

sshproxy-0.5 documentation

Index

- 1 Introduction
- 2 Acknowledgement
- 3 Getting the sources
- 4 Installation
- 5 Configuration
- 6 Running the proxy
- 7 Administering the proxy
- 8 Using the proxy
- 9 ACL: Access Control List
 - 9.1 ACL rules
 - 9.2 ACL tags
 - 9.3 ACL reference
 - 9.3.1 ACL syntax
 - 9.3.2 ACL rules
 - 9.3.3 ACL tags
 - 9.3.3.1 namespace client
 - 9.3.3.2 namespace site
 - 9.3.3.3 namespace proxy
 - 9.3.4 More examples
- 10 Backends
 - 10.1 The file database backend
 - 10.1.1 The **client.db** file database format
 - 10.1.2 The **site.db**/ directory database format
 - 10.1.3 The **acl.db** file database format
 - 10.2 The MySQL database backend
- 11 Extra plugins
 - 11.1 Plugin acl_funcs
 - 11.1.1 len()
 - 11.1.2 substr()
 - 11.1.3 diskspace()
 - 11.2 Plugin console_extra
 - 11.2.1 open
 - 11.2.2 run
- 12 Comments

1 Introduction

sshproxy is an ACL-enabled authentication proxy based on the SSH2 protocol. In the version 0.5, it is able to handle shell sessions, remote command execution and scp file transfer. Port forwarding and sftp will be added in a future release.

Authentication can be done with passwords and/or public keys, both on client side and remote host side.

Sensible data (passwords and keys) is encrypted and all data can be stored in three different databases: client, acl and site. Of course, a common database can be used too, but for documentation purpose we will talk about three different databases.

Two database engines are included in v0.5, a file based backend, and a MySQL one.

You can mix database types in the application, ie. use the MySQL for client and site databases, and use a plain file for the ACL database.

2 Acknowledgement

Since sshproxy version 0.5 is still in development, information in this documentation may not be accurate, and is subject to change without notice. The best documentation as of yet is the source code.

3 Getting the sources

Until 0.5 final release, you can get the sources with git or cogito:

```
$ cg-clone http://penguin.fr/git/sshproxy.git
$ cd sshproxy
```

4 Installation

Installation is as easy as typing:

```
# python setup.py install
```

5 Configuration

To configure **sshproxy** before a first run, you need to use the script *sshproxy-setup*:

```
$ sshproxy-setup
```

This will display a menu allowinging you to tweak the configuration options:

```
Configure sshproxy
=====

Global options
1. IP address or interface [any]
2. Port [2242]
3. Auto-add public key [no]
4. Public key id string [sshproxy@penguin.fr]
5. Cipher engine [blowfish]
6. Blowfish secret passphrase [*****]

Choose backends
7. ACL database backend type [file_db]
8. Client database backend type [file_db]
9. Site backend type [file_db]

Configure backends
10. File Backend

Plugin options
11. Select plugins
```

```
0. Quit
```

```
Please make a choice:
```

Default values should be OK for a first try, using the **file_db** backend for both three databases.

Note: If you want to use the **mysql_db** backend, you will need to create the database tables with the script *sshproxy/misc/mysql_db.sql*.

When you quit the menu, **sshproxy-setup** will create the necessary configuration directories and files in *\$HOME/.sshproxy*.

The main configuration file is *sshproxy.ini*. Here is a sample *sshproxi.ini*:

```
[sshproxy]
bindip =
port = 2242
plugin_dir = /usr/lib/sshproxy
logger_conf = /usr/share/sshproxy/logger.conf
log_dir = @log
pkey_id = sshproxy@penguin.fr
auto_add_key = no
client_db = file_db
acl_db = file_db
site_db = file_db
plugin_list = file_db
cipher_type = blowfish

[blowfish]
secret = This should be a valid passphrase

[client_db.file]
file = @client.db

[acl_db.file]
file = @acl.db

[site_db.file]
db_path = @site.db
```

In this file, there are several sections. The first one, **[sshproxy]** is the main section, while all the other sections are provided by plugins or different optional features of the program.

Note: the @ sign for a path is replaced by the absolute path to the configuration directory, ie. */home/sshproxy/.sshproxy/* or */var/lib/sshproxy/.sshproxy/* on Gentoo default installation.

The next step is to create an admin client user that will be able to create clients and sites, and edit ACL rules:

```
$ sshproxy-setup --add-admin admin
Enter password:
```

```
Enter password (verify):
Admin admin added.
$
```

Now, we're ready to start the proxy daemon **sshproxyd**.

