rssht Documentation

Release 1.0

Jérémie Roquet

Apr 16, 2023

Contents

1 Installation **3**
- 1.1 Getting rssht and its dependencies . . . 3
- 1.2 Setting up SSH . . . 3
- 1.3 Security considerations . . . 4

2 Usage **7**

3 Troubleshooting **9**

4 Software recommendations **11**
- 4.1 Keeping sessions alive . . . 11
- 4.2 Using graphical applications . . . 11

5 Contributing **13**

6 License **15**

7 Acknowledgements **17**

8 Related projects **19**

rssht creates and maintains a reverse SSH tunnel with optional SSH over HTTP.

In other words, it opens a port on a *remote client* (ie. the machine you want to connect from) forwarding to a port on the *local host* (ie. the machine you want to connect to) so that the remote client can connect to the local host even if the latter is not visible from the outside network (because it is behind some proxy, gateway, firewall or some NAT device).

When the connection is lost (because the remote client has been shut down, or because of a network issue), rssht takes care of re-establishing the tunnel.

It is useful when all you have on the machine you want to connect to is a SSH client and no root access. If you are root on the machine you want to connect to, you may prefer using a tool like Nebula.

CHAPTER 1

Installation

1.1 Getting rssht and its dependencies

rssht is not yet packaged, but the latest version is available for download on Github.

If you plan to use the optional SSH over HTTP feature, you will also need to install httptunnel on both the local host *and* the client host.

To install rssht, use the following on the local host:

```
sudo wget 'https://raw.githubusercontent.com/Arkanosis/rssht/master/rssht' -O /usr/
↪bin/rssht
sudo chmod a+x /usr/bin/rssht
```

If you want to use SSH over HTTP, install httptunnel on both the local host *and* the client host.

On Debian-based systems, use:

```
sudo apt install httptunnel
```

… and on Arch-based systems, use:

```
sudo pacman -S httptunnel
```

Note that you can of course install httptunnel and rssht anywhere, as long as the binaries are in your `$PATH`.

1.2 Setting up SSH

Using rssht may involve as many as four different users A, B, C and D:

- user A connecting from the *local host* to the *remote client* as user B using `rssht` to establish the reverse SSH tunnel;
- user C connecting from the *remote client* to the *local host* as user D using `ssh` to establish the SSH connexion.

A and D on the *local host* may be the same user — and will be, in most cases, and B and C on the *remote client* may be the same user too — but this is not recommended, for security reasons (see *Security considerations* below).

Obviously, user C on the *remote client* must be able to connect to the *local host* as user D, as it would have to in a normal SSH connection. Any authentification method would work.

Maybe less obviously, user A on the *local host* must be able to connect to the *remote client* as user B. Since it will connect unattendedly, the authentification should be passwordless and passphraseless. It is recommended to use public key authentification with a passphraseless private key.

To create such key, use the following command:

```
ssh-keygen -t rsa -b 4096 -N '' -f ~/.ssh/rssht_id_rsa
```

Don't use that key for anything but `rssht` itself since the lack of passphrase is a serious security issue for most applications (not for rssht if you follow the recommendations in the *Security considerations* section below).

Once you have a passphraseless private key for the user A, you have to add the associated public key to the list of authorized keys to log in as user B on the remote client. To do so, copy this public key in the user B's `~/.ssh/authorized_keys` on the remote client.

To avoid multiple SSH connections between the same hosts, it is recommended to use the `ControlMaster` feature of the OpenSSH client, which enables multiplexing. To enable it, put the following at the end of the user C's `~/.ssh/config` on the remote client:

```
Host *
ControlMaster auto
ControlPath ~/.ssh/master-%r@%h:%p
```

You may want to use the `ControlPersist` feature as well to stay connected even after closing the terminal. To enable it, add it below the lines you have already added to enable multiplexing:

```
Host *
ControlMaster auto
ControlPath ~/.ssh/master-%r@%h:%p
ControlPersist yes
```

1.3 Security considerations

Using a dedicated user on the client host (user B in the *Setting up SSH* section), with no right appart from being able to connect via SSH is recommended. This prevents the local host from doing anything nasty on the client host. This is particulary important if the local host is outside of your control (eg. a corporate host), and even more important as it will connect using a passphraseless private key.

Use the following on the client host to create a user named "rssht-user" who can connect with SSH:

```
sudo useradd rssht-user -m -s /bin/false # Create the user with home directory and␣
↪forbid anything other than SSH
sudo mkdir ~rssht-user/.ssh # Create the SSH configuration directory
sudo cat rssht_id_rsa.pub >> ~rssht-user/.ssh/authorized_keys # Allow the local host␣
↪to connect on the client host as rssht-user; the rssht_id_rsa.pub file must have␣
↪been copied from the local host
sudo chown -R rssht-user:rssht-user ~rssht-user/.ssh # Restore correct ownership
sudo sed -i 's/AllowUsers .*/& rssht-user/' /etc/ssh/sshd_config # Allow rssht-user␣
↪to connect via SSH
```

Restart the SSH daemon for the last change to be taken into account.

