COURSEWARE

OpenSSH

Contents

Overview

SSH stands for **S**ecure **Sh**ell and is used in nearly every data center and every large enterprise.
OpenSSH is a suite of tools for configuring and making connections using the SSH protocol.
It is a software package which enables secure system administration and file transfers over insecure networks, this could include the Internet for example.
Used to remotely access a server from a client such as:

- Engineer on to a cloud virtual machine
- Automation server on to application servers

SSH Clients

There are many clients that are available, a couple of the most common though are:

- **OpenSSH**: Includes a CLI client
- **PuTTY**: SSH and Telnet client, originally for Microsoft Windows and is used more commonly on Windows

Installation

OpenSSH is most likely already installed on your Linux distribution, you can check with this command:

```
ssh -V
```

If OpenSSH isn't installed then you can used `apt` to install it:

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

Key Pairs

A fundamental component to creating a secure connection using SSH is the keys that are used, this is based on Public Key Infrastructure (PKI).

`ssh-keygen` is one of the OpenSSH utilities that can generate key pairs for you, simply run `ssh-keygen` on its own to generate a default key pair, each step is self-explanatory.

You will be prompted with a few options:

- Where to save the key
  The default location for the key is `~/.ssh/id_rsa`, a corresponding public key (`.pub`) will also be created.
- Passphrase
  Some extra security can be added to your authentication using SSH by making a password, this will require you to enter the passphrase on connections however.
  Typically you will not want to set a passphrase as it requires manual intervention.

The output will look similar to this:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/bob/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/bob/.ssh/id_rsa.
Your public key has been saved in /home/bob/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:nVetDTMXXMgeN/K4I23YImy5UGpFwEohYuPjBicZ/CU bob@test
The key's randomart image is:
+---[RSA 2048]----+
|o+ . .o..    ..oo|
|o+oE.o . .   .++o|
|+oo + . .    .Oo+|
|oo.. .   + . o.O |
| o      S + = o .|
|.      + = = *   |
|      . o o + .  |
|         .       |
|                 |
+----[SHA256]-----+
```

Authorized Keys

To be able to establish a connection between the client and server using SSH, there needs to be a trust relationship between the two to allow the connection. This can be accomplished by installing the *public key* on to the *server*, which is done by appending the *public key* to the `~/.ssh/authorized_keys` file on the remote host.

You can use a tool like `cat` to get your public key:

```
cat ~/.ssh/id_rsa.pub
```

It should look something like this:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCzfjA70YaacZdgYig0urm8S2EIaii9zq2n7F4UyoZPgp+VrBd6
YDcFpMUaEdSDR5DCMY3HVOJxjf2zDfHCd/VqGOjI0+yaniz+EzEAbPfEnlRFRGnzIaCJlHt0L27bLO9e
IEvy0uerjdPUxU6ROM9AdepHbTtMxZUtfbkqTtdsY+JvLAzVz47zj9EJuezJx3vyHKq8XXfhXqG2LeHl
V+u7c7PtodJQW+bYeKka/fz7LyM2dMBCna8XaVkEncJqH3cJVI7FumCXN9pb4IYqh9B4CCqhRAC89OOr
SFjO17xdD3OrGd2DmwNpHgrXtMbNTAvgpsECI8q7PxQ5/co/G4tJ bob@work-laptop
```

A text editor can be then used to enter that public key on to a new line in the `~/.ssh/authorized_keys` file on the server.

> Be sure to copy the *entire* public key, including the `ssh-rsa` at the start and make sure that it is only on one line in the `~/.ssh/authorized_keys` file

Known Hosts

By default SSH will need confirmation when connecting to a host that has been connected to before.
Once confirmed, the host and some other relevant details will be appended to the `~/.ssh/known_hosts` file.
The known hosts file (`~/.ssh/known_hosts`) is used to authenticate with the server, to ensure that it isn't trying to connect to an impersonator.

Entry Data

Each entry (one per line) in the known hosts file contains the following delimited by white space:

- One or more server names or IP addresses, split by commas:

  ```
  example.com,123.123.123.123
  ```

  These may also be hashed, so that if someone was to get your known hosts file they wouldn't know what servers you have been authenticating with:

  ```
  |1|knO3DM+50pfwcbebbAIK0GJQ+6o=|Agr853yfNHp8T2WBAD05hp/+pII=
  ```

- Type of key used, there are a few types this could be, RSA is an example:

  ```
  ssh-rsa
  ```

- Public key from the server, this is generated on the server side automatically and stored in `/etc/ssh`; you do not need to manage this key. In the known hosts file it might look something like this:

  ```
  AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBFSMqzJeV9rUzU4kWitGjeR
  4PWSa29SPqJ1fVkhtj3Hw9xjLVXVYrU9QlYWrOLXBpQ6KWjbjTDTdDkoohFzgbEY=
  ```

Managing the Known Hosts

Using Default Behaviour

Just by trying to SSH on to a machine you will get a prompt similar to this which will ask you to confirm the connection:

```
The authenticity of host 'gitlab.com (35.231.145.151)' can't be established.
ECDSA key fingerprint is SHA256:HbW3g8zUjNSksFbqTiUWPWg2Bq1x8xdGUrliXFzSnUw.
Are you sure you want to continue connecting (yes/no)?
```

Upon accepting this the host is then configured in the known hosts file.

