximum amount of time a command is allowed to run before it is terminated. See the *Timeout_Spec* section for a description of the timeout syntax.

This setting is only supported by version 1.8.20 or higher.

log_server_timeout The maximum amount of time to wait when connecting to a log server or waiting for a server response. See the *Timeout_Spec* section for a description of the timeout syntax. The default value is 30 seconds.

This setting is only supported by version 1.9.0 or higher.

maxseq   The maximum sequence number that will be substituted for the '%{seq}' escape in the I/O log file (see the *iolog_dir* description below 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.

syslog_maxlen   On many systems, syslog(3) has a relatively small log buffer. IETF RFC 5424
                      states that syslog servers must support messages of at least 480 bytes and should
                      support messages up to 2048 bytes. By default, **sudoers** creates log messages up to
                      980 bytes which corresponds to the historic BSD syslog implementation which
                      used a 1024 byte buffer to store the message, date, hostname, and program name.
                      To prevent syslog messages from being truncated, **sudoers** will split up log
                      messages that are larger than *syslog_maxlen* bytes. When a message is split,
                      additional parts will include the string "(command continued)" after the user name
                      and before the continued command line arguments.

                      This setting is only supported by version 1.8.19 or higher.

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 password 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   File mode creation mask to use when running the command. Negate this option or
                     set it to 0777 to prevent **sudoers** from changing the umask. Unless the
                     umask_override flag is set, the actual umask will be the union of the user's umask
                     and the value of the *umask* setting, which defaults to 0022. This guarantees that
                     sudo never lowers the umask when running a command.

                     If *umask* is explicitly set in *sudoers*, it will override any umask setting in PAM or
                     login.conf. If *umask* is not set in *sudoers*, the umask specified by PAM or
                     login.conf will take precedence. The umask setting in PAM is not used for
                     sudoedit, which does not create a new PAM session.

Strings:

authfail_message   Message that is displayed after a user fails to authenticate. The message may
                     include the '%d' escape which will expand to the number of failed password
                     attempts. If set, it overrides the default message, "%d incorrect password
                     attempt(s)".

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 editor path names used by **sudoedit** and **visudo**. For
                     sudoedit, this list is used to find an editor when none of the SUDO_EDITOR,
                     VISUAL or EDITOR environment variables are set to an editor that exists and is
                     executable. For **visudo**, it is used as a white list of allowed editors; **visudo** will
                     choose the editor that matches the user's SUDO_EDITOR, VISUAL or EDITOR
                     environment variable if possible, or the first editor in the list that exists and is
                     executable if not. Unless invoked as **sudoedit**, **sudo** does not preserve the
                     SUDO_EDITOR, VISUAL or EDITOR environment variables unless they are
                     present in the *env_keep* list or the *env_reset* option is disabled. The default is
                     /usr/bin/vi.

intercept_type   The underlying mechanism used by the *intercept* and *log_subcmds* options. It has
                     the following possible values:

                     dso   Preload a dynamic shared object (shared library) that intercepts the
                             execve(2), execl(3), execle(3), execlp(3), execv(3), execvp(3), execvpe(3),
                             and system(3) library functions. A value of *dso* is incompatible with
                             sudo's SELinux RBAC support.

trace   Use ptrace(2) to intercept the execve(2) system call. This is only
            supported on Linux systems where seccomp(2) filtering is enabled. If the
            /proc/sys/kernel/seccomp/actions_avail file is missing or does not contain
            a "trap" element, setting *intercept_type* to *trace* will have no effect and *dso*
            will be used instead.

The default is to use *trace* if it is supported by the system and *dso* if it is not.

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.

Any path name separator characters ('/') present in the user, group or host name will be replaced with an underbar ('_') during expansion.

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. *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 *X*s will have the *X*s 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 *X*s.

iolog_flush   If set, **sudo** will flush I/O log data to disk after each write instead of buffering it. This makes it possible to view the logs in real-time as the program is executing but may significantly reduce the effectiveness of I/O log compression. This flag is *off* by default.

