Secure Shell (SSH) Tunnelling - Three Examples

Mark G.

February 24, 2022

This presentation lightly introduces the secure shell (SSH). We'll look at very specific configuration parameters and command line options to demonstrate three ways to use SSH with other applications.

SSH software is able to provide a private network tunnel for these applications, and by doing so, protects authentication credentials and content.

The three applications discussed are a web browser (Firefox) configured to use the SOCKS5 proxy protocol; X Windows and its associated network protocol; and system administration using virtual network computing (VNC).

Contents

List of Figures

1 Systems and Apparatus

Technologies used in this demonstration are:

- Secure Shell (SSH) client and server software.
- Virtual Network Computing (VNC) software.
- X server and X client software.
- XUbuntu 19.04 workstation (web browser for SOCKS testing, X server).
- Raspberry Pi 3/4 (X Client, VNC Server).
- Ubuntu 20.04.3 LTS Server (SOCKS5 Proxy).
- Microsoft Windows 10 workstation (VNC client)

2 Nomenclature and Files

The following is a list of definitions that we'll use in the presentation.

ssh client host The source host from which we start a connection, uses `ssh`. Most likely our usual workstation. Does not need to be running a secure shell server, i.e. `sshd`.

ssh server host The target host to which we connect, runs the secure shell server: `sshd`. Servers will act as proxy servers, or be subject to administration via remote desktop software.

X server The X server displays and manages X client programs. It runs on our usual workstation and will show us the `ssh server host's` X client programs, such as `firefox`.

X client The X client software (i.e. `firefox`) is run on the `ssh server host` and displayed on the `ssh client host's` X server.

The next two lists are summaries of the files used by SSH.

SSH client files are located on both the `ssh client host` and on the `ssh server host`, in the user's home directory.

$\sim$ represents the computer user's home directory, e.g. `/home/pi`.

$\sim$**/.ssh/** The user's personal, hidden, SSH configuration directory (on both `ssh client host` and user's home directory on the `ssh server host`).

~/.ssh/config The user's default SSH client configuration file (on `ssh client host`).

~/.ssh/authorized_keys The user file that contains public keys used for authentication (on `ssh server host`).

~/.ssh/known_hosts The user's list of already contacted SSH servers' public keys. This file is updated whenever a client connects to a new SSH server (on `ssh client host`).

Relevant SSH server configuration files are listed below.

/etc/ssh/ The SSH server's configuration directory.

/etc/ssh/sshd_config The SSH server's default configuration file.

3 SSH Server Configuration Considerations

We need an SSH server / daemon (`sshd`) running on the `ssh server host(s)`. It must allow the various forwarding options (X and port) at the server level. If these options are set to `no` then none of the tunnelling described herein will work. The usual location for these options is the `sshd_config` file in directory `/etc/ssh/` (or just `/etc/` on some older systems).

- `AllowTcpForwarding yes` - This allows local and remote port forwarding, as well as dynamic port forwarding.
- `#GatewayPorts no` - Only allow the local host to use forwarded ports, rather than allowing other systems to connect to the `ssh client host` and also use the forwarded ports. A good security setting, leave it as set.
- `X11Forwarding yes` - Set to allow forwarding of the X protocol (tcp port 6000 + X11DisplayOffset = 6010).
- `#X11DisplayOffset 10` - X displays are numbered starting with 0, this offset removes the possibility of the SSH tunnelled display conflicting with the workstation's X display number. We don't need to adjust this.
- `#X11UseLocalhost yes` - The X server will only listen on the localhost interface, a good security setting, leave it as set.

The `#` symbol means that the value uses the default as shown. Most SSH server configurations allow port and X forwarding by default.

3.1 Matching Users

It is often a good idea to have features turned off as a default. To this end, we can turn off all forwarding options, for all users in the main portion of the `sshd_config` file:

```
AllowTcpForwarding no
X11Forwarding no
```

Now, for each user that requires some forwarding feature, we create a specific configuration section for them using the `Match` directive.

```
Match user pi
  AllowTcpForwarding yes
  X11Forwarding yes
```

Now only the `pi` user will be able to use forwarded ports. The `Match` directives must be at the end of the `sshd_config` file. More than one can be placed one after the other.

