NAME

sudo_plugin_python - Sudo Plugin API (Python)

DESCRIPTION

Starting with version 1.9, **sudo** plugins can be written in python. The API closely follows the C **sudo** plugin API described by sudo_plugin(5).

The supported plugins types are:

- Policy plugin
- I/O plugin
- Audit plugin
- Approval plugin
- Group provider plugin

Python plugin support needs to be explicitly enabled at build time with the configure option "--enable-python". Python version 3.0 or higher is required.

Sudo Python Plugin Base

A plugin written in Python should be a class in a python file that inherits from *sudo.Plugin*. The *sudo.Plugin* base class has no real purpose other than to identify this class as a plugin.

The only implemented method is a constructor, which stores the keyword arguments it receives as fields (member variables) in the object. This is intended as a convenience to allow you to avoid writing the constructor yourself.

For example:

```
import sudo

class MySudoPlugin(sudo.Plugin):
    # example constructor (optional)
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    # example destructor (optional)
    def __del__(self):
        pass
```

Both the constructor and destructor are optional and can be omitted.

The customized Plugin class should define a few plugin-specific methods. When the plugin loads, **sudo** will create an instance of this class and call the methods. The actual methods required depent on the type of the plugin, but most return an "int" result code, as documented in sudo_plugin(@mansctsu@), that indicates whether or not the method was successful. The Python sudo module defines the following constants to improve readability:

Define	**Value**
sudo.RC.OK	1
sudo.RC.ACCEPT	1
sudo.RC.REJECT	0
sudo.RC.ERROR	-1
sudo.RC.USAGE_ERROR	-2

If a function returns *None* (for example, if it does not call return), it will be considered to have returned sudo.RC.OK. If an exception is raised (other than sudo.PluginException), the backtrace will be shown to the user and the plugin function will return sudo.RC.ERROR. If that is not acceptable, you must catch the exception and handle it yourself.

Instead of just returning sudo.RC.ERROR or sudo.RC.REJECT result code the plugin can also provide a message describing the problem. This can be done by raising one of the special exceptions:

```
raise sudo.PluginError("Message")
raise sudo.PluginReject("Message")
```

This added message will be used by the audit plugins. Both exceptions inherit from sudo.PluginException

Python Plugin Loader

Running the Python interpreter and bridging between C and Python is handled by the **sudo** plugin python_plugin.so. This shared object can be loaded like any other dynamic **sudo** plugin and should receive the path and the class name of the Python plugin it is loading as arguments.

Example usage in sudo.conf(5):

```
Plugin python_policy python_plugin.so ModulePath=<path> ClassName=<class>
Plugin python_io python_plugin.so ModulePath=<path> ClassName=<class>
Plugin python_audit python_plugin.so ModulePath=<path> ClassName=<class>
Plugin python_approval python_plugin.so ModulePath=<path> ClassName=<class>
```

Example group provider plugin usage in the *sudoers* file:

Defaults group_plugin="python_plugin.so ModulePath=<path> ClassName=<class>"

The plugin arguments are as follows:

ModulePath
: The path of a python file which contains the class of the sudo Python plugin. It must be either an absolute path or a path relative to the sudo Python plugin directory: "/usr/local/libexec/sudo/python".

ClassName
: (Optional.) The name of the class implementing the sudo Python plugin. If not supplied, the one and only sudo.Plugin that is present in the module will be used. If there are multiple such plugins in the module (or none), it will result in an error.

Policy plugin API

Policy plugins must be registered in sudo.conf(5). For example:

Plugin python_policy python_plugin.so ModulePath=<path> ClassName=<class>

Currently, only a single policy plugin may be specified in sudo.conf(5).

A policy plugin may have the following member functions:

constructor

```
__init__(self, user_env: Tuple[str, ...], settings: Tuple[str, ...],
    version: str, user_info: Tuple[str, ...],
    plugin_options: Tuple[str, ...])
```

Implementing this function is optional. The default constructor will set the keyword arguments it receives as member variables in the object.

The constructor matches the **open**() function in the C sudo plugin API.

The function arguments are as follows:

user_env
: The user's environment as a tuple of strings in "key=value" format.

settings

A tuple of user-supplied *sudo* settings in the form of "key=value" strings.

version
The version of the Python Policy Plugin API.

user_info
A tuple of information about the user running the command in the form of "key=value" strings.

plugin_options
The plugin options passed as arguments in the sudo.conf(5) plugin registration. This is a tuple of strings, usually (but not necessarily) in "key=value" format.