This setting is only supported by version 1.8.20 or higher.

iolog_group   The group name to look up when setting the group-ID on new I/O log files and directories. If *iolog_group* is not set, the primary group-ID of the user specified by *iolog_user* is used. If neither *iolog_group* nor *iolog_user* are set, I/O log files and directories are created with group-ID 0.

This setting is only supported by version 1.8.19 or higher.

iolog_mode   The file mode to use when creating I/O log files. Mode bits for read and write permissions for owner, group, or other are honored, everything else is ignored. The

file permissions will always include the owner read and write bits, even if they are not present in the specified mode. When creating I/O log directories, search (execute) bits are added to match the read and write bits specified by *iolog_mode*. Defaults to 0600 (read and write by user only).

This setting is only supported by version 1.8.19 or higher.

iolog_user   The user name to look up when setting the user and group-IDs on new I/O log files and directories. If *iolog_group* is set, it will be used instead of the user's primary group-ID. By default, I/O log files and directories are created with user and group-ID 0.

This setting can be useful when the I/O logs are stored on a Network File System (NFS) share. Having a dedicated user own the I/O log files means that **sudoers** does not write to the log files as user-ID 0, which is usually not permitted by NFS.

This setting is only supported by version 1.8.19 or higher.

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/db/sudo/lectured*.

log_server_cabundle

The path to a certificate authority bundle file, in PEM format, to use instead of the system's default certificate authority database when authenticating the log server. The default is to use the system's default certificate authority database. This setting has no effect unless *log_servers* is set and the remote log server is secured with TLS.

This setting is only supported by version 1.9.0 or higher.

log_server_peer_cert

The path to the **sudo** client's certificate file, in PEM format. This setting is required when the remote log server is secured with TLS and client certificate validation is enabled. For **sudo_logsrvd**, client certificate validation is controlled by the *tls_checkpeer* option, which defaults to *false*.

This setting is only supported by version 1.9.0 or higher.

log_server_peer_key

> The path to the **sudo** client's private key file, in PEM format. This setting is required when the remote log server is secured with TLS and client certificate validation is enabled. For **sudo_logsrvd**, client certificate validation is controlled by the *tls_checkpeer* flag, which defaults to *false*.
>
> This setting is only supported by version 1.9.0 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_askpass_service

> On systems that use PAM for authentication, this is the service name used when the **-A** option is specified. The default value is either 'sudo' or 'sudo', depending on whether or not the **-i** option is also specified. See the description of *pam_service* for more information.
>
> This setting is only supported by version 1.9.9 or higher.

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

On systems that use PAM for authentication, *passprompt* will only be used if the prompt provided by the PAM module matches the string "Password: " or "username's Password: ". This ensures that the *passprompt* setting does not interfere with challenge-response style authentication. The *passprompt_override* flag can be used to change this behavior.

The default value is 'Password: '.

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

sudoers_locale   Locale to use when parsing the sudoers file, logging commands, and sending email. Changing the locale may affect how sudoers is interpreted. Defaults to 'C'.

timestamp_type   **sudoers** uses per-user time stamp files for credential caching. The *timestamp_type* option can be used to specify the type of time stamp record used. It has the following possible values:

global   A single time stamp record is used for all of a user's login sessions, regardless of the terminal or parent process ID. An additional record is used to serialize password prompts when **sudo** is used multiple times in a pipeline, but this does not affect authentication.

ppid   A single time stamp record is used for all processes with the same parent process ID (usually the shell). Commands run from the same shell (or other common parent process) will not require a password for *timestamp_timeout* minutes (5 by default). Commands run via **sudo** with a different parent process ID, for example from a shell script, will be

authenticated separately.

tty   One time stamp record is used for each terminal, which means that a
           user's login sessions are authenticated separately. If no terminal is
           present, the behavior is the same as *ppid*. Commands run from the same
           terminal will not require a password for *timestamp_timeout* minutes (5 by
           default).

kernel   The time stamp is stored in the kernel as an attribute of the terminal
           device. If no terminal is present, the behavior is the same as *ppid*.
           Negative *timestamp_timeout* values are not supported and positive values
           are limited to a maximum of 60 minutes. This is currently only supported
           on OpenBSD.

The default value is *tty*.