4 SOCKS5 Proxy Protocol Support Configuration

The first application using SSH tunnels is a web browser (`firefox` in this case) that uses a proxy, that is, a different, so-called 'route' to the Internet. This proxy is known as the SOCKS version 5 (SOCKS5) protocol. It allows a SOCKS aware application program to redirect its connections through the SOCKS proxy server (i.e. the `ssh server host`).

For example, a browser will send its web requests and DNS queries to a specific proxy host using the SOCKS version 5 protocol. Those requests will appear to come from the proxy host.

This may be a corporate requirement or may be used as a poor man's virtual private network (VPN). This requires the `AllowTcpForwarding` option to be set to `yes` on the `ssh server host`.

As a baseline, here is an image of the browser's results of an IP address query before we use the proxy.

Note the 70.66.* personal ISP address.

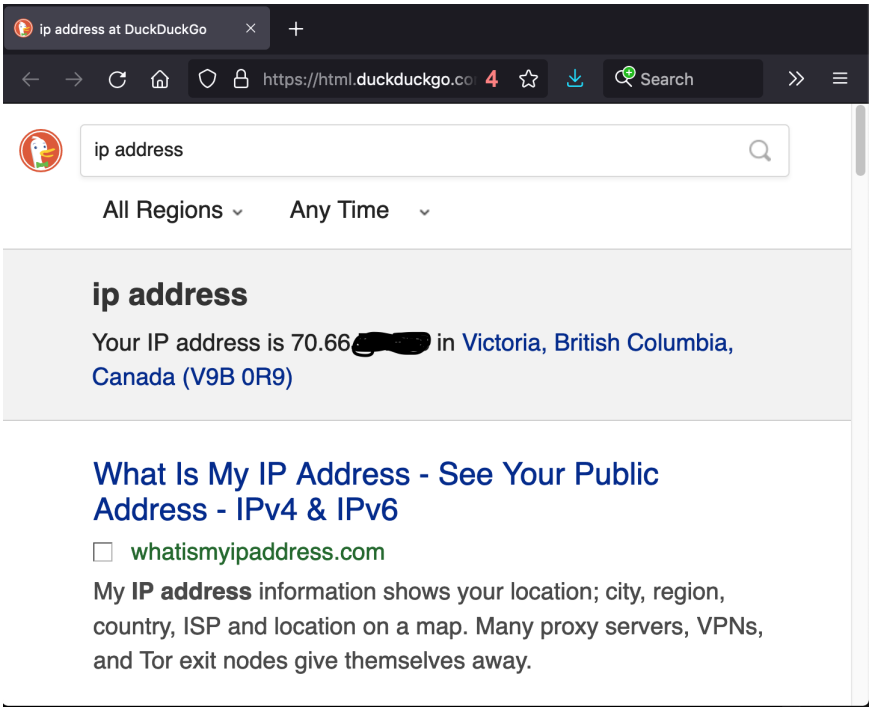

Figure 1: **R**esults of an IP address query without proxy

4.1 Command Line Options for SOCKS5

In this demonstration, we'll use an IPv6 SSH connection, since we have an IPv6 enabled SSH server available. This means we will be able proxy IPv4 and IPv6 connections through the SOCKS5 proxy server.

The SSH client connection command will look like this:

```
$ ssh -6 -D 1080 pi@fdea:557c:9747:1060::62
```

The options explained:

1. `-6` connect over IPv6.
2. `-D 1080` specify the local port to use as the SOCKS5 proxy port. There is an implied `::1` host address to which the dynamic port forwarding feature is bound.
3. `pi@fdea:557c:9747:1060::62` the SSH server to connect to that will be the SOCKS5 proxy server.

4.2 Browser Network Settings Options

The browser in this case is `firefox`. We have to set the browser to use a proxy via the preferences menu under Network Settings.