The **sudo.options_as_dict**() convenience function can be used to convert "key=value" pairs to a dictionary. For a list of recognized keys and their supported values, see the policy plugin **open**() documentation in sudo_plugin(5).

check_policy
check_policy(self, argv: Tuple[str, ...], env_add: Tuple[str, ...])

The **check_policy**() function is called by **sudo** to determine whether the user is allowed to run the specified command. Implementing this function is mandatory for a policy plugin.

The function arguments are as follows:

argv A tuple describing the command the user wishes to run.

env_add
Additional environment variables specified by the user on the command line in the form of a tuple of "key=value" pairs. The **sudo.options_as_dict**() convenience function can be used to convert them to a dictionary.

This function should return a result code or a tuple in the following format:

return (rc, command_info_out, argv_out, user_env_out)

The tuple values are as follows:

rc The result of the policy check, one of the sudo.RC.* constants. sudo.RC.ACCEPT if the command is allowed, sudo.RC.REJECT if not allowed, sudo.RC.ERROR for a general

error, or sudo.RC.USAGE_ERROR for a usage error.

command_info_out
Optional (only required when the command is accepted). Information about the command being run in the form of "key=value" strings.

To accept a command, at the very minimum the plugin must set in the *command*, *runas_uid* and *runas_gid* keys.

For a list of recognized keys and supported values, see the **check_policy**() documentation in sudo_plugin(5).

argv_out
Optional (only required when the command is accepted). The arguments to pass to the execve(2) system call when executing the command.

user_env_out
Optional (only required when the command is accepted). The environment to use when executing the command in the form of a tuple of strings in "key=value" format.

init_session
init_session(self, user_pwd: Tuple, user_env: Tuple[str, ...])

Perform session setup (optional). The **init_session**() function is called before **sudo** sets up the execution environment for the command before any uid or gid changes.

The function arguments are as follows:

user_pwd
A tuple describing the user's passwd entry. Convertible to pwd.struct_passwd or *None* if the user is not present in the password database.

Example conversion:
user_pwd = pwd.struct_passwd(user_pwd) if user_pwd else None

user_env
The environment the command will run in. This is a tuple of strings in "key=value" format.

This function should return a result code or a tuple in the following format:

return (rc, user_env_out)

The tuple values are as follows:

rc The result of the session init, one of the sudo.RC.* constants. sudo.RC.OK on success, 0 on failure, or sudo.RC.ERROR if an error occurred.

user_env_out
Optional. If the **init_session**() function needs to modify the user environment, it can return the new environment in *user_env_out*. If this is omitted, no changes will be made to *user_env*.

list

list(self, argv: Tuple[str, ...], is_verbose: int, user: str)

List available privileges for the invoking user.

The function arguments are as follows:

argv If not set to *None*, an argument vector describing a command the user wishes to check against the policy.

is_verbose
Flag indicating whether to list in verbose mode or not.

user The name of a different user to list privileges for if the policy allows it. If *None*, the plugin should list the privileges of the invoking user.

validate

validate(self)

For policy plugins that cache authentication credentials, this function is used to validate and cache the credentials (optional).

invalidate

invalidate(self, remove: int)

For policy plugins that cache authentication credentials, this function is used to invalidate the credentials (optional).

The function arguments are as follows:

remove
If this flag is set, the plugin may remove the credentials instead of simply invalidating them.

show_version
show_version(self, is_verbose: int)

Display the plugin version information to the user. The **sudo.log_info**() function should be used.

The function arguments are as follows:

is_verbose
A flag to indicate displaying more verbose information. Currently this is 1 if ‘sudo -V’ is run as the root user.

close
close(self, exit_status: int, error: int)

Called when a command finishes executing.

Works the same as the **close**() function in the C sudo plugin API, except that it only gets called if **sudo** attempts to execute the command.

The function arguments are as follows:

exit_status
The exit status of the command if was executed, otherwise -1.

error If the command could not be executed, this is set to the value of errno set by the execve(2) system call, otherwise 0.