6 Running the proxy

Start sshproxy by executing the following command:

```
$ sshproxyd
Server ready, clients may login now ...
```

You can make it start as a daemon with the `--daemon` option:

```
$ sshproxyd --daemon
$
```

7 Administering the proxy

We now have the proxy running, and an admin user has been added to the client database. We can now connect to the proxy with this admin user to add other client users and sites.

To connect to the proxy as the admin user, run the following command:

```
$ USER=admin pssh
sshproxy>
```

Or, with the normal ssh client:

```
$ ssh -tp 2242 admin@localhost
sshproxy>
```

Type help to see the list of available commands:

```
sshproxy> help
Documented commands (type help <topic>):
=====
add_client  del_client  list_aclrules  message      shutdown
watch
add_site    del_site    list_clients   nb_con       tag_client
del_aclrule kill        list_sites     set_aclrule  tag_site

sshproxy>
```

The commands that interest us for now are the `add_*`, `list_*` and `tag_*` sets of commands.

Let's start by listing clients:

```
sshproxy> list_clients
admin

Total: 1
sshproxy>
```

We now add another client:

```
sshproxy> add_client sasha
Client sasha added
sshproxy>
```

Set the password for sasha:

```
sshproxy> tag_client sasha password=sashapassword
username = "sasha"
password = "64fba6d2f1c7082f2e682b91492cb777804ab03e"
sshproxy>
```

You can set several tags at once:

```
sshproxy> tag_client sasha groups="web mail" ip_src="192.168.1.7"
username = "sasha"
password = "64fba6d2f1c7082f2e682b91492cb777804ab03e"
ip_src = "192.168.1.7"
groups = "web mail"
sshproxy>
```

Read the section ACL tags for details about tags.

Let's add a site, so that sasha will be able to log into its favorite server:

```
sshproxy> add_site bigtruck
Site bigtruck added
sshproxy>
```

Set the tags of the bigtruck server:

```
sshproxy> tag_site bigtruck ip_address=1.2.3.4 port=22
ip_address = "1.2.3.4"
port = "22"
name = "bigtruck"
sshproxy>
```

Let's add a login to the bigtruck server:

```
sshproxy> add_site nine@bigtruck
Site nine@bigtruck added
sshproxy> tag_site nine@bigtruck password="ninepasswd"
login = "nine"
password = "$blowfish$sFFqBatXr057vnj2g=="
name = "bigtruck"
```

```
sshproxy>
```

Now we need to create a special relation between the client *sasha* and the site login *nine@bigtruck*, so that sasha will be able to log into bigtruck as user nine:

```
sshproxy> set_aclrule sasha client.username == "sasha"
sshproxy> set_aclrule authenticate acl(sasha)
sshproxy> set_aclrule authorize acl(sasha)
sshproxy> set_aclrule shell_session site.name in
split(client.sites)
sshproxy> tag_client sasha sites="bigtruck"
```

Read the section ACL rules for details about what we've done here.

8 Using the proxy

Once the server has been started, you can connect to the proxy either with the standard *ssh* client, or with the *pssh* wrapper.

To connect to the proxy with the standard *ssh* client, enter the following command:

```
$ ssh -tp 2242 user@sshproxy -- --help
Password:
```

Enter the password, hit enter, and, if you have set the ACL rules in a file like the one described in The *acl.db* file database format, you may see something like:

```
$ ssh -tp 2242 user@sshproxy -- --help
Password:
usage:
    pssh [options]
    pssh [user@site [cmd]]

options:
  -h, --help          show this help message and exit
  -l, --list-sites    list allowed sites
  --get-pssh          display pssh client script.
  --get-pscp         display pscp client script.
Connection to sshproxy closed.
$
```

You can see it does mention the usage of *pssh*. In fact *pssh* is the wrapper to connect to the *sshproxy* server. So you can do the same as with the standard *ssh* client in a much simpler manner:

```
$ export SSHPROXY_HOST='127.0.0.1'
$ export SSHPROXY_PORT='2242'
$ export SSHPROXY_USER='user'
$ pssh --help
Password:
usage:
    pssh [options]
    pssh [user@site [cmd]]
```

```
options:
  -h, --help          show this help message and exit
  -l, --list-sites    list allowed sites
  --get-pssh          display pssh client script.
  --get-pscp          display pscp client script.
Connection to sshproxy closed.
$
```

Note that you don't have to type the *export* lines every time in the same console. You can put them in your *~/.bashrc* or *~/.bash_profile* if you often use the *pssh* and *pscp* commands

Note also that the different options listed by *--help* are subject to restrictions due to ACL rules, so you may or may not see the same, depending on your ACL settings, and the loaded plugins.

