Client

SSH client & key policies

class **paramiko.client.AutoAddPolicy**

Policy for automatically adding the hostname and new host key to the local **HostKeys** object, and saving it. This is used by **SSHClient**.

class **paramiko.client.MissingHostKeyPolicy**

Interface for defining the policy that **SSHClient** should use when the SSH server's hostname is not in either the system host keys or the application's keys. Pre-made classes implement policies for automatically adding the key to the application's **HostKeys** object (**AutoAddPolicy**), and for automatically rejecting the key (**RejectPolicy**).

This function may be used to ask the user to verify the key, for example.

__weakref__ ¶

list of weak references to the object (if defined)

missing_host_key(*client*, *hostname*, *key*)

Called when an **SSHClient** receives a server key for a server that isn't in either the system or local **HostKeys** object. To accept the key, simply return. To reject, raised an exception (which will be passed to the calling application).

class **paramiko.client.RejectPolicy**

Policy for automatically rejecting the unknown hostname & key. This is used by **SSHClient**.

class **paramiko.client.SSHClient**

A high-level representation of a session with an SSH server. This class wraps **Transport**, **Channel**, and **SFTPClient** to take care of most aspects of authenticating and opening channels. A typical use case is:

```
client = SSHClient()
client.load_system_host_keys()
client.connect('ssh.example.com')
stdin, stdout, stderr = client.exec_command('ls -l')
```

You may pass in explicit overrides for authentication and server host key checking. The default mechanism is to try to use local key files or an SSH agent (if one is running).

Instances of this class may be used as context managers.

New in version 1.6.

__init__()

Create a new SSHClient.

close()

Close this SSHClient and its underlying **Transport**.

> Warning:
> Failure to do this may, in some situations, cause your Python interpreter to hang at shutdown (often due to race conditions). It's good practice to **close** your client objects anytime you're done using them, instead of relying on garbage collection.

connect(*hostname*, *port=22*, *username=None*, *password=None*, *pkey=None*, *key_filename=None*, *timeout=None*, *allow_agent=True*, *look_for_keys=True*, *compress=False*, *sock=None*, *gss_auth=False*, *gss_kex=False*, *gss_deleg_creds=True*, *gss_host=None*, *banner_timeout=None*, *auth_timeout=None*, *gss_trust_dns=True*)

Connect to an SSH server and authenticate to it. The server's host key is checked against the system host keys (see **load_system_host_keys**) and any local host keys (**load_host_keys**). If the server's hostname is not found in either set of host keys, the missing host key policy is used (see **set_missing_host_key_policy**). The default policy is to reject the key and raise an **SSHException**.

Authentication is attempted in the following order of priority:

v: latest

- The `pkey` or `key_filename` passed in (if any)
    - `key_filename` may contain OpenSSH public certificate paths as well as regular private-key paths; when files ending in `-cert.pub` are found, they are assumed to match a private key, and both components will be loaded. (The private key itself does *not* need to be listed in `key_filename` for this to occur - *just* the certificate.)
- Any key we can find through an SSH agent
- Any "id_rsa", "id_dsa" or "id_ecdsa" key discoverable in `~/.ssh/`
    - When OpenSSH-style public certificates exist that match an existing such private key (so e.g. one has `id_rsa` and `id_rsa-cert.pub`) the certificate will be loaded alongside the private key and used for authentication.
- Plain username/password auth, if a password was given

If a private key requires a password to unlock it, and a password is passed in, that password will be used to attempt to unlock the key.

Parameters:
- **hostname** (*str*) – the server to connect to
- **port** (*int*) – the server port to connect to
- **username** (*str*) – the username to authenticate as (defaults to the current local username)
- **password** (*str*) – a password to use for authentication or for unlocking a private key
- **pkey** (*PKey*) – an optional private key to use for authentication
- **key_filename** (*str*) – the filename, or list of filenames, of optional private key(s) and/or certs to try for authentication
- **timeout** (*float*) – an optional timeout (in seconds) for the TCP connect
- **allow_agent** (*bool*) – set to False to disable connecting to the SSH agent
- **look_for_keys** (*bool*) – set to False to disable searching for discoverable private key files in `~/.ssh/`
- **compress** (*bool*) – set to True to turn on compression
- **sock** (*socket*) – an open socket or socket-like object (such as a **`Channel`**) to use for communication to the target host
- **gss_auth** (*bool*) – `True` if you want to use GSS-API authentication
- **gss_kex** (*bool*) – Perform GSS-API Key Exchange and user authentication
- **gss_deleg_creds** (*bool*) – Delegate GSS-API client credentials or not
- **gss_host** (*str*) – The targets name in the kerberos database. default: hostname
- **gss_trust_dns** (*bool*) – Indicates whether or not the DNS is trusted to securely canonicalize the name of the host being connected to (default `True`).
- **banner_timeout** (*float*) – an optional timeout (in seconds) to wait for the SSH banner to be presented.
- **auth_timeout** (*float*) – an optional timeout (in seconds) to wait for an authentication response.