Policy plugin example

Sudo ships with an example Python policy plugin. To try it, register it by adding the following lines to */etc/sudo.conf*:

```
Plugin python_policy python_plugin.so \
  ModulePath=/usr/local/share/doc/sudo/examples/example_policy_plugin.py \
  ClassName=SudoPolicyPlugin
```

Be aware, however, that you cannot enable the Python policy plugin in addition to another policy plugin, such as sudoers(5).

I/O plugin API

I/O plugins must be registered in sudo.conf(5). For example:

```
Plugin python_io python_plugin.so ModulePath=<path> ClassName=<class>
```

Sudo supports loading multiple I/O plugins. Currently only 8 python I/O plugins can be loaded at once.

An I/O plugin may have the following member functions:

constructor

```
__init__(self, user_env: Tuple[str, ...], settings: Tuple[str, ...],
  version: str, user_info: Tuple[str, ...],
  plugin_options: Tuple[str, ...])
```

Implementing this function is optional. The default constructor will set the keyword arguments it receives as member variables in the object.

The constructor matches the **open**() function in the C sudo plugin API.

The function arguments are as follows:

user_env
: The user's environment as a tuple of strings in "key=value" format.

settings
: A tuple of user-supplied *sudo* settings in the form of "key=value" strings.

version
: The version of the Python I/O Plugin API.

user_info
: A tuple of information about the user running the command in the form of "key=value" strings.

plugin_options
: The plugin options passed as arguments in the sudo.conf(5) plugin registration. This is a tuple of strings, usually (but not necessarily) in "key=value" format.

The **sudo.options_as_dict**() convenience function can be used to convert "key=value" pairs to a dictionary. For a list of recognized keys and their supported values, see the I/O plugin **open**() documentation in sudo_plugin(5).

open

open(self, argv: Tuple[str, ...],
command_info: Tuple[str, ...]) -> int

Receives the command the user wishes to run.

Works the same as the **open**() function in the C sudo plugin API except that:

- It only gets called before the user would execute some command (and not for a version query for example).
- Other arguments of the C API **open**() function are received through the constructor.

The function arguments are as follows:

argv A tuple of the arguments describing the command the user wishes to run.

command_info
Information about the command being run in the form of "key=value" strings.

The **sudo.options_as_dict**() convenience function can be used to convert "key=value" pairs to a dictionary. For a list of recognized keys and their supported values, see the I/O plugin **open**() documentation in sudo_plugin(5).

The **open**() function should return a result code, one of the sudo.RC.* constants. If the function returns sudo.RC.REJECT, no I/O will be sent to the plugin.

log_ttyin, **log_ttyout**, **log_stdin**, **log_stdout**, **log_stderr**

log_ttyin(self, buf: str) -> int
log_ttyout(self, buf: str) -> int
log_stdin(self, buf: str) -> int
log_stdout(self, buf: str) -> int
log_stderr(self, buf: str) -> int

Receive the user input or output of the terminal device and application standard input / output / error. See the matching calls in sudo_plugin(5).

The function arguments are as follows:

buf The input (or output) buffer in the form of a string.

The function should return a result code, one of the sudo.RC.* constants.

If sudo.RC.ERROR is returned, the running command will be terminated and all of the plugin's logging functions will be disabled. Other I/O logging plugins will still receive any remaining input or output that has not yet been processed.

If an input logging function rejects the data by returning sudo.RC.REJECT, the command will be terminated and the data will not be passed to the command, though it will still be sent to any other I/O logging plugins. If an output logging function rejects the data by returning sudo.RC.REJECT, the command will be terminated and the data will not be written to the terminal, though it will still be sent to any other I/O logging plugins.

change_winsize

change_winsize(self, line: int, cols: int) -> int

Called whenever the window size of the terminal changes. The function arguments are as follows:

line The number of lines of the terminal.

cols The number of columns of the terminal.

log_suspend

log_suspend(self, signo: int) -> int
Called whenever a command is suspended or resumed.

The function arguments are as follows:

signo
The number of the signal that caused the command to be suspended or SIGCONT if the command was resumed.

show_version

show_version(self, is_verbose: int)
Display the plugin version information to the user. The **sudo.log_info**() function should be used.

The function arguments are as follows:

is_verbose
A flag to indicate displaying more verbose information. Currently this is 1 if ‘sudo -V’ is run as the root user.

close

close(self, exit_status: int, error: int) -> None
Called when a command execution finished.

