thnx to : http://manas.tungare.name/blog/ssh-port-forwarding-on-mac-os-x/

SSH Port Forwarding on Mac OS X

30 May 2008

After spending about an hour configuring what should, in theory, be a simple matter, I figured I'd write a blog post that might one day save another soul an hour or so from his or her life. So, for good karma, basically. In the past, I have set up port forwarding on Linux, Mac OS X and Windows, so I was a little worried that it took me about an hour trying to appease the SSH deities (and daemons).

The command itself is just a single line; the devil is in the parameters. I'm splitting the command over several lines and adding line numbers to illustrate the details and separate the parts of the long-ish command for easier explanation. Feel free to type it all on a single line (after removing the line numbers and the line-break markers ("") of course!)

1. ssh 2. -L **local_port**:**service_host**:**service_port** 3. -p **ssh_server_port** 4. -l **ssh_server_username**
5. -N 6. **ssh_server_host**

Parameters

Now for the various parameters used in the command above. Some of them may be omitted if the defaults are used, but I have included all of them in the example above to cover the most general case.

local_port

The port on your local machine that your local program expects to be able to connect to. If this is one of the reserved ports (i.e., under 1023), you will have to run your ssh tunnel command as root (using sudo). Ports above 1024 are freely available for any user to listen on.

service_host

The fully-qualified domain name or the IP address of the server that is hosting the service that you wish to connect to. For example, if this is a web site, it could be google.com or yahoo.com. It does not have to be under your control, nor does it have to be the machine that you're SSHing into. It is just any host on the Internet that you can access from ssh_server_host. Often this is a server you are not allowed to access from your own machine, e.g. a chat server or IRC server. Or you may wish to hide the fact from the administrator of your local network that you are connecting to this server (e.g. when you're out at a coffee shop on a sniffable insecure wireless network, or in a country with laws forbidding access to free information.)

Important: If you're trying to access a service running on the same machine as ssh_server_host, remember to use 127.0.0.1, not localhost. What's the difference, you say? Well, since IPv6 is here to stay, localhost can map to either 127.0.0.1 (IPv4) or ::1/128 (IPv6). If your applications aren't all IPv6-compliant, this can cause some headache. Hopefully, we will all be on IPv6 in the near future, but till then, this is a way to make things work. If you're trying to use IPv6, you need to use **local_port**/**service_host**/**service_port** (slashes instead of colons.)

service_port

The port number on which the desired service is running. Here are some common port numbers:

Service	Port
Web: HTTP	80
Web over SSL: HTTPS	443
Outgoing email: SMTP	25
Incoming email: POP3	110
Incoming email: IMAP	143
VNC	5900
iTunes Music Sharing	3689

ssh_server_host

The machine that you're SSHing into. This is the one that is running sshd, the SSH daemon.

ssh_server_port

The port number on which the SSH daemon is listening on ssh_server_host. This is most likely 22; you should only use a different value if your sysadmin has told you that the SSH server is

running on another port (or if you're a sysadmin yourself and you set up your SSH server to run on a non-standard port for security through obscurity.)

ssh_server_username

The username you would use to connect to ssh_server_host in a regular SSH session. This may or may not be the same as the username you currently use on your local machine.

The Entire Command, Line by Line

- Line 1 simply calls the ssh program;
- Line 2 sets up the port forwarding. The **-L** parameter specifies that this is a remote-to-local tunnel. If you wanted to create a local-to-remote tunnel, you'd have used **-R** instead of **-L**. The next three parameters are from our list above, separated by colons. (Use slashes instead of colons for IPv6.) If you want to set up multiple tunnels from the same host, simply repeat line 2 as many times as you'd like, once for each set of **local_port:service_host:service_port**.
- Line 3 selects a port on the ssh_server_host to connect to. Omit this line if you're connecting to the default port 22.
- Line 4 specifies the username to use on the ssh_server_host. It is also possible to use the **ssh_server_username@ssh_server_host** syntax instead of the **-l** parameter.
- Line 5 indicates to ssh that no commands be run on the remote system. Since you're using this SSH connection simply for tunneling, this is a useful option to set.
- Line 6 contains the most basic parameter of this entire process. Please don't get this wrong.

Common Errors and Solutions

Error message: channel 3: open failed: connect failed: Connection refused

Change localhost to 127.0.0.1 in the ssh -L parameter.

Cannot listen on port X on local machine because of network policies.

Try to use another port locally. Ports such as 3306 (MySQL) may have been left open. These are good to use for SSH tunneling if you aren't already running MySQL.

Error message: Privileged ports can only be forwarded by root.

Use a port above 1024, or try to set up the SSH tunnel as root.

Error message: bind: Address already in use, channel_setup_fwd_listener: cannot listen to port: xxxx, Could not request local forwarding.

Some local server process is already listening on the local port you're trying to forward to. Pick a different local port and configure your program to connect to th at port instead. If your program cannot be configured to listen to a different port, try to find what server process is occupying that port (netstat -a on Linux or lsof -i -P on Mac OS X) and stop it. Retry setting up the tunnel.

I want other hosts on my network to be able to use the tunnel I established.

By default, only local clients can connect to SSH tunnels established this way.

Use the -g option when setting up the tunnel. Realize that this is insecure, but it may make sense in certain scenarios.

I don't know what local port is available for me to use.

Linux: netstat -a | grep LISTEN

Mac OS X: lsof -i -P | grep LISTEN

will show you the ports that are in use. Generally, you can pick any that's not already taken. To make sure you're not breaking some other unknown protocol, check the IANA Well-known Port Numbers list and pick one that's not taken.

If you've not been able to debug this so far, try passing the -v parameter to ssh to see verbose output. Add another -v for more verbose output.

If you're reading this, and come across any specific source of trouble, please let me know so I can add it to this mini HOWTO.