A file transfer is possible too, with the same approach. You can either use the standard *scp* command, or use the *pscp* wrapper (make sure you set the ACL rule *scp_transfer* to *True* in the ACL database):

```
$ scp -oPort=2242 user@sshproxy some_local_file
user@proxy:root@remote:some_path/
```

Is the same as:

```
$ pscp some_local_file root@remote:some_path/
```

And to download a file:

```
$ scp -oPort=2242 user@sshproxy
user@proxy:root@remote:some_path/some_remote_file .
```

The same with *pscp*:

```
$ pscp root@remote:some_path/some_remote_file .
```

9 ACL: Access Control List

9.1 ACL rules

ACL rules are defined by a simple expression syntax.

Let's learn by looking at examples, and we'll see the ACL reference later.

There are three namespaces in ACLs: *client*, *site*, *proxy*. In each namespace, there are static ACL tags, and custom tags defined in the client and site databases.

There are different phases in the process of connecting a client to a remote site. The first one is called **authenticate**.

This rule only allows clients from IP address 192.168.1.7:

```
authenticate:
  client.ip_addr = "192.168.1.7"
```

This rule allows client who's username is 'foo':

```
authenticate:
  client.username = "foo"
```

The **authenticate** ACL allows clients to connect to the proxy, but nothing more. By itself it is not very usefull.

The second phase is **authorize**. This rule allows client 'foo' to connect to site 'mailhost':

```
authorize:
  client.username = "foo" and site.name = "mailhost"
```

The **authorize** ACL does not allow a client to do anything yet.

The next phase is to allow the client to be able to open a session. Let's start with a shell session:

```
shell_session:
  proxy.time >= "08:00" and proxy.time <= "18:00"
```

The last rule allows shell sessions during work hours only.

There exist other phases for other types of connection:

- **remote_command** is to allow remote command execution.
- **scp_transfer** is to allow scp transfer, in whichever direction.
- **scp_upload** is to allow only file uploads.
- **scp_download** is to allow only file downloads.
- **console_session** is to allow clients to connect to the proxy integrated command console.
- **admin** is to declare which user has administrator rights.
- **opt_list_sites** is to allow a client to use the --list-sites command line on pssh.
- **opt_get_pkey** is to allow a client to use the --get-pkey command line on pssh.

Other ACLs can be added by plugins, so check the plugin documentation for details.

Custom ACL rules can also be created by the administrator to share an expression between several ACL rules.

Consider this example:

```
authenticate:
  # anyone is allowed to authenticate on work hours,
  # but only clients from the LAN are allowed anytime
  client.ip_addr := "192.168.1." or acl(work_hours)

authorize:
```

```
# authorize anyone anytime
True

admin:
# david is admin
client.username = "david"

shell_session:
# shell session is allowed on work hours,
# or anytime for admin
acl(work_hours) or acl(admin)

remote_command:
# only free and uptime are allowed,
# unless the user is admin
proxy.cmdline in split("uptime free") or acl(admin)

work_hours:
# from 09:00 to 19:00 on Monday to Friday
proxy.time >= "09:00" and proxy.time <= "19:00"
and proxy.dow in "12345"
```

In the above example (relatively complex, but pretty realistic), the ACL **work_hours** is used by two other different ACLs.

9.2 ACL tags

ACL tags are simply key-value mappings attached to the client, site and/or proxy objects. These tags can be used to store contextual information for ACL checking.

ACL tags provide an elegant way to group objects sharing the same qualities - is a *group* attribute can gather users together -- or to set an attribute on objects to allow connection under certain circumstances only - ie. a *dayofweek* attribute could allow connection on monday only.

ACL tags are grouped together by namespaces corresponding to database objects: **client**, **site** and **proxy**.

The **proxy** namespace holds application-related tags, like the command line when the client execute a remote command, and virtual tags, like the current time and date.

The complete list of tags is described in the section ACL reference.

9.3 ACL reference

9.3.1 ACL syntax

The ACL system is very powerful, and very easy to adapt to your needs, once you've understood how it works.

At the heart of this system, are ACL rules. These are simple expressions evaluated to True or False, to allow or deny access.

The syntax is composed of a few keywords, several operators, and some functions.