This setting is only supported by version 1.8.21 or higher.

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**.

Strings that can be used in a boolean context:

admin_flag   The *admin_flag* option specifies the path to a file that is created the first time a user that
                is a member of the *sudo* or *admin* groups runs **sudo**. Only available if **sudo** is
                configured with the --enable-admin-flag option. The default value is
                ~/.sudo_as_admin_successful.

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 enclosed in single or double quotes. Variables in this file are only added if
                the variable does not already exist in the environment. This file is considered to be part
                of the security policy, its contents are not subject to other **sudo** environment restrictions
                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.

fdexec   Determines whether **sudo** will execute a command by its path or by an open file
                descriptor. It has the following possible values:

                always   Always execute by file descriptor.

                never   Never execute by file descriptor.

                digest_only
                        Only execute by file descriptor if the command has an associated digest in the
                        sudoers file.

                The default value is *digest_only*. This avoids a time of check versus time of use race
                condition when the command is located in a directory writable by the invoking user.

                fdexec will change the first element of the argument vector for scripts ($0 in the shell)
                due to the way the kernel runs script interpreters. Instead of being a normal path, it will
                refer to a file descriptor. For example, */dev/fd/4* on Solaris and */proc/self/fd/4* on Linux.
                A workaround is to use the SUDO_COMMAND environment variable instead.

                The *fdexec* setting is only used when the command is matched by path name. It has no
                effect if the command is matched by the built-in **ALL** alias.

                This setting is only supported by version 1.8.20 or higher. If the operating system does
                not support the fexecve(2) system call, this setting has no effect.

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 ("").

                On 64-bit systems, if the plugin is present but cannot be loaded, **sudoers** will look for a
                64-bit version and, if it exists, load that as a fallback. The exact rules for this vary by
                system. On Solaris, if the plugin is stored in a directory ending in "lib", **sudoers** will
                create a fallback path by appending "/64" to the directory name;
                /usr/local/lib/group_plugin.so becomes */usr/local/lib/64/group_plugin.so*. On Linux, a
                directory ending in "lib" will be transformed to "lib64" as the fallback path;
                /usr/local/lib/group_plugin.so becomes */usr/local/lib64/group_plugin.so*. On all other

systems, the fallback path is generated by adding a "64" before the file extension; *group_plugin.so* becomes *group_plugin64.so*.

On AIX systems, the plugin may be either a shared object ending in '.so' or an archive file containing a shared object ending in '.a' with the name of the shared object in parentheses at the end.

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*.

log_format   The event log format. Supported log formats are:

                 json   Logs in JSON format. JSON log entries contain the full user details as well as the
                        execution environment if the command was allowed. Due to limitations of the
                        protocol, JSON events sent via *syslog* may be truncated.

                 sudo   Traditional sudo-style logs, see *EVENT LOGGING* for a description of the log
                        file format.

                 This setting affects logs sent via syslog(3) as well as the file specified by the *logfile*
                 setting, if any. The default value is *sudo*.

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 (negate to prevent **sudo** from sending
                 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 (negate to prevent **sudo** from sending mail).
                 The address should be enclosed in double quotes ("") to protect against **sudo** interpreting
                 the '@' sign. Defaults to root.

rlimit_as   The maximum size to which the process's address space may grow (in bytes), if
                 supported by the operating system. See *Resource limits* for more information.

rlimit_core   The largest size core dump file that may be created (in bytes). See *Resource limits* for
                 more information. Defaults to 0 (no core dump created).

rlimit_cpu   The maximum amount of CPU time that the process may use (in seconds). See
                 Resource limits for more information.

rlimit_data   The maximum size of the data segment for the process (in bytes). See *Resource limits*

for more information.

rlimit_fsize   The largest size file that the process may create (in bytes). See *Resource limits* for more
                   information.

rlimit_locks   The maximum number of locks that the process may establish, if supported by the
                   operating system. See *Resource limits* for more information.

rlimit_memlock
                   The maximum size that the process may lock in memory (in bytes), if supported by the
                   operating system. See *Resource limits* for more information.

rlimit_nofile   The maximum number of files that the process may have open. See *Resource limits* for
                   more information.