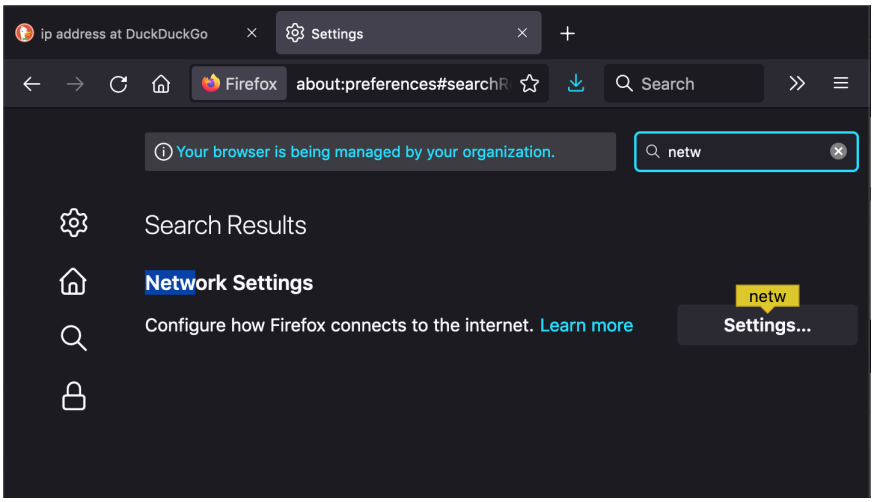

Figure 2: **F**irefox Settings - Search for 'netw' to quickly show Network Settings

When the network settings dialog opens, we specify that the browser will use a `Manual proxy configuration`. In the SOCKS host field we enter the IPv6 localhost address `::1`, and set the port to the well know SOCKS port of 1080 (this must match our `ssh` command line value for the `-D` option).

Make sure the radio buttons declaring the type of proxy is set to 'SOCKS v5' as well.

