SSH PORT FORWARDING

The Problem To Be Solved

Wrap encryption around an otherwise un-encrypted network connection.

Description

SSH port forwarding allows you to map a local TCP port to a TCP port on a remote computer. Any traffic you send to the local port is sent through the SSH connection and then routed to its final destination after it comes out of the other end of the SSH connection.

While it is inside the SSH connection it is encrypted. Once it leaves the SSH connection on the other side it continues the rest of its journey unprotected.

This can be used in two ways:

- To map a port on the computer you are SSHing to a port in your local computer. When used in this way the traffic is encrypted for its entire journey. This is the most secure way to use port forwarding.
- To map a port on third computer to a port on your local computer, with all traffic flowing through the computer you are SSHing to. In this scenario the traffic is encrypted between your computer and the computer you are SSHing to, but not for the remainder of the journey from the computer you are

> SSHing to onto the third computer. I would recommend against using port forwarding in this way if possible.

A crude analogy would be to think of SSH port forwarding as a single-port VPN.

Instructions

A single SSH connection can forward many ports. For each port to be forwarded, an instance of the **-L** flag should be used in the following way:

> **-L[LOCAL_PORT]:[DESTINATION_HOST]:[DESTINATION_PORT]**

Example Use Case – MySQL

Many websites are powered by MySQL databases. It's common when working on a website to need access to the MySQL server powering your site from your local computer. You can do this using the MySQL command line tools, or, using a MySQL GUI. The problem is that the MySQL protocol is insecure (at least by default, it is possible to configure it to use SSL, but that's not straight forward). Your username, password, and all the queries you issue and the server's responses are all sent across the network un-encrypted. Because this is so dangerous, it's common to limit MySQL to using the **localhost** IP address (**127.0.0.1**), or to firewall off access so that only computers within a secured network segment can access the server.

This is no good if you are working from home! SSH port forwarding can save the day, assuming you have SSH access to either the server running MySQL (or another server in the same network as the MySQL server that has been granted access to it.)

Assuming the most secure scenario, MySQL limited to **127.0.0.1** only, and SSH access to the server running MySQL, you would map the port with a command of the form:

> **ssh user@computer -L 3306:127.0.0.1:3306**

As long as that SSH connection is left open, port 3306 on your computer (the standard MySQL port) is mapped to port 3306 on the remote computer's localhost IP. You now instruct your favourite MySQL client to connect to port 3306 on your local computer, and SSH then securely forwards that connection to the remote server for you, allowing you safe and secure access to MySQL.

This is such a common use case that many modern MySQL QUI clients allow you to configure this kind of port forwarding from within the GUI, removing the need to remember the terminal command. An example of a beautiful free MySQL GUI with SSH port forwarding support is Sequel Pro (OS X Only). I use SSH port forwarding with Sequel Pro each and every day!

Source: https://www.bartbusschots.ie/s/2015/04/06/taming-the-terminal-part-32-of-n-ssh-tunnelling/