Works the same as the **close**() function in the C sudo plugin API, except that it only gets called if **sudo** attempts to execute the command.

The function arguments are as follows:

exit_status
The exit status of the command if was executed, otherwise -1.

error If the command could not be executed, this is set to the value of errno set by the execve(2) system call, otherwise 0.

I/O plugin example

Sudo ships a Python I/O plugin example. To try it, register it by adding the following lines to */etc/sudo.conf*:

```
Plugin python_io python_plugin.so \
  ModulePath=/usr/local/share/doc/sudo/examples/example_io_plugin.py \
  ClassName=SudoIOPlugin
```

Audit plugin API

Audit plugins must be registered in sudo.conf(5). For example:

```
Plugin python_audit python_plugin.so ModulePath=<path> ClassName=<class>
```

Sudo supports loading multiple audit plugins. Currently only 8 python audit plugins can be loaded at once.

An audit plugin may have the following member functions (all of them are optional):

constructor

__init__(self, user_env: Tuple[str, ...], settings: Tuple[str, ...],
version: str, user_info: Tuple[str, ...], plugin_options: Tuple[str, ...])

The default constructor will set the keyword arguments it receives as member variables in the object.

The constructor matches the **open**() function in the C sudo plugin API.

The function arguments are as follows:

user_env
The user's environment as a tuple of strings in "key=value" format.

settings
A tuple of user-supplied *sudo* settings in the form of "key=value" strings.

version
The version of the Python Audit Plugin API.

user_info
A tuple of information about the user running the command in the form of "key=value" strings.

plugin_options
The plugin options passed as arguments in the sudo.conf(5) plugin registration. This is a tuple of strings, usually (but not necessarily) in "key=value" format.

open

open(self, submit_optind: int,
submit_argv: Tuple[str, ...]) -> int

The function arguments are as follows:

submit_optind
The index into *submit_argv* that corresponds to the first entry that is not a command line option.

submit_argv
The argument vector sudo was invoked with, including all command line options.

close

close(self, status_type: int, status: int) -> None

Called when sudo is finished, shortly before it exits.

The function arguments are as follows:

status_type
The type of status being passed. One of the sudo.EXIT_REASON.* constants.

status
Depending on the value of *status_type*, this value is either ignored, the command's exit status as returned by the wait(2) system call, the value of errno set by the execve(2) system call, or the value of errno resulting from an error in the **sudo** front end.

show_version
show_version(self, is_verbose: int) -> int

Display the plugin version information to the user. The **sudo.log_info**() function should be used.

The function arguments are as follows:

is_verbose
A flag to indicate displaying more verbose information. Currently this is 1 if 'sudo -V' is run as the root user.

accept
accept(self, plugin_name: str, plugin_type: int, command_info: Tuple[str, ...],
run_argv: Tuple[str, ...], run_envp: Tuple[str, ...]) -> int

This function is called when a command or action is accepted by the policy plugin. The function arguments are as follows:

plugin_name
The name of the plugin that accepted the command.

plugin_type
The type of plugin that accepted the command, currently always sudo.PLUGIN_TYPE.POLICY.

command_info
A vector of information describing the command being run. See the sudo_plugin(5) manual for possible values.

run_argv
Argument vector describing a command that will be run.

run_envp
The environment the command will be run with.

reject

reject(self, plugin_name: str, plugin_type: int, audit_msg: str,
command_info: Tuple[str, ...]) -> int

This function is called when a command or action is rejected by the policy plugin. The function arguments are as follows:

plugin_name
The name of the plugin that accepted the command.

plugin_type
The type of plugin that accepted the command, currently always sudo.PLUGIN_TYPE.POLICY.

audit_msg
An optional string describing the reason the command was rejected by the plugin. If the plugin did not provide a reason, audit_msg will be *None*

command_info
A vector of information describing the rejected command. See the sudo_plugin(5) manual for possible values.

error

error(self, plugin_name: str, plugin_type: int, audit_msg: str,
command_info: Tuple[str, ...]) -> int

This function is called when a plugin returns an error. The function arguments are as follows:

plugin_name
The name of the plugin that accepted the command.

plugin_type
The type of plugin that accepted the command, currently sudo.PLUGIN_TYPE.POLICY or sudo.PLUGIN_TYPE.IO

audit_msg
An optional string describing the plugin error. If the plugin did not provide a description, it will be *None*

command_info
A vector of information describing the command. See the sudo_plugin(5) manual for possible values.