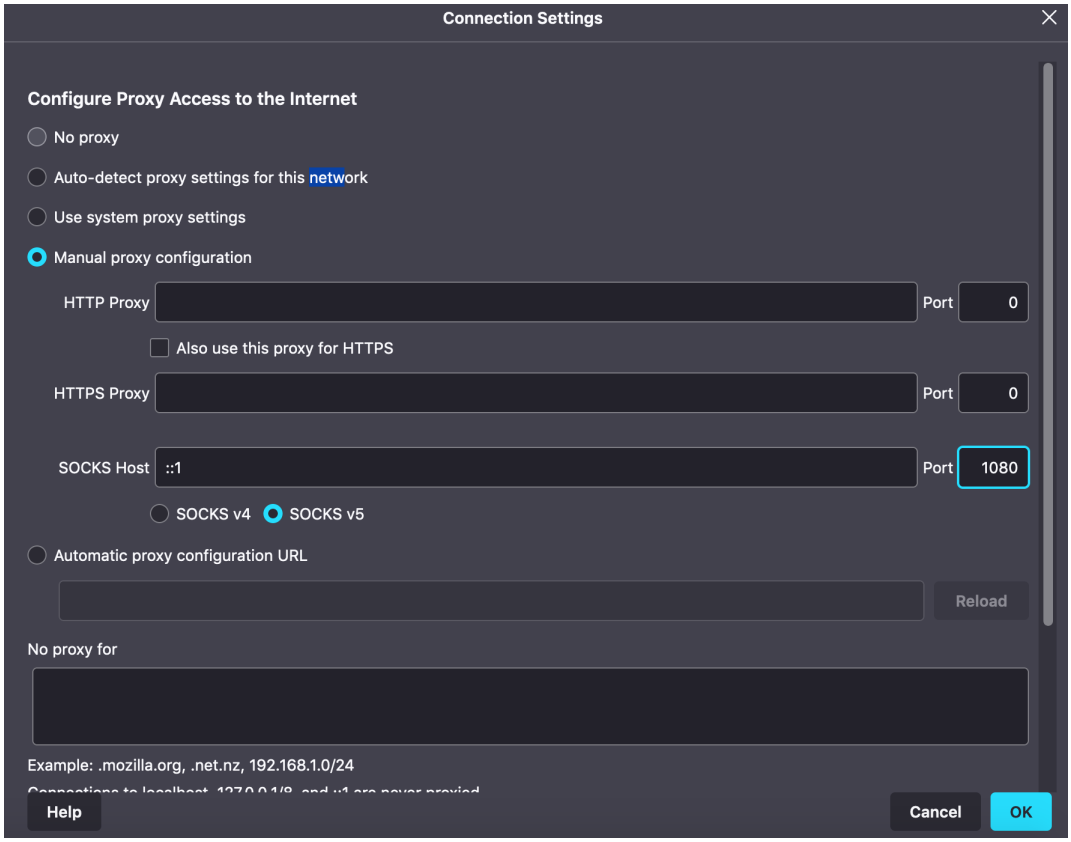

Figure 3: **F**irefox network settings - Manual Proxy

We are not done yet, and scrollong further down the settings page shows a DNS related checkbox labelled 'Proxy DNS when using SOCKS v5' that we should enable:

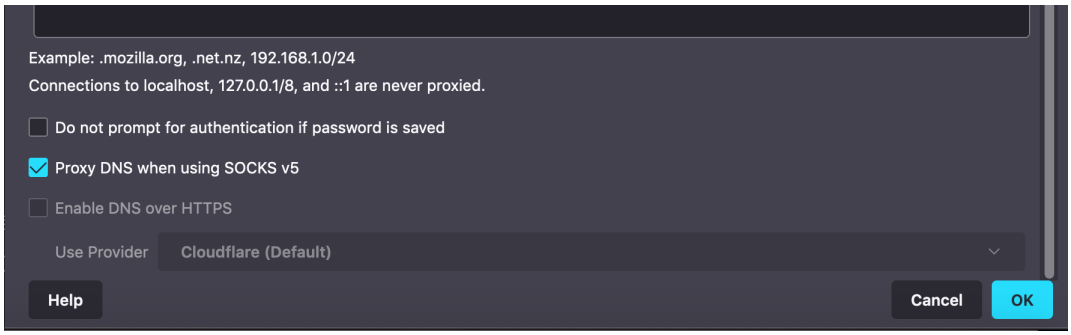

Figure 4: **F**irefox network settings - Use SOCKS5 Proxy for DNS

Save the settings and return to the browser proper. Make sure the SOCKS5 `ssh` com-

mand is running, and we can now check what our IP address is while using the proxy:

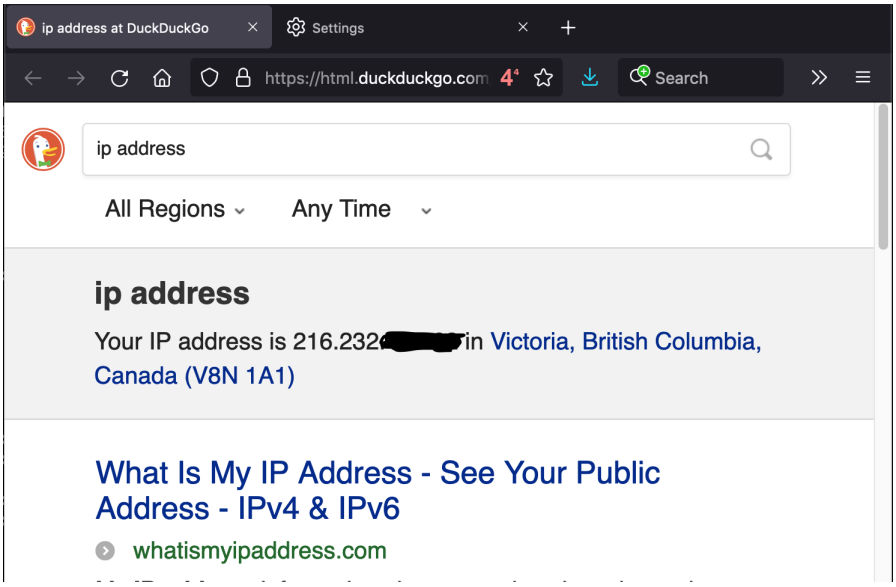

Figure 5: **R**esults of an IP address query using the proxy

Note the 216.232.* business ISP address.