Using `ssh-keyscan` to Add Hosts

Using the default behaviour to add hosts is fine, but it does require a user to respond to the prompt - this isn't ideal for automation purposes.
To avoid this we can explicitly add the host ourselves using `ssh-keyscan`:

```
# ssh-keyscan -H [HOST] >> ~/.ssh/known_hosts
ssh-keyscan -H gitlab.com >> ~/.ssh/known_hosts
```

The `-H` option is telling `ssh-keyscan` to hash the hosts being added:

```
-H      Hash all hostnames and addresses in the output. Hashed names may be used
normally by ssh and sshd, but they do not reveal identify-ing information should
the file's contents be disclosed.
```

Removing Hosts

To remove all hosts you can just delete the known hosts file (`rm ~/.ssh/known_hosts`).

Removing specific hosts can be done using the `ssh-keygen` command:

```
# ssh-keygen -R [HOST]
ssh-keygen -R gitlab.com
```

SSH Commands

Before you can use SSH commands to make connections to other hosts you must have configured a key pair and installed your public key into the authorized keys file on the remote host as discussed earlier.

SSH will try to connect as the current user you are *locally* by default, so a user with the same name and authorized keys must be setup on the remote host beforehand.
The default shell for that user on the remote machine will be used, `/bin/bash` for example.

Terminal Session

Typically when using SSH you are just wanting to have a simple terminal session, provide the name or IP address of the server to do this:

```
# ssh [HOST]
ssh example.com
```

Connect As Another User

You might not be connecting as the same user on the remote host as your are on the local machine.
The user that you would like to connect as can included before the host in the command:

```
# ssh [USER]@[HOST]
ssh bob@example.com
```

Running a Single Command

You may want to only run a single command, this can be useful in automation scenarios:

```
# ssh [HOST] [COMMAND]
ssh example.com echo "I'm being executed on the example.com server"
```

Multiple Commands

Running multiple commands on a host becomes repetitive unless you utilise standard input, this can be from a file or Heredoc for example:

```
# using heredoc
ssh example.com << EOF
echo 1
echo 2
echo 3
EOF

# using a script
# ssh [HOST] < [SCRIPT]
ssh example.com < /tmp/myscript.sh
```

Specify Private Key File

If your private key file isn't stored in the conventional folder `~/.ssh/` then it can be referred with the `-i` option, `ssh` will then attempt to authenticate with that private key:

```
# ssh -i [PRIVATE_KEY_FILE] [HOST]
ssh -i ~/.keys/my_id_rsa example.com
```

Tutorial

This tutorial will take you through connecting 3 VMs using SSH.
It is reccommeded that you use GCP (Google Cloud Platorm) for this tutorial to simplify the infrastructure creation.

Prerequisites

- 3 Ubuntu or Debian Virtual Machines on the same network
  - Create these in the same region & zone on Google Cloud Platform
  - Call them the following:
    - ssh-vm-1
    - ssh-vm-2
    - ssh-vm-3

Virtual Machine 1 Configuration

There will need to be the following configured on the first VM:

- `jenkins` user
- SSH Key Pair

The reason for having a `jenkins` user is to use it for connecting to the other instances, it does't *have* to be called `jenkins`, this is just for the purpose of this tutorial.

Run the following commands on the first virtual machine to configure this:

```
# create the jenkins user
sudo useradd -m -s /bin/bash jenkins
# switch to the jenkins user
sudo su - jenkins
# create a key pair as the jenkins user
ssh-keygen
# just keep pressing enter after this command to accept the defaults
```

We will now need to make a note of the public key for installing on to the other machines, display is with this command:

```
cat ~/.ssh/id_rsa.pub
```

Virtual Machines 2 and 3 Configuration

These machines will also need a `jenkins` user with the public key installed in their `~/.ssh/athorized_keys` file.

Run these commands to make the authorized keys file with the correct permissions:

```
# make the jenkins user
sudo useradd -m -s /bin/bash jenkins
# switch to the jenkins user
sudo su - jenkins
# make the ssh directory and authorized keys file
mkdir -p ~/.ssh
touch ~/.ssh/authorized_keys
# make sure those files cannot be read by anyone else
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
```

You will now need to use a text editor to enter the public key from the first virtual machine in to the `~/.ssh/authorized_keys` file on these machines. Complete this section for both virtual machines 2 & 3.

Attempt SSH Connections

Try connecting from VM 1 to VM 2 & 3.

From VM 1 run:

```
ssh ssh-vm-2
```

You should then be able to connect:

```
jenkins@ssh-vm-1:~$ ssh ssh-vm-2
The authenticity of host 'ssh-vm-2 (10.154.0.39)' can't be established.
ECDSA key fingerprint is SHA256:1haPo8RjkJthieFVrYptHQvzD+YBYlJXoIkHo1sb0pU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ssh-vm-2,10.154.0.39' (ECDSA) to the list of known
hosts.
Linux ssh-vm-2 4.9.0-12-amd64 #1 SMP Debian 4.9.210-1 (2020-01-20) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
jenkins@ssh-vm-2:~$
```

Exit from VM 2 and try to now connect to VM 3.

Exercises

There are no exercises for this module.