Audit plugin example

Sudo ships a Python Audit plugin example. To try it, register it by adding the following lines to */etc/sudo.conf*:

```
Plugin python_audit python_plugin.so \
  ModulePath=/usr/local/share/doc/sudo/examples/example_audit_plugin.py \
  ClassName=SudoAuditPlugin
```

It will log the plugin accept / reject / error results to the output.

Approval plugin API

Approval plugins must be registered in sudo.conf(5). For example:

```
Plugin python_approval python_plugin.so ModulePath=<path> ClassName=<class>
```

Sudo supports loading multiple approval plugins. Currently only 8 python approval plugins can be loaded at once.

An approval plugin may have the following member functions:

constructor

```
__init__(self, user_env: Tuple[str, ...], settings: Tuple[str, ...],
  version: str, user_info: Tuple[str, ...], plugin_options: Tuple[str, ...],
  submit_optind: int, submit_argv: Tuple[str, ...])
```

Optional. The default constructor will set the keyword arguments it receives as member variables in the object.

The constructor matches the **open**() function in the C sudo plugin API.

The function arguments are as follows:

user_env
The user’s environment as a tuple of strings in "key=value" format.

settings
A tuple of user-supplied *sudo* settings in the form of "key=value" strings.

version
The version of the Python Approval Plugin API.

user_info
A tuple of information about the user running the command in the form of "key=value" strings.

plugin_options
The plugin options passed as arguments in the sudo.conf(5) plugin registration. This is a tuple of strings, usually (but not necessarily) in "key=value" format.

submit_optind
The index into *submit_argv* that corresponds to the first entry that is not a command line option.

submit_argv
The argument vector sudo was invoked with, including all command line options.

show_version
show_version(self, is_verbose: int) -> int

Display the version. (Same as for all the other plugins.)

check
check(self, command_info: Tuple[str, ...], run_argv: Tuple[str, ...],
run_env: Tuple[str, ...]) -> int

This function is called after policy plugin’s check_policy has succeeded. It can reject execution of the command by returning sudo.RC.REJECT or raising the special exception:

raise sudo.PluginReject("some message")

with the message describing the problem. In the latter case, the audit plugins will get the description.

The function arguments are as follows:

command_info
: A vector of information describing the command that will run. See the sudo_plugin(5) manual for possible values.

run_argv
: Argument vector describing a command that will be run.

run_env
: The environment the command will be run with.

Approval plugin example

Sudo ships a Python Approval plugin example. To try it, register it by adding the following lines to */etc/sudo.conf*:

```
Plugin python_approval python_plugin.so \
    ModulePath=/usr/local/share/doc/sudo/examples/example_approval_plugin.py \
    ClassName=BusinessHoursApprovalPlugin
```

It will only allow execution of commands in the "business hours" (from Monday to Friday between 8:00 and 17:59:59).

Sudoers group provider plugin API

A group provider plugin is registered in the sudoers(5) file. For example:

```
Defaults group_plugin="python_plugin.so ModulePath=<path> ClassName=<class>"
```

Currently, only a single group plugin can be registered in *sudoers*.

A group provider plugin may have the following member functions:

constructor
: __init__(self, args: Tuple[str, ...], version: str)

  Implementing this function is optional. The default constructor will set the keyword arguments it receives as member variables in the object.

  The function arguments are as follows:

args The plugin options passed as arguments in the *sudoers* file plugin registration. All the arguments are free form strings (not necessarily in "key=value" format).

version
The version of the Python Group Plugin API.

query
query(self, user: str, group: str, user_pwd: Tuple)

The **query**() function is used to ask the group plugin whether *user* is a member of *group*. This method is required.

The function arguments are as follows:

user The name of the user being looked up in the external group database.

group
The name of the group being queried.

user_pwd
The password database entry for the user, if any. If *user* is not present in the password database, *user_pwd* will be NULL.

Group plugin example

Sudo ships a Python group plugin example. To try it, register it in the *sudoers* file by adding the following lines:

```
Defaults group_plugin="python_plugin.so \
    ModulePath=/usr/local/share/doc/sudo/examples/example_group_plugin.py \
    ClassName=SudoGroupPlugin"
```

The example plugin will tell **sudo** that the user *test* is part of the non-unix group *mygroup*. If you add a rule that uses this group, it will affect the *test* user. For example:

```
%:mygroup ALL=(ALL) NOPASSWD: ALL
```

Will allow user *test* to run **sudo** without a password.