rlimit_nproc   The maximum number of processes that the user may run simultaneously. See
                   Resource limits for more information.

rlimit_rss   The maximum size to which the process's resident set size may grow (in bytes). See
                   Resource limits for more information.

rlimit_stack   The maximum size to which the process's stack may grow (in bytes). See *Resource
                   limits* for more information.

restricted_env_file
                   The *restricted_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 enclosed in single or double quotes. Variables in this file are
                   only added if the variable does not already exist in the environment. Unlike *env_file*,
                   the file's contents are not trusted and are processed in a manner similar to that of the
                   invoking user's environment. If *env_reset* is enabled, variables in the file will only be
                   added if they are matched by either the *env_check* or *env_keep* list. If *env_reset* is
                   disabled, variables in the file are added as long as they are not matched by the
                   env_delete list. In either case, the contents of *restricted_env_file* are processed before
                   the contents of *env_file*.

runchroot   If set, **sudo** will use this value for the root directory when running a command. The
                   special value "*" will allow the user to specify the root directory via **sudo**'s **-R** option.
                   See the *Chroot_Spec* section for more details.

It is only possible to use *runchroot* as a command-specific Defaults setting if the command exists with the same path both inside and outside the chroot jail. This restriction does not apply to global, host, or user-based Defaults settings or to a *Cmnd_Spec* that includes a *Chroot_Spec*.

This setting is only supported by version 1.9.3 or higher.

runcwd   If set, **sudo** will use this value for the working directory when running a command. The special value "*" will allow the user to specify the working directory via **sudo**'s **-D** option. See the *Chdir_Spec* section for more details.

This setting is only supported by version 1.9.3 or higher.

secure_path   If set, **sudo** will use this value in place of the user's PATH environment variable. This option can be used to reset the PATH to a known good value that contains directories for system administrator commands such as */usr/sbin*.

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 authpriv.

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

syslog_badpri   Syslog priority to use when the user is not allowed to run a command or when authentication is unsuccessful. Defaults to alert.

The following syslog priorities are supported: **alert**, **crit**, **debug**, **emerg**, **err**, **info**, **notice**, **warning**, and **none**. Negating the option or setting it to a value of **none** will disable logging of unsuccessful commands.

syslog_goodpri

Syslog priority to use when the user is allowed to run a command and authentication is successful. Defaults to notice.

See *syslog_badpri* for the list of supported syslog priorities. Negating the option or setting it to a value of **none** will disable logging of successful commands.

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. Many operating systems will remove
potentially dangerous variables from the environment of any set-user-ID 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.

Preserving the HOME environment variable has security implications since many
programs use it when searching for configuration or data files. Adding HOME to
env_keep may enable a user to run unrestricted commands via **sudo** and is strongly
discouraged. Users wishing to edit files with **sudo** should run **sudoedit** (or **sudo -e**)
to get their accustomed editor configuration instead of invoking the editor directly.

log_servers   A list of one or more servers to use for remote event and I/O log storage, separated
by white space. Log servers must be running **sudo_logsrvd** or another service that
implements the protocol described by sudo_logsrv.proto(5).

Server addresses should be of the form "host[:port][(tls)]". The host portion may
be a host name, an IPv4 address, or an IPv6 address in square brackets.

If the optional *tls* flag is present, the connection will be secured with Transport
Layer Security (TLS) version 1.2 or 1.3. Versions of TLS prior to 1.2 are not
supported.

If a port is specified, it may either be a port number or a well-known service name
as defined by the system service name database. If no port is specified, port 30343
will be used for plaintext connections and port 30344 will be used for TLS
connections.

When *log_servers* is set, event log data will be logged both locally (see the *syslog*

and *log_file* settings) as well as remotely, but I/O log data will only be logged remotely. If multiple hosts are specified, they will be attempted in reverse order. If no log servers are available, the user will not be able to run a command unless either the *ignore_iolog_errors* flag (I/O logging enabled) or the *ignore_log_errors* flag (I/O logging disabled) is set. Likewise, if the connection to the log server is interrupted while **sudo** is running, the command will be terminated unless the *ignore_iolog_errors* flag (I/O logging enabled) or the *ignore_log_errors* flag (I/O logging disabled) is set.

