Finally, let's take a look at our last tool of this section: [sshuttle](https://github.com/sshuttle/sshuttle).

This tool is quite different from the others we have covered so far. It doesn't perform a port forward, and the proxy it creates is nothing like the ones we have already seen. Instead it uses an SSH connection to create a tunnelled proxy that acts like a new interface. In short, it simulates a VPN, allowing us to route our traffic through the proxy *without the use of proxychains* (or an equivalent). We can just directly connect to devices in the target network as we would normally connect to networked devices. As it creates a tunnel through SSH (the secure shell), anything we send through the tunnel is also encrypted, which is a nice bonus. We use sshuttle entirely on our attacking machine, in much the same way we would SSH into a remote server.

Whilst this sounds like an incredible upgrade, it is not without its drawbacks. For a start, sshuttle only works on Linux targets. It also requires access to the compromised server via SSH, and Python also needs to be installed on the server. That said, with SSH access, it could theoretically be possible to upload a static copy of Python and work with that. These restrictions do somewhat limit the uses for sshuttle; however, when it *is* an option, it tends to be a superb bet!

First of all we need to install sshuttle. On Kali this is as easy as using the apt package manager:  
sudo apt install sshuttle

The base command for connecting to a server with sshuttle is as follows:  
sshuttle -r username@address subnet

For example, in our fictional 172.16.0.x network with a compromised server at 172.16.0.5, the command may look something like this:  
sshuttle -r user@172.16.0.5 172.16.0.0/24

We would then be asked for the user's password, and the proxy would be established. The tool will then just sit passively in the background and Logo

Description automatically generatedforward relevant traffic into the target network.

Rather than specifying subnets, we could also use the -N option which attempts to determine them automatically based on the compromised server's own routing table:  
sshuttle -r username@address -N

Bear in mind that this may not always be successful though!

As with the previous tools, these commands could also be backgrounded by appending the ampersand (&) symbol to the end.

If this has worked, you should see the following line:  
c : Connected to server.

Well, that's great, but what happens if we don't have the user's password, or the server only accepts key-based authentication?

Unfortunately, sshuttle doesn't currently seem to have a shorthand for specifying a private key to authenticate to the server with. That said, we can easily bypass this limitation using the --ssh-cmd switch.

This switch allows us to specify what command gets executed by sshuttle when trying to authenticate with the compromised server. By default this is simply ssh with no arguments. With the --ssh-cmd switch, we can pick a different command to execute for authentication: say, ssh -i keyfile, for example!

So, when using key-based authentication, the final command looks something like this:  
sshuttle -r user@address --ssh-cmd "ssh -i KEYFILE" SUBNET

To use our example from before, the command would be:  
sshuttle -r user@172.16.0.5 --ssh-cmd "ssh -i private\_key" 172.16.0.0/24

**Please Note:** When using sshuttle, you may encounter an error that looks like this:  
client: Connected.  
client\_loop: send disconnect: Broken pipe  
client: fatal: server died with error code 255

This can occur when the compromised machine you're connecting to is part of the subnet you're attempting to gain access to. For instance, if we were connecting to 172.16.0.5 and trying to forward 172.16.0.0/24, then we would be including the compromised server inside the newly forwarded subnet, thus disrupting the connection and causing the tool to die.

To get around this, we tell sshuttle to exclude the compromised server from the subnet range using the -x switch.

To use our earlier example:  
sshuttle -r user@172.16.0.5 172.16.0.0/24 -x 172.16.0.5

This will allow sshuttle to create a connection without disrupting itself.