Setting up SSH

This guide was taken from homebrewserver.club and only lightly adapted to fit better fit the content of this publication.

2.1 Introduction

Some of the essential things that separate a server from other computers is that 1. they are usually not where you are and 2. that often come without screen and keyboard.

In order to use a server, you need to connect to it over the network using the command-line interface or shell. This is usually done with a program called SSH which stands for Secure Shell.

One of the more important and foundational skills needed for experimenting and maintaining servers is understanding, using and troubleshooting SSH.

Setting up and using SSH can be a challenge at first. There are many moving parts to consider. Working with SSH means knowing something about user accounts, permissions, public key cryptography, protocols, clients, servers and agents. And yet despite so much to consider, SSH is praised as something easy to use and quite often presupposed knowledge between peers.

With this in mind, there is a need to demystify SSH. In this guide we aim to show how to setup an SSH client and server as well as discuss common issues and approaches for day to day use and troubleshooting.

2.2 Prerequisites

The SSH ecosystem is very well established. It is available on all modern GNU/Linux distributions, MacOS and Windows. The commands shown in this guide were run on a Debian Stretch distribution but the actual tool names should be the same on other distributions.

2.3 SSH and OpenSSH

The term “SSH” can refer to a number of different but related things.

SSH stands for the Secure Shell which is a protocol for describing how to provide a secure channel of communication from a client to a server. However, it more typically refers to the OpenSSH suite of tools which are the programs you will install and use when dealing with SSH.

The OpenSSH tools are an implementation of the SSH protocol. This means that these tools follow the behaviour described in the documents of the protocol.

2.4 Installing the SSH server and client

It is important to understand the client/server architecture of SSH. If you are remotely connecting to your server from your laptop, then your laptop is the client and the server is the ...server.

There are two packages which contain all the tools that the OpenSSH tool suite provides. The openssh-server and openssh-client packages.

To install the SSH server on your **server**, run:

```
dietpi-software install openssh-server
```

And on your client, run:

```
sudo apt install -y openssh-client
```

2.5 Creating a user account on the server

In order for your client to connect to the server, a user account must be created on the server. This can be done with the following command (where `homebrewer` is the new user account):

```
sudo useradd --create-home --shell /bin/bash homebrewer
```

You should then set a password for this new user:

```
sudo passwd homebrewer
```

This password will be used in the following step.

2.6 Logging in for the first time

You can now SSH into the server from your client. Assuming that you have a DNS entry which points to your home server (`myhomebrewserver.com`) then you can log in with:

```
ssh homebrewer@myhomebrewserver.com
```

On first connection to a new server, you will be shown a fingerprint and asked if you would like to continue connecting. For now, choose to continue without validating the fingerprint.

You should then be asked for the password that you entered when creating the account. If you have any issues at this point, please see the Troubleshooting SSH section.

2.7 Passwords, keys and security

As we have seen so far, connecting to an SSH server using password authorisation is relatively simple. However, password authorisation is typically recommended against due to security considerations.

Security is relative and you may not be concerned with defending against a brute-force attack. However, since other methods of authorisation are so commonly used and often the source of of problems when dealing with SSH connectivity, we will also cover this topic also.

The alternative to password authorisation is public key cryptography. The idea is to generate two keys, one public and the other private. The public key part can be shared publicly and will be registered with the SSH server. The private key part cannot be shared with anyone else and will be used on the client side for authentication.

For those not familiar with public key encryption, a brief and pragmatic introduction is provided by this ssd.eff.org guide.

2.8 Validating server host keys

When logging into your server you were asked to validate a key fingerprint:

```
The authenticity of host 'myhomebrewserver (666.666.666.666)' ed.
ECDSA key fingerprint is SHA256:lUhBmQXjkOL0zyoDNarUIbhd3RXo7X
Are you sure you want to continue connecting (yes/no)?
```

When an SSH server is installed, key pairs are generated on the server. The design of SSH is such that when you make a secure connection you should be sure that where you are connecting to is the server you expect.

On the server, you can validate that this is the fingerprint you expect:

```
sudo ssh-keygen -lf /etc/ssh/ssh_host_ecdsa_key
256 SHA256:lUhBmQXjkOL0zyoDNarUIbhd3RXo7X root@myhomebrewserver (ECDSA)
```

These values should match the ones you were first shown. Where the ECDSA in the first message corresponds to the key file name in `/etc/ssh/`. The SSH server will generate key pairs for each algorithm it supports.