On Debian-based systems, use:

```
sudo service ssh restart
```

. . . and on Arch-based systems, use:

```
sudo servicectl restart sshd.service
```

CHAPTER 2

Usage

`rssht` must be run on the *local host* (ie. the machine you want to connect to). It is used as follow:

```
rssht [user@]host[:port] [-f port] [-t port] [-k key] [-n time] [-p␣
↪[user@]proxy[:port]] [--http] [-d]
```

The following options are supported:

- **host**: client hostname
- **user**: username on client host (should have limited rights) [default: same as local username]
- **port**: open port on client host [default: 22 for SSH, 80 for HTTP]
- **-f port**: free port on client host which will redirect to the local host [default: 22000]
- **-t port**: open SSH port on local host to forward on the client host [default: 22]
- **-k key**: the SSH private key to use exclusively on the local host [default: ~/.ssh/rssht_id_rsa]
- **-n time**: restore connection *time* seconds after failure [default: 5]
- **-p [user@]proxy[:port]**: use the proxy to connect to the client [default username: same as local username; default port: 22]
- **–http**: use SSH over HTTP instead of plain ssh. The client port must forward HTTP traffic to an open SSH port (using httptunnel's `hts`)
- **-d**: run as daemon

Example:

```
rssht rssht-user@httptunnel.example.com:80 -f 12345 -t 22 --http -d
```

If you want to use SSH over HTTP, you also need to have `hts` running on the *client host* (ie. the machine you want to connect from):

```
hts -F localhost:22 80
```

Then, you can use ssh on the *client host* to connect to the *local host* as follow:

```
ssh localhost -p 12345
```

If the connection is lost, rssht will restore the tunnel after a few seconds, so you can connect again.

If you are using SSH over HTTP and for some reason `hts` is hanging after losing the connection (it happens), kill it, start it again and wait for rssht to restore the tunnel.

If you are looking for a persistent reverse SSH tunnel, then you probably want it to be restored even if the local host is rebooted. The recommended approach is to start rssht using cron.

```
crontab -e
```

This command opens a list of tasks to be run by the cron service. Add a line as follow to have rssht started when the local host is rebooted:

```
@reboot /usr/bin/rssht rssht-user@httptunnel.example.com:80 -f 12345 -t 22 --http > /
↪dev/null 2>&1
```

If you prefer using a systemd service rather than cron, you can download the example service:

```
wget 'https://raw.githubusercontent.com/Arkanosis/rssht/master/systemd/rssht.service'␣
↪-O ~/.config/systemd/user/rssht.service
```

... edit it with you own settings, and then enable it:

```
systemctl --user enable --now rssht
```

Be aware that a systemd user service is only started after the first login of a user and is stopped when the user session ends, unless lingering is enabled for that user (in which case, the user service is started after boot and is only stopped at shutdown). To enable lingering for your user, use:

```
sudo loginctl enable-linger $USER
```

Note that for rssht to restore the tunnel, the remote client's public key must be present in the local host's (ie. the user A's) `~/.ssh/known_hosts`.

Just run rssht manually as user A on the first time: this will add the public key to the `~/.ssh/known_hosts` after confirmation and subsequent connexions won't ask for confirmation again.

CHAPTER 3

Troubleshooting

There is unfortunately a lot of ways for the reverse tunnel not to work.

The place to start troubleshooting is the authentification log on the *remote client*.

On Debian-based systems, use:

```
sudo tail -f /var/log/auth.log
```

(note that the use of `sudo` is not necessary on Ubuntu), and on Arch-based systems, use:

```
sudo journald -fu sshd
```

Login attempts from the *local host* will be logged there and it is often possible to understand what is going wrong just by looking at this file.

If nothing is being written there, even after the specified rssht delay (`-n` flag), it probably means that rssht is not even able to access the SSH port on the remote client. Double check the open port on the client host and, if you are using SSH over HTTP (`--http` flag), make sure that `hts` is running on the remote client, and double check its input and output ports as well (the input port must match the open port specified when running `rssht` and the ouput port must match an open port fort a running `sshd` on the client host).

You can use telnet from the local host to check if the remote client is reachable:

```
telnet httptunnel.example.com 80
```

If you cannot get a connection, then there might be some NAT device (such as a router) hiding the remote client from the outside network. If so, the NAT device must be configured to route the port used by rssht to the actual client host.

If it hangs, it can be because the connection has been lost and `hts` is hanging on the remote client. Kill it and restart it, then wait again for rssht's connection delay.

If it answers, but not with a OpenSSH greeting message, it is probably because either `sshd` is not running (in that case, start the ssh service) or running but listening on the wrong port (in that case, adjust the destination port with `hts` or by changing sshd_config and restarting the ssh service).

If it answers with a OpenSSH greeting message, then it should be good.

CHAPTER 4

Software recommendations

4.1 Keeping sessions alive

Since a SSH connection is not particulary reliable, especially if established within a HTTP tunnel to traverse a HTTP proxy, it is highly recommended to use some screen-like software to keep a detached session alive on the local host that can be re-attached later.

The author recommends the use of tmux which handles this task very well and provides a handful of additional features, such as terminal sharing, screen splitting…

4.2 Using graphical applications

SSH is great for terminal applications, but not so much for graphical applications. While it is possible to run some graphical applications on the *client host* and have them use the resources of the *local host* like network (eg. using a SOCKS proxy) or filesystem (eg. using sshfs), sometimes, one may want to run an application entirely on the *local host* and only show its GUI on the *client host*.

The author recommends the use of xpra which handles this task as well as it can be and provides some interesting features such as keeping the application alive if the connection is lost.

CHAPTER 5

Contributing

You can contribute by reporting bugs and feature requests on Github.

Pull requests for code and documentation are welcome too.

CHAPTER 6

License

CHAPTER 7

Acknowledgements

The author would like to thank the following people:

- Anne-Sophie Denommé-Pichon, for her precious feedback and extensive testing;
- Richard Groux, for his tips with SSH's ControlMaster and ControlPersist;
- Xavier Roche, the author of pepette, the script from which the inspiration for rssht comes from.

CHAPTER 8

Related projects

The following projects are related: OpenSSH, autossh, Corkscrew, httptunnel.

The current version of rssht is heavily based on OpenSSH and relies on httptunnel for the optional SSH over HTTP.