Hook function API

The hook function API is currently not supported for plugins written in Python.

Conversation API

A Python plugin can interact with the user using the **sudo.conv**() function which displays one or more messages described by the **sudo.ConvMessage** class. This is the Python equivalent of the **conversation**() function in the C sudo plugin API. A plugin should not attempt to read directly from the standard input or the user's tty (neither of which are guaranteed to exist).

The **sudo.ConvMessage** class specifies how the user interaction should occur:

```
sudo.ConvMessage(msg_type: int, msg: str, timeout: int)
```

sudo.ConvMessage member variables:

msg_type
: Specifies the type of the conversation. See the sudo.CONV.* constants below.

msg
: The message to display to the user. The caller must include a trailing newline in msg if one is to be displayed.

timeout
: Optional. The maximum amount of time for the conversation in seconds. If the timeout is exceeded, the **sudo.conv**() function will raise a sudo.ConversationInterrupted exception. The default is to wait forever (no timeout).

To specify the message type, the following constants are available:

- sudo.CONV.PROMPT_ECHO_OFF
- sudo.CONV.PROMPT_ECHO_ON
- sudo.CONV.ERROR_MSG
- sudo.CONV.INFO_MSG
- sudo.CONV.PROMPT_MASK
- sudo.CONV.PROMPT_ECHO_OK
- sudo.CONV.PREFER_TTY

See the sudo_plugin(5) manual for a description of the message types.

The **sudo.conv**() function performs the actual user interaction:

```
sudo.conv(message(s), on_suspend=suspend_function,
  on_resume=resume_function)
```

The function arguments are as follows:

message(s)
: One of more messages (of type **sudo.ConvMessage**), each describing a conversation. At least one message is required.

on_suspend
: An optional callback function which gets called if the conversation is suspended, for example by the user pressing control-Z. The specified function must take a single argument which will be filled with the number of the signal that caused the process to be suspended.

on_resume
: An optional callback function which gets called when the previously suspended conversation is resumed. The specified function must take a single argument which will be filled with the number of the signal that caused the process to be suspended.

The **sudo.conv**() function can raise the following exceptions:

sudo.SudoException
: If the conversation fails, for example when the conversation function is not available.

sudo.ConversationInterrupted
: If the conversation function returns an error, e.g., the timeout passed or the user interrupted the conversation by pressing control-C.

Conversation example

Sudo ships with an example plugin demonstrating the Python conversation API. To try it, register it by adding the following lines to */etc/sudo.conf*:

```
Plugin python_io python_plugin.so \
    ModulePath=/usr/local/share/doc/sudo/examples/example_conversation.py \
    ClassName=ReasonLoggerIOPlugin
```

Information / error display API

```
sudo.log_info(string(s), sep=" ", end="\n")
sudo.log_error(string(s), sep=" ", end="\n")
```

To display information to the user, the **sudo.log_info**() function can be used. To display error messages, use **sudo.log_error**(). The syntax is similar to the Python **print**() function.

The function arguments are as follows:

string(s)
: One or more strings to display.

sep An optional string which will be used as the separator between the specified strings. The default is a space character, (‘ ’).

end An optional string which will be displayed at the end of the message. The default is a new line character (‘\n’).

Debug API

Debug messages are not visible to the user and are only logged debugging is explicitly enabled in sudo.conf(5). Python plugins can use the **sudo.debug**() function to make use of **sudo**’s debug system.

Enabling debugging in sudo.conf

To enable debug messages, add a Debug line to sudo.conf(5) with the program set to *python_plugin.so*. For example, to store debug output in */var/log/sudo_python_debug*, use a line like the following:

```
Debug python_plugin.so /var/log/sudo_python_debug \
  plugin@trace,c_calls@trace
```

The debug options are in the form of multiple "subsystem@level" strings, separated by commas (‘,’). For example to just see the debug output of **sudo.debug**() calls, use:

```
Debug python_plugin.so /var/log/sudo_python_debug plugin@trace
```

See sudo_conf(5) for more details.