As you have accepted your server host key fingerprint, the public key of the server will be placed in your `$HOME/.ssh/known_hosts` file on your client where it will be remembered for future connections.

If the host key ever changes, you will see something like:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

```
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
...
```

This simply means that that the host key you accepted for this host has changed and you should first figure out why this is the case before making a secure connection.

2.9 Dealing with SSH keys

Generating a public and private key pair

Now that we understand the idea of host keys, we should generate client keys. On your client, run the following:

```
ssh-keygen -t ed25519
```

This uses the currently preferred algorithm for generating keys. The key generation will ask you to enter a passphrase. It is typically recommended use a strong passphrase (see diceware) for the reason that if someone else gets your private key part, they still will still not be able to use it because they also need to know the passphrase.

You should now have two keys in your `$HOME/.ssh` folder. A `id_ed25519` (private key part) and a `id_ed25519.pub` file (public key part).

Your public key part might look something like:

```
cat $HOME/.ssh/id_ed25519
ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIOTWIy+Em89QR/Xs6RUmr+0U0F3GVXY/UA2MXs9MoqdY
myuser@myhost
```

You can then start an `ssh-agent` instance and register the private key part:

```
eval "$(ssh-agent -s)"
ssh-add $HOME/.ssh/id_ed25519
```

The `ssh-agent` should ask for your passphrase. The agent is useful because it will store the passphrase of your key and re-use it the next time you log into the server. This can be particularly useful later on when you need to use multiple SSH keys.

You can confirm that the `ssh-agent` has loaded this key by running:

```
ssh-add -L
ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIOTWIy+Em89QR/Xs6RUmr+0U0F3GVXY/UA2MXs9MoqdY
myuser@myhost
```

And finally, you can then configure your client to use this key. Create a `$HOME/.ssh/config` file and

put something like the following configuration into it (remember to replace with your server details):

```
Host myhomebrewserver
  Hostname myhomebrewserver.com
  User homebrewer
  Port 22
  IdentityFile ~/.ssh/id_ed25519
```

This is useful because it allows you to only type:

```
ssh myhomebrewserver
```

And the correct hostname, user, port and key will be chosen. However, before connecting, you must register your public key part on the server.

Register the public key with the server

On the server, you should register the public key. This is achieved by putting the public key part of the key into the `/home/homebrewer/.ssh/authorized_keys` file.

You can do this like so (replace the public key part with the one you generated):

```
sudo -su homebrewser
cd $HOME && mkdir .ssh
echo "ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIOTWIy+Em89QR/Xs6RUmr+0U0F3GVXY/UA2MXs9MoqdY
myuser@myhost" > .ssh/authorized_keys
```

You must then ensure that your SSH server is accepting keys as a method of authentication. You should edit the `/etc/ssh/sshd_config` file which is the main configuration file for the SSH server.

You need to ensure the following is present:

```
PubkeyAuthentication yes
```

After editing the file, save your changes and then validate the configuration:

```
sudo sshd -t
```

No output should be shown if everything is validating correctly. This is important to do before restarting the SSH server in case you lock yourself out of the server.

You can restart the SSH server with:

```
sudo systemctl restart sshd
```

Now it should be possible to connect from your client with:

```
ssh myhomebrewserver
```

If you have any issues, please see the Troubleshooting SSH section.

Troubleshooting SSH

Despite our best intentions, we are often confronted with a failed login attempt:

```
ssh myhomebrewserver
Permission denied (publickey).
```

This is not the most helpful message. In order to go about solving this issue, we need to focus our detective work on the client/server architecture and try to discover which side is responsible for the problem. Hopefully the following tips can help you in this process.

On the server

Here are some questions to ask yourself:

- Is your public key registered on the server in the `$HOME/.ssh/authorized_keys` folder?
- Are the `$HOME/.ssh` permissions correct? (see here)
- Is the SSH server running?
- Is the `/etc/ssh/sshd_config` correct?
- What does `sudo tail -f /var/log/auth.log` say?

On the client

Here are some questions to ask yourself:

- What does `ssh -vvvvv myhomebrewserver` tell you?
- Are the `$HOME/.ssh` folder permissions correct? (see here)
- Is the SSH server available at the port you expect?
- Is your `$HOME/.ssh/config` correct?
- What is registered with the local `ssh-agent`?

Conclusions

As we can see, SSH is no simple topic! There are many moving parts and a number of topics which require familiarity in order to be able to get remote access to your server.