The keywords are:

- **and**
evaluate the left and right part, and returns *right* if *left* is True, or False otherwise.
- **or**
evaluate the left and right parts, and returns *right* if *left* is False, or *left* otherwise.
- **not**
evaluate the right part, and returns True if *right* is False, or False otherwise.
- **in**
evaluate the left and right parts, and returns True if *left* is part of *right*, or False otherwise.

The operators are:

- **== < <= > >= !=**
these comparison operators have the same behavior as in python and a lot of well-known languages.
- **:=**
this operator compares the *left* part with the start or the *right* part.
- **=:**
this operator compares the *left* part with the end of the *right* part.
- **()**
parenthesis can group expressions to change operators precedence.

The functions are:

- **acl(*expression*)**
this function evaluate the ACL rule *expression*. *expression* can be a string or the name of another ACL rule.
WARNING: no check is done for recursivity, so you shall not give *expression* the name of the current ACL rule.
- **split(*expression*)**
this function split the string *expression* to return a list. *expression* will be splitted by ' ', 't', 'r' and 'n' (like in python).

The constants are:

- **True False**
boolean values, comparison is the same as in python: "", 0, None return **False**, everything else returns **True**. There is one exception, every expression that is not syntactically or grammatically correct returns **False**.

9.3.2 ACL rules

- **authenticate**
authenticate is the *first stage* rule that allows or denies a client to authenticate on the proxy.
- **authorize**
authorize is a *second stage* rule that allows or denies a client to get the remote site data.

- **admin**
admin is a *second stage* rule to give a client administrative rights on the proxy.
- **remote_exec**
remote_exec is a *third stage* rule to give the client the ability to execute remote commands on the site.
- **shell_session**
shell_session is a *third stage* rule to give the client the ability to open a shell session on the site.
- **scp_transfer**
scp_transfer is a *third stage* rule to give the client the ability to transfer files from or to a remote site.
- **scp_upload**
scp_upload is a *third stage* rule to give the client the ability to transfer files to a remote site.
- **scp_download**
scp_download is a *third stage* rule to give the client the ability to transfer files from a remote site.
- **console_session**
console_session is a *second stage* rule to give the client to open an administrative console on the proxy.
- **opt_list_sites**
this allows display and execution of the command line option *--list-sites*, which may be useful to check which sites a client can connect to.

9.3.3 ACL tags

9.3.3.1 namespace client

- **username**
username is a string containing the username used by the client to connect to the proxy.
- **ip_addr**
ip_addr is a string containing the IP address of the client.

9.3.3.2 namespace site

- **login**
login is a string containing the username used to connect to the remote site.
- **name**
name is the name of the remote site.
- **ip_address**
ip_address is the IP address of the remote site.
- **port**
port is the port on which the ssh server is listening on the remote site.

9.3.3.3 namespace proxy

- **date**
date is a string in '%Y-%m-%d' format: '2006-07-13'
- **time**
time is a string in '%H:%M' format: '08:30'
- **datetime**
datetime is the concatenation of date and time separated with a space '%Y-%m-%d %H:%M': '2006-07-13 08:30'
- **doy**
doy means *day of year* and is a string containing the day number starting from January 1st, padded with zeroes: '085' for March 26th (on a non-bisextile year).
- **week**
week is a string containing the week number padded with zeroes: '05' for week starting on January 30th.
- **dow**
dow means *day of week* and is an integer between 0 (sunday) and 6 (saturday).
- **cmdline**
cmdline is a string containing the command line to be executed when a remote command is to be executed.
- **scp_path**
scp_path is a string containing the path to the file to be copied from or to a site during an scp transfer.
- **scp_dir**
scp_dir is a string equal to 'upload' or 'download', depending on the direction of the scp transfer.
- **scp_args**
scp_args is a string containing the argument used to do the scp transfer. Typically, it will contain '-t' (upload) or '-f' (download) but can also contain '-v' (verbose).

9.3.4 More examples

Allow any client from only one IP address to connect to the proxy:

```
authenticate: client.ip_addr == "172.30.0.1"
```

Allow client "johann" to obtain site information only from the network 192.168.7.0/24:

```
authorize: client.username == "johann" and client.ip_addr := "192.168.7."
```

Allow monitors only to access to the sshproxy console:

```
# tag_client username client.groups = "monitor user"
monitors: "monitor" in split(client.groups)
console_session: acl(monitors)
```

Allow clients to download files from their own IP addresses only:

```
# tag_client username client.hostip = "10.0.0.42"
scp_download: client.ip_addr == client.hostip
```

Allow uploads only in a specific place on each server:

```
# tag_site user@sitename
repository="/var/www/example.com/htdocs/download/"
scp_upload: proxy.scp_path := site.repository
```

Allow everyone to list its allowed sites:

```
opt_list_sites: True
```

Allow guests to remotely execute the command *date* between 8:00 and 18:00:

```
remote_exec:
  "guest" in split(client.groups)
  and proxy.time >= "08:00" and proxy.time < "18:00"
  and proxy.cmdline == "date"
```

10 Backends

10.1 The file database backend

The file database backend is implemented in the form of a plugin located in */usr/lib/sshproxy/file_db*.