The most interesting subsystems for Python plugin development are:

plugin
: Logs each **sudo.debug**() API call.

py_calls
: Logs whenever a C function calls into the python module. For example, calling the **__init__**() function.

c_calls

Logs whenever python calls into a C **sudo** API function.

internal
Logs internal functions of the python language wrapper plugin.

sudo_cb
Logs when **sudo** calls into the python plugin API.

load Logs python plugin loading / unloading events.

You can also specify "all" as the subsystem name to log debug messages for all subsystems.

The **sudo.debug**() function is defined as:

sudo.debug(level, message(s))

The function arguments are as follows:

level an integer, use one of the log level constants below

message(s)
one or more messages to log

Available log levels:

sudo.conf name	**Python constant**	**description**
crit	sudo.DEBUG.CRIT	only critical messages
err	sudo.DEBUG.ERROR	
warn	sudo.DEBUG.WARN	
notice	sudo.DEBUG.NOTICE	
diag	sudo.DEBUG.DIAG	
info	sudo.DEBUG.INFO	
trace	sudo.DEBUG.TRACE	

debug sudo.DEBUG.DEBUG
very extreme verbose debugging

Using the logging module

Alternatively, a plugin can use the built in logging module of Python as well. Sudo adds its log handler to the root logger, so by default all output of a logger will get forwarded to sudo log system, as it would call sudo.debug.

The log handler of sudo will map each Python log level of a message to the appropriate sudo debug level. Note however, that sudo debug system will only get the messages not filtered out by the Python loggers. For example, the log level of the python logger will be an additional filter for the log messages, and is usually very different from what level is set in sudo.conf for the sudo debug system.

Debug example

Sudo ships an example debug plugin by default. To try it, register it by adding the following lines to */etc/sudo.conf*:

```
Plugin python_io python_plugin.so \
    ModulePath=/usr/local/share/doc/sudo/examples/example_debugging.py \
    ClassName=DebugDemoPlugin

Debug python_plugin.so \
    /var/log/sudo_python_debug plugin@trace,c_calls@trace
```

Option conversion API

The Python plugin API includes two convenience functions to convert options in "key=value" format to a dictionary and vice versa.

options_as_dict
options_as_dict(options)

The function arguments are as follows:

options
An iterable (tuple, list, etc.) of strings, each in "key=value" format. This is how the plugin API passes options and settings to a Python plugin.

The function returns the resulting dictionary. Each string of the passed in *options* will be split at the first equal sign (‘=’) into a *key* and *value*. Dictionary keys will never contain this symbol (but

values may).

options_from_dict
options_from_dict(options_dict)

The function arguments are as follows:

options_dict
A dictionary where both the key and the value are strings. Note that the key should not contain an equal sign (‘=’), otherwise the resulting string will have a different meaning. However, this is not currently enforced.

The function returns a tuple containing the strings in "key=value" form for each key and value in the *options_dict* dictionary passed in. This is how the plugin API accepts options and settings.

PLUGIN API CHANGELOG (Python)

None yet

LIMITATIONS

Only a maximum number of 8 python I/O plugins can be loaded at once. If */etc/sudo.conf* contains more, those will be rejected with a warning message.

The Event API and the hook function API is currently not accessible for Python plugins.

SEE ALSO

sudo.conf(5), sudo_plugin(5), sudoers(5), sudo(8)

AUTHORS

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

Todd C. Miller

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

BUGS

Python plugin support is currently considered experimental.

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

SECURITY CONSIDERATIONS

All Python plugin handling is implemented inside the python_plugin.so dynamic plugin. Therefore, if no Python plugin is registered in sudo.conf(5) or the *sudoers* file, **sudo** will not load the Python interpreter or the Python libraries.

By default, a Python plugin can only import Python modules which are owned by *root* and are only writable by the owner. The reason for this is to prevent a file getting imported accidentally which is modifiable by a non-root user. As **sudo** plugins run as *root*, accidentally importing such file would make it possible for any user (having write access) to execute any code with administrative rights.

However, during development of a plugin this might not be very convenient. The sudo.conf(5) developer_mode option can be used to disable it. For example:

```
Set developer_mode true
```

Please note that this creates a security risk, so it is not recommended on critical systems such as a desktop machine for daily use, but is intended to be used in development environments (VM, container, etc). Before enabling developer mode, ensure you understand the implications.

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