This setting is only supported by version 1.9.0 or higher.

passprompt_regex   A list of POSIX extended regular expressions used to match password prompts in the terminal output. As an extension, if the regular expression begins with "(?i)", it will be matched in a case-insensitive manner. Each regular expression is limited to 1024 characters. This option is only used when *log_passwords* has been disabled. The default value is "[Pp]assword[:]*"

This setting is only supported by version 1.9.10 or higher.

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* 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(3) and getgrid(3). 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(5).

EVENT LOGGING

sudoers can log events in either JSON or *sudo* format, this section describes the *sudo* log format. Depending on *sudoers* configuration, **sudoers** can log events via syslog(3), to a local log file, or both. The log format is almost identical in both cases. Any control characters present in the log data are formatted in octal with a leading '#' character. For example, a horizontal tab is stored as '#011' and an embedded carriage return is stored as '#015'. In addition, space characters in the command path are stored as '#040'. Command line arguments that contain spaces are enclosed in single quotes (''). This makes it possible to distinguish multiple command line arguments from a single argument that contains spaces. Literal single quotes and backslash characters ('\') in command line arguments are escaped with a backslash.

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 ; CHROOT=chroot ; \
    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.

chroot   The root directory that the command was run in, if one was specified.

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, including any command line arguments.

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
    The **-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 /usr/local/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 /usr/local/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 */usr/local/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 open /usr/local/etc/sudoers
    The */usr/local/etc/sudoers* file is missing.

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

/usr/local/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, 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.

/usr/local/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.

/usr/local/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, 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/user-ID
  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/user-ID
  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 system's syslog(3) function, not **sudoers** itself. As such, they may vary in format on different systems.

The maximum size of syslog messages varies from system to system. The *syslog_maxlen* setting can be used to change the maximum syslog message size from the default value of 980 bytes. For more information, see the description of *syslog_maxlen*.

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* field is not present.

2. The *hostname* is only logged if the *log_host* option is enabled.

3. The date does not include the year unless the *log_year* option is enabled.

4. 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 LOGGING

When I/O logging is enabled, **sudo** will runs the command in a pseudo-terminal, logging user input and/or output, depending on which **sudoers** flags are enabled. There are five distinct types of I/O that can be logged, each with a corresponding **sudoers** flag.

Type	Flag	Description
terminal input	log_ttyin	keystrokes entered by the user
terminal output	log_ttyout	command output displayed to the screen
standard input	log_stdin	input from a pipe or a file
standard output	log_stdout	output to a pipe or a file
standard error	log_stderr	output to a pipe or a file

In addition to flags described the above, the *log_input* flag and LOG_INPUT command tag set both *log_ttyin* and *log_stdin*. The *log_output* flag and LOG_OUTPUT command tag set *log_ttyout*, *log_stdout*, and *log_stderr*.

To capture terminal input and output, **sudo** run the command in a pseudo-terminal, logging the input and output before passing it on to the user. To capture the standard input, standard output or standard error, **sudo** uses a pipe to interpose itself between the input or output stream, logging the I/O before passing it to the other end of the pipe.

I/O can be logged either to the local machine or to a remote log server. For local logs, 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. For remote logs, the *log_servers* setting is used to specify one or more log servers running **sudo_logsrvd** or another server that implements the protocol described by sudo_logsrv.proto(5).

I/O logging pitfals

When logging standard input, anything sent to the standard input will be consumed, regardless of whether or not the command run via **sudo** is actively reading the standard input. This may have unexpected results when using **sudo** in a shell script that expects to process the standard input. For example, given the following shell script:

```
#!/bin/sh
sudo echo testing
echo done
```

It will behave as expected when the script is passed to the shell as a an argument:

```
$ sh test.sh
testing
done
```

However, if the script is passed to the shell on the standard input, the 'sudo echo testing' command will consume the rest of the script. This means that the 'echo done' statement is never executed.

```
$ sh -s < test.sh
testing
```

There are several ways to work around this problem:

1. Redirect the standard input from */dev/null* when running a command via **sudo** that does not need to read the standard input.