It contains handlers for the three different databases types **client**, **ACL** and **site**, witch are represented as different file formats.

10.1.1 The client.db file database format

The **client.db** file format is the same as an .ini file, with each section being a username, and options in that section describes authentication tokens, and optionally ACL tags.

Here is an example minimal client.db file format:

```
[david]
password = user_password
pkey =
```

You can add as many sections as you have users allowed to connect to the proxy. Note that an entry in this file does not necessarily provide access to a remote site.

10.1.2 The site.db/ directory database format

The **site.db/** database format is a directory containing files in the same as an .ini file. Each filename is a site name, the *[DEFAULT]* section in the file correspond to the site data and each additional section is a username, with options in that section describing authentication tokens, and optionally ACL tags.

Here is an example minimal *site.db/sitename* file:

```
[DEFAULT]
ip_address = 192.168.1.5
port = 22

[root]
password = $blowfish$RXX7kQrE13g=
pkey =
priority = 1
```

Here you can see that the password is blowfish-encrypted.

10.1.3 The *acl.db* file database format

The **acl.db** file database is not an .ini file, but it contains ACL rules. Here is a minimal example to let everyone connect to a remote site with a shell session:

```
authenticate:
    True

authorize:
    True

shell_session:
    True
```

Note: With this configuration, there is no restriction for an authenticated user to open a shell session, but non-authenticated users *can not* do anything anyway.

ACLs can be more complex and powerfull, but this is not the purpose of this chapter to describe it in details. See ACL: Access Control List for an exhaustive description.

10.2 The MySQL database backend

The mysql database backend can be chosen by setting the following options in *sshproxy.ini*:

```
[sshproxy]
... keep other values as needed ...
plugin_list = mysql_db
client_db = mysql
acl_db = mysql
site_db = mysql

[client_db.mysql]
db = sshproxy
host = localhost
user = sshproxy
password = sshproxypw
port = 3306

[acl_db.mysql]
db = sshproxy
```

```
host = localhost
user = sshproxy
password = sshproxypw
port = 3306

[site_db.mysql]
db = sshproxy
host = localhost
user = sshproxy
password = sshproxypw
port = 3306
```

11 Extra plugins

11.1 Plugin `acl_funcs`

This plugin defines three ACL functions: `len()`, `substr()` and `diskspace()`.

11.1.1 `len()`

The `len()` function returns the length of a string or a list:

```
my_acl:
    len(split(client.groups)) > 3
```

11.1.2 `substr()`

The `substr()` function returns a substring:

```
my_acl:
    # client can connect every hour from HH:00 to HH:09 only
    substr(3, 4, proxy.time) == "0"
```

11.1.3 `diskspace()`

The `diskspace()` function returns the disk occupation on a mountpoint, in percent multiplied by 10:

```
remote_exec:
    # when /var occupation is over 95%, bob can remove a big file
    client.username == "bob" and diskspace("/var") >= 950
    and proxy.cmdline == "rm -f /var/tmp/big_file"
```

11.2 Plugin `console_extra`

This plugin adds the following new console commands: `open` and `run`.

11.2.1 `open`

The `open` command allows a user to connect to a site in the same way he would do with `pssh`:

```
$ pssh user@site  
  
is equivalent to:  
  
$ pssh  
sshproxy> open user@site
```

This avoid to have to define a lot of sessions in PuTTY for example, or this gives more confidentiality when connecting from an internet-cafe.

11.2.2 run

The *run()* command allows a user to execute a remote command in the same way he would do with *pssh*:

```
$ pssh user@site cat /proc/cpuinfo  
  
is equivalent to:  
  
$ pssh  
sshproxy> run user@site cat /proc/cpuinfo
```

warning:	This command has a bug occuring on some SSH servers, that could freeze your connection.
-----------------	---

To be continued...

12 Comments

Add your comments here.

A useful note: The acl to able to use the console_extra plugin needs is cmd_open, create an acl for cmd_open and you should be able to use the open command.