Raises: **`BadHostKeyException`** – if the server's host key could not be verified

Raises: **`AuthenticationException`** – if authentication failed

Raises: **`SSHException`** – if there was any other error connecting or establishing an SSH session

Raises: socket.error – if a socket error occurred while connecting

Changed in version 1.15: Added the `banner_timeout`, `gss_auth`, `gss_kex`, `gss_deleg_creds` and `gss_host` arguments.

Changed in version 2.3: Added the `gss_trust_dns` argument.

`exec_command`(*command*, *bufsize=-1*, *timeout=None*, *get_pty=False*, *environment=None*)

Execute a command on the SSH server. A new **`Channel`** is opened and the requested command is executed. The command's input and output streams are returned as Python `file`-like objects representing stdin, stdout, and stderr.

Parameters:
- **command** (*str*) – the command to execute
- **bufsize** (*int*) – interpreted the same way as by the built-in `file()` function in Python
- **timeout** (*int*) – set command's channel timeout. See **`Channel.settimeout`**
- **environment** (*dict*) – a dict of shell environment variables, to be merged into the default environment that the remote command executes within.

> Warning:
> Servers may silently reject some environment variables; see the warning in

`Channel.set_environment_variable` for details.

Returns: the stdin, stdout, and stderr of the executing command, as a 3-tuple

Raises: `SSHException` – if the server fails to execute the command

`get_host_keys`()

Get the local `HostKeys` object. This can be used to examine the local host keys or change them.

Returns: the local host keys as a `HostKeys` object.

`get_transport`()

Return the underlying `Transport` object for this SSH connection. This can be used to perform lower-level tasks, like opening specific kinds of channels.

Returns: the `Transport` for this connection

`invoke_shell`(*term='vt100'*, *width=80*, *height=24*, *width_pixels=0*, *height_pixels=0*, *environment=None*)

Start an interactive shell session on the SSH server. A new `Channel` is opened and connected to a pseudo-terminal using the requested terminal type and size.

Parameters:
- **term** (*str*) – the terminal type to emulate (for example, `"vt100"`)
- **width** (*int*) – the width (in characters) of the terminal window
- **height** (*int*) – the height (in characters) of the terminal window
- **width_pixels** (*int*) – the width (in pixels) of the terminal window
- **height_pixels** (*int*) – the height (in pixels) of the terminal window
- **environment** (*dict*) – the command's environment

Returns: a new `Channel` connected to the remote shell

Raises: `SSHException` – if the server fails to invoke a shell

`load_host_keys`(*filename*)

Load host keys from a local host-key file. Host keys read with this method will be checked after keys loaded via `load_system_host_keys`, but will be saved back by `save_host_keys` (so they can be modified). The missing host key policy `AutoAddPolicy` adds keys to this set and saves them, when connecting to a previously-unknown server.

This method can be called multiple times. Each new set of host keys will be merged with the existing set (new replacing old if there are conflicts). When automatically saving, the last hostname is used.

Parameters: **filename** (*str*) – the filename to read

Raises: `IOError` – if the filename could not be read

`load_system_host_keys`(*filename=None*)

Load host keys from a system (read-only) file. Host keys read with this method will not be saved back by `save_host_keys`.

This method can be called multiple times. Each new set of host keys will be merged with the existing set (new replacing old if there are conflicts).

If `filename` is left as `None`, an attempt will be made to read keys from the user's local "known hosts" file, as used by OpenSSH, and no exception will be raised if the file can't be read. This is probably only useful on posix.

Parameters: **filename** (*str*) – the filename to read, or `None`

Raises: `IOError` – if a filename was provided and the file could not be read

`open_sftp`()

Open an SFTP session on the SSH server.

Returns: a new `SFTPClient` session object

`save_host_keys`(*filename*)

Save the host keys back to a file. Only the host keys loaded with `load_host_keys` (plus any added directly) will be saved – not any host keys loaded with `load_system_host_keys`.

Parameters: **filename** (*str*) – the filename to save to

Raises: `IOError` – if the file could not be written

`set_log_channel`(*name*)

Set the channel for logging. The default is `"paramiko.transport"` but it can be set to anything you want.

Parameters: **name** (*str*) – new channel name for logging

`set_missing_host_key_policy`(*policy*)

Set policy to use when connecting to servers without a known host key.

Specifically:

- A **policy** is a "policy class" (or instance thereof), namely some subclass of **`MissingHostKeyPolicy`** such as **`RejectPolicy`** (the default), **`AutoAddPolicy`**, **`WarningPolicy`**, or a user-created subclass.
- A host key is **known** when it appears in the client object's cached host keys structures (those manipulated by **`load_system_host_keys`** and/or **`load_host_keys`**).

Parameters: **policy** (*MissingHostKeyPolicy*) – the policy to use when receiving a host key from a previously-unknown server

class `paramiko.client.`**`WarningPolicy`**

Policy for logging a Python-style warning for an unknown host key, but accepting it. This is used by **`SSHClient`**.