   ```
   sudo echo testing < /dev/null
   ```

2. Pass the script to the shell by path name instead of via the standard input.

   ```
   sh test.sh
   ```

3. Disable logging the standard input for commands that do not need to read the standard input.

   ```
   Defaults!/bin/echo !log_stdin
   ```

Depending on the command, it may not be desirable to log the standard input or standard output. For example, I/O logging of commands that send or receive large amount of data via the standard output or standard input such as rsync(1) and tar(1) could fill up the log file system with superfluous data. It is

possible to disable logging of the standard input and standard output for such commands as follows:

Cmnd_Alias COPY_CMDS = /usr/bin/tar, /usr/bin/cpio, /usr/bin/rsync

Log input and output but omit stdin and stdout when copying files.
Defaults log_input, log_output
Defaults!COPY_CMDS !log_stdin, !log_stdout

However, be aware that using the *log_input* flag or the LOG_INPUT command tag will also enable *log_stdin*. Likewise, the *log_ouput* flag or the LOG_OUTPUT command tag will enable *log_stdout* and *log_stderr.* Careful ordering of rules may be necessary to achieve the results that you expect.

I/O log format
For both local and remote I/O logs, each log is stored in a separate directory that contains the following files:

log   A text file containing information about the command. The first line consists of the following colon-delimited fields: 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, and the number of lines and columns of the terminal. The second and third lines contain the working directory the command was run from and the path name of the command itself (with arguments if present).

log.json A JSON-formatted file containing information about the command. This is similar to the *log* file but contains additional information and is easily extensible. The *log.json* file will be used by sudoreplay(8) in preference to the *log* file if it exists. The file may contain the following elements:

timestamp
        A JSON object containing time the command was run. It consists of two values, *seconds* and *nanoseconds*.

columns
        The number of columns of the terminal the command ran on, or zero if no terminal was present.

command
        The fully-qualified path of the command that was run.

lines   The number of lines of the terminal the command ran on, or zero if no terminal was

present.

runargv

A JSON array representing the command's argument vector as passed to the execve(2) system call.

runenv

A JSON array representing the command's environment as passed to the execve(2) system call.

rungid

The group ID the command ran as. This element is only present when the user specifies a group on the command line.

rungroup

The name of the group the command ran as. This element is only present when the user specifies a group on the command line.

runuid

The user ID the command ran as.

runuser

The name of the user the command ran as.

submitcwd

The current working directory at the time **sudo** was run.

submithost

The name of the host the command was run on.

submituser

The name of the user who ran the command via **sudo**.

ttyname

The path name of the terminal the user invoked **sudo** from. If the command was run in a pseudo-terminal, *ttyname* will be different from the terminal the command actually ran in.

timing   Timing information used to replay the session. Each line consists of the I/O log entry type and amount of time since the last entry, followed by type-specific data. The I/O log entry types and

their corresponding type-specific data are:

    0   standard input, number of bytes in the entry
    1   standard output, number of bytes in the entry
    2   standard error, number of bytes in the entry
    3   terminal input, number of bytes in the entry
    4   terminal output, number of bytes in the entry
    5   window change, new number lines and columns
    6   bug compatibility for **sudo** 1.8.7 terminal output
    7   command suspend or resume, signal received

ttyin   Raw input from the user's terminal, exactly as it was received. This file is only present if the *log_input* or *log_ttyin* flags are set and **sudo** was run from a terminal. No post-processing is performed. For manual viewing, you may wish to convert carriage return characters in the log to line feeds. For example: 'gunzip -c ttyin | tr "\r" "\n"'

stdin   The standard input when no terminal is present, or input redirected from a pipe or file. This file is only present if the *log_input* or *log_stdin* flags are set and the standard input is not connected to a terminal.

ttyout   Output from the pseudo-terminal (what the command writes to the screen). Terminal-specific post-processing is performed before the data is logged. This means that, for example, line feeds are usually converted to line feed/carriage return pairs and tabs may be expanded to spaces. This file is only present if the *log_output* or *log_ttyout* flags are set and **sudo** was run from a terminal.

stdout   The standard output when no terminal is present, or output redirected to a pipe or file. This file is only present if the *log_output* or *log_stdout* flags are set and the standard output is not connected to a terminal.

stderr   The standard error when no terminal is present, or output redirected to a pipe or file. This file is only present if the *log_output* or *log_stderr* flags are set and the standard error is not connected to a terminal.