4.3 SSH Client Config for SOCKS5

The `ssh client host` can use the ~`/.ssh/config` file to store groups of command line options for various `ssh server hosts`. The quality of life config entry for our SOCKS proxy host is shown below:

```
Host sockshost
 Hostname fdea:557c:9747:1060::62
 AddressFamily inet6
 Port 22
 User pi
 DynamicForward 1080
```

It can be invoked simply as:

```
$ ssh sockshost
```

The above command will give you a terminal session and command prompt on the `ssh server host`, a.k.a. the SOCKS5 proxy. To end the SOCKS5 proxy session, just log out of the terminal session on the `ssh server host`.

4.4 Don't Run A Shell on the SSH Server Host

If you don't want to start a terminal on the `ssh server host` you can use the `-N` option to not execute a remote command. This is useful for just forwarding ports.

```
$ ssh -N sockshost
```

To end the above SOCKS5 session, you need to use the keyboard interrupt sequence: `CTRL-C`.

Note about the `-N` option: when in use, the ssh command on the `ssh client host` will not result in a terminal session on the `ssh server host`. The `ssh -N...` command will simply remain in the foreground of the `ssh client host` terminal, and ignore input with some exceptions, mainly the interrupt key sequence, i.e. `CTRL-C`. Indeed, `CTRL-C` is the usual way to terminate this form of proxy session, and return the command prompt to the `ssh client host's` terminal.

4.5 Start Forwarding And Go Away

You can use the `-f` option with the `-N` option to regain the `ssh client host's` terminal command prompt, since `-f` tells the `ssh` command to switch into the background.

```
$ ssh -fN sockshost
```

You get your `ssh client host's` terminal prompt back and can continue working with that terminal. The downside is that in order to terminate the backgrounded SOCKS5 proxy session, you must find the process ID of the `ssh -fN...` command and use the process `kill` command. More complicated than either `CTRL-C` or logging out.

To find the `ssh` process, use `ps -ef` with `grep`:

```
$ ps -ef | grep ssh
  501  9302     1   0 17Jan22 ??         0:00.10 /usr/bin/ssh-agent -l
  501 34677     1   0  5:30pm ??         0:00.00 ssh -fN sockshost
  501 34681   387   0  5:30pm ttys001    0:00.00 grep ssh
```

Our command has process ID 34677 (second column). Terminate it using:

```
$ kill 34677
```

I find that just using a terminal on the `ssh server host` and logging out when done is simpler than using either or both of the `-N` and `-f` options.

5 X Forwarding Configuration

X Windows has been around the Unix universe for many years and is still being used on many operating systems. Many linux based desktop environments use X Windows, as does FreeBSD. Apple's MacOS has third party support for X Windows through the `XQuartz` software[1].

Forwarding X programs means 'to run a program on the `ssh server host` (which makes it the `X client`), and display its graphical user interface (GUI) on the `ssh client host`'. It will also accept keyboard and mouse input from the `ssh client host` workstation (which is the `X server`).

Some uses for this feature are: remote administration, take advantage of a faster / more powerful CPU or other resources. Developers and programmers can use this to run a resource heavy integrated development environment (IDE) program on a development server.

This requires the `X11Forwarding` option to be set to `yes` on the `ssh server host`.

The basic SSH client connection command will look like this:

```
$ ssh -f -Y pi@10.13.0.53 geany
```

The command connects to `ssh server host` at the given address, using the specified user and requests that an X11 forwarding (trusted) session be established. The options explained:

1. `-f` requests ssh to go to background just before command execution.
2. `-Y` specifies that trusted X forwarding is to be used. There is another option `-X` which can be tried first. I just prefer to use `-Y` right off the bat, as sometimes `-X` will fail, while `-Y` works.