All files other than *log* are compressed in gzip format unless the *compress_io* flag has been disabled. Due to buffering, it is not normally possible to display the I/O logs in real-time as the program is executing. The I/O log data will not be complete until the program run by **sudo** has exited or has been terminated by a signal. The *iolog_flush* flag can be used to disable buffering, in which case I/O log data is written to disk as soon as it is available. 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.

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. When logging input, consider disabling the *log_passwords* flag.

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

/usr/local/etc/sudo.conf	Sudo front-end configuration
/usr/local/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/db/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.  Other programs use HOME to locate 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          WEBADMIN = 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 = primary, 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 and for commands to be run with the target user's home directory as the working directory. We don't want to subject the full time staff to the **sudo** lecture and we want to allow them to run commands in a chroot(2) "sandbox" via the **-R** option. User **millert** need not provide a password and we don't want to reset the LOGNAME or USER 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*). This will not effectively constrain users with **sudo ALL**

privileges.

Override built-in defaults
Defaults   syslog=auth,runcwd=~
Defaults>root   !set_logname
Defaults:FULLTIMERS   !lecture,runchroot=*
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/*. 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. On systems with GNU getopt(3), 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* (primary, 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.

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

On the host www, any user in the WEBADMIN *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.

Another potential issue is that when **sudo** executes the command, it must use the command or path specified by the user instead of a path listed in the *sudoers* file. This may lead to a time of check versus time of use race condition.

Wildcards in command arguments

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. A safer alternative is to use a regular expression for matching command line arguments. The above example can be rewritten as a regular expression:

    %operator ALL = /bin/cat ^/var/log/messages[^[:space:]]*$

The regular expression will only match a single file with a name that begins with */var/log/messages* and does not include any white space in the name. It is often better to do command line processing outside of the *sudoers* file in a scripting language for anything non-trivial.

Regular expressions in command names

Using a regular expression to match a command name has the same security implications as using the *fast_glob* option:

⊕ It is not possible to reliably negate commands when the path name is a regular expression.

⊕ When **sudo** executes the command, it must use the command or path specified by the user instead of a path listed in the *sudoers* file. This may lead to a time of check versus time of use race condition.

These issues do not apply to rules where only the command line options are matched using a regular expression.

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 four 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.

intercept On most systems, **sudo**'s *intercept* functionality can be used to transparently intercept an attempt to run a new command, allow or deny it based on *sudoers* rules, and log the result. For example, this can be used to restrict the commands run from within a privileged shell or editor. However, not all programs operate correctly when *intercept* is enabled.

There are two underlying mechanisms that may be used to implement *intercept* mode: *dso* and *trace*. The *intercept_type* setting can be used to select between them.

The first mechanism, *dso*, overrides the standard C library functions that are used to execute a command. It does this by setting an environment variable (usually LD_PRELOAD) to the path of a dynamic shared object, or shared library, containing custom versions of the execve(2), execl(3), execle(3), execlp(3), execv(3), execvp(3), execvpe(3), and system(3) library functions that connect back to **sudo** for a policy decision. Note, however, that this applies only to dynamically-linked executables. It is not possible to intercept commands for statically-linked executables or executables that run under binary emulation this way. Because most dynamic loaders ignore LD_PRELOAD (or the equivalent) when running set-user-ID and set-group-ID programs, **sudoers** will not permit such programs to be run in *intercept* mode by default. The *dso* mechanism is incompatible with **sudo**'s SELinux RBAC support (but see below). SELinux disables LD_PRELOAD by default and interferes with file descriptor inheritance, which **sudo** relies on.

The second mechanism, *trace*, is available on Linux systems that support seccomp(2) filtering. It uses ptrace(2) and seccomp(2) to intercept the execve(2) system call instead of pre-loading a dynamic shared object. Both static and dynamic executables are supported and it is compatible with **sudo**'s SELinux RBAC mode. Functions utilizing the execveat(2) system call, such as fexecve(3), are not currently intercepted. Programs that rely on ptrace(2) themselves, such as debuggers and system call tracers (such as strace(1) and truss(1)) will be unable to function if *intercept* is enabled in *trace* mode. This same restriction applies to the *log_subcmds* sudoers option.

The *intercept* feature is known to work on Solaris, *BSD, Linux, macOS, 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 or an equivalent. It is not possible to intercept shell built-in commands or restrict the ability to read or write sensitive files from within a shell.

To enable intercept mode on a per-command basis, use the INTERCEPT tag as documented in the User Specification section above. Here is that example again:

chuck   research = INTERCEPT: ALL

This allows user **chuck** to run any command on the machine "research" in intercept mode. Any commands run via shell escapes will be validated and logged by **sudo**. If you are unsure whether or not your system is capable of supporting *intercept*, you can always just try it out and check whether or not external commands run via a shell are logged when *intercept* is enabled.

There is an inherent race condition between when a command is checked against **sudoers** rules and when it is actually executed. If a user is allowed to run arbitrary commands, they may be able to change the execve(2) arguments in the program after the **sudoers** policy check has completed but before the new command is executed. Starting with version 1.9.12, the *trace* method will verify that the command and its arguments have not changed after execve(2) has completed but before execution of the new program has had a chance to run. This is not the case with the *dso* method. See the description of the *intercept_verify* setting for more information.

log   There are two separate but related ways to log additional commands. The first is to enable I/O logging using the *log_output* flag. This will log the command's output but will not create an event log entry when the additional command is run. The second is to enable the *log_subcmds* flag in *sudoers* which will create an event log entry every time a new command is run. If I/O logging is also enabled, the log entry will include a time offset into the I/O log to indicate when the command was run. This offset can be passed to the sudoreplay(8) utility to replay the I/O log at the exact moment when the command was run. The *log_subcmds* flag uses the same mechanism as *intercept* (see above) and has the same limitations.

noexec   **sudo**'s *noexec* functionality can be used to prevent a program run by **sudo** from executing any other programs. On most systems, it uses the same LD_PRELOAD mechanism as *intercept* (see above) and thus the same caveats apply. The *noexec* functionality is capable of blocking execution of commands run via the execve(2), execl(3), execle(3), execlp(3), exect(3), execv(3), execveat(3), execvP(3), execvp(3), execvpe(3), fexecve(3), popen(3), posix_spawn(3), posix_spawnp(3), system(3), and wordexp(3) functions. On Linux, a seccomp(2) filter is used to implement *noexec*. 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.

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 on any machine:

    operator ALL = 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 */run* or */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.

If the *timestamp_type* option is set to "tty", the time stamp record includes the device number of the terminal the user authenticated with. This provides per-terminal granularity but time stamp records may still outlive the user's session.

Unless the *timestamp_type* option is set to "global", 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. On systems where a process's start time can be queried, the start time of the session leader is recorded in the time stamp record. If no terminal is present or the *timestamp_type* option is set to "ppid", the start time of the parent process is used instead. In most cases this will prevent a time stamp record from being re-used without the user entering a password when logging out and back in again.

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-terminal related code

rbtree   redblack tree internals

sssd   SSSD-based sudoers

util   utility functions

For example:

Debug sudoers.so /var/log/sudoers_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), sudo_plugin(5), sudoers.ldap(5), sudoers_timestamp(5), sudo(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.md file in the **sudo** distribution (https://www.sudo.ws/about/contributors/) for an exhaustive list of people who have contributed to **sudo**.

CAVEATS
The *sudoers* file should **always** be edited by the **visudo** utility which locks the file and checks for syntax errors. If *sudoers* contains syntax errors, **sudo** may refuse to run, which is a serious problem if **sudo** is your only method of obtaining superuser privileges. Recent versions of **sudoers** will attempt to recover after a syntax error by ignoring the rest of the line after encountering an error. Older versions of **sudo** will not run if *sudoers* contains a syntax error.

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 believe you have found a bug in **sudo**, you can 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

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