[1] Find it at: https://www.xquartz.org/

3. `pi@10.13.0.53` is the `ssh server host` to which we connect that will host the X client.

4. `geany` is the X program to execute on the remote host. `geany` is a python language IDE.

5.1 SSH Client Config for X Forwarding

We can create an SSH client config section in the ~`/.ssh/config` file to more easily manage the ssh connection:

```
Host piholex
 Hostname 10.13.0.53
 Port 22
 User pi
 ForwardX11 yes
 ForwardX11Trusted yes
 ExitOnForwardFailure yes
```

If you have the above config `Host` defined then use this command:

```
$ ssh -f piholex geany
```

The python mini-IDE will display on your `ssh client host's` X window server, i.e. your desktop.

6 Port Forwarding Configuration for VNC

The third tunnelling application is to use SSH port forwarding to facilitate securing VNC to allow remote administration of systems. The example for this section uses a Microsoft Windows 10 computer and a raspberry pi server. It is necessary to install a VNC viewer on the windows system and a VNC server on the raspberry pi. Appendix 8 has the former while appendix 9 has the latter.

A VNC server uses a default TCP port of 5901 for a display value of `:1`. If the display value is `:2` or `:3`, then the TCP port is 5902 or 5903, respectively. This is the port value that we will be forwarding and it is the value to provide to the VNC viewer.

Here is an image from Wikipedia[2] that is a decent representation of how port forwarding works. The top image covers our use case.

[2]https://en.wikipedia.org/wiki/File:Ssh-L-Tunnel.png

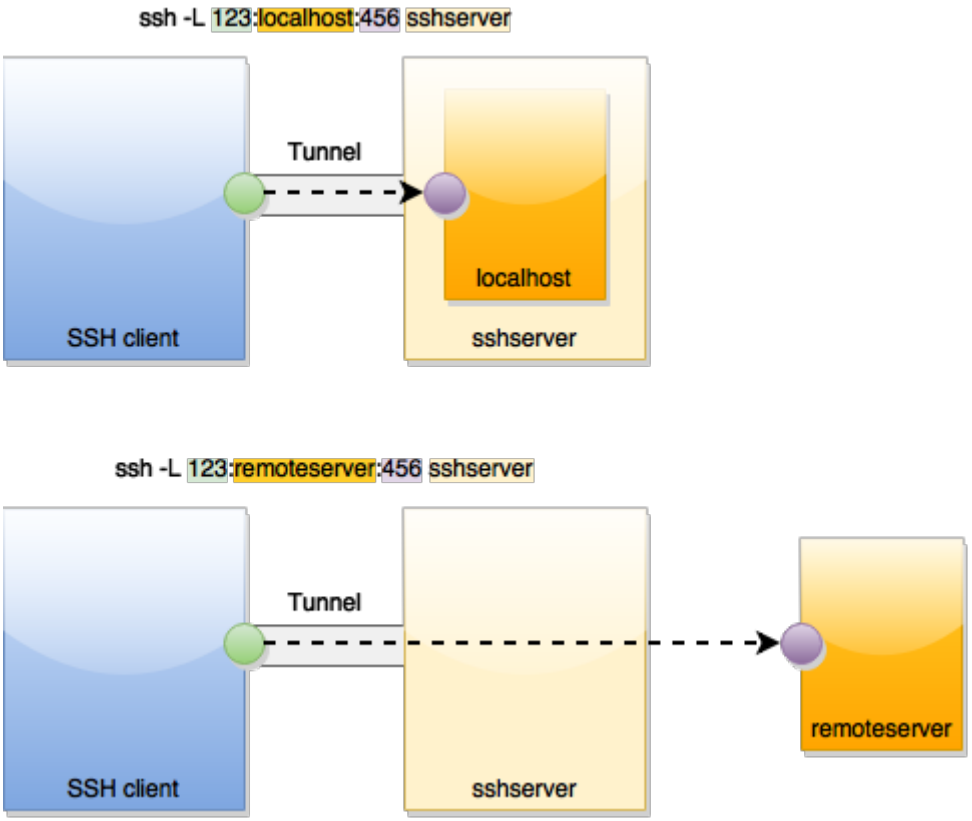

Figure 6: **A**bstract image of port forwarding using SSH (Creative Commons from Wikipedia)

Microsoft Windows 10 now comes with an `ssh` client program. I'm not going to cover `PuTTY` or other versions of Windows, since there are many tutorials already available.

In windows, the user has a home directory such as `C:\Users\games>` and the relevant SSH files are the same as on a Unix system:

- `.ssh/` - directory holding SSH configuration files and key related files.
- `.ssh/config` - file holding SSH configuration options.
- `.ssh/known_hosts` - previously contacted SSH servers' host keys.

The windows command prompt program is where we run our `ssh client host` commands and we'll work in the `.ssh` directory.

```
C:\Users\gamez>cd .ssh
C:\Users\gamez\.ssh>
```

The `ssh` client command for forwarding port 5901 from the `ssh client host` (windows workstation) to the `ssh server host` is as follows:

```
C:\Users\gamez\.ssh>ssh -L 5901:localhost:5901 pi@192.168.0.213
```

The options explained:

1. `-L 5901:localhost:5901` requests that ssh intercept network connections on the `ssh client host`, to local port 5901 and forward them across the SSH connection to the `ssh server host`. The service that listens on the `ssh server host's` loopback on port 5901 will answer the network requests.
2. `pi@192.168.0.213` is the `ssh server host` to which we connect that will host the VNC server.

The results are:

```
pi@192.168.0.213's password:
Linux raspberrypi 5.10.63-v7l+ #1459 SMP Wed Oct 6 16:41:57 BST 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
...
pi@raspberrypi:~ $
```

Now that we are logged into the raspberry pi (`ssh server host`) we can start the VNC server:

```
pi@raspberrypi:~ $ vncserver -localhost -nolisten tcp

New 'X' desktop is raspberrypi:1

Starting applications specified in /home/pi/.vnc/xstartup
Log file is /home/pi/.vnc/reaspberrypi:1.log

pi@raspberrypi:~ $
```

It is important to note that the first run of the `vncserver` program is when the VNC session password is set.

The option `-localhost` instructs the VNC server to listen on the loopback interface only (no external internet addresses), and the `-nolisten tcp` option tells the X server used by the VNC server to disable listening on TCP ports.

6.1 Connecting using TightVNC On Windows

Run the TightVNC Viewer from the start menu and we see the new connection dialog:

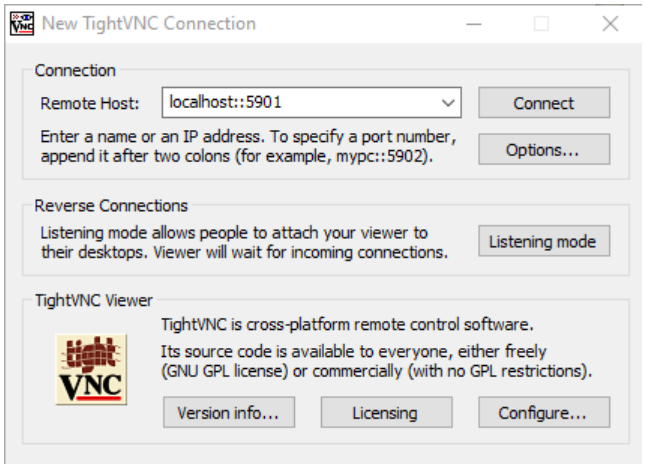

Figure 7: **T**ightVNC New Connection Dialog

Enter our VNC server's socket information, which is localhost::5901. Notice that there are two colons, rather than one. This is confusing, but is needed. We get the VNC password prompt, and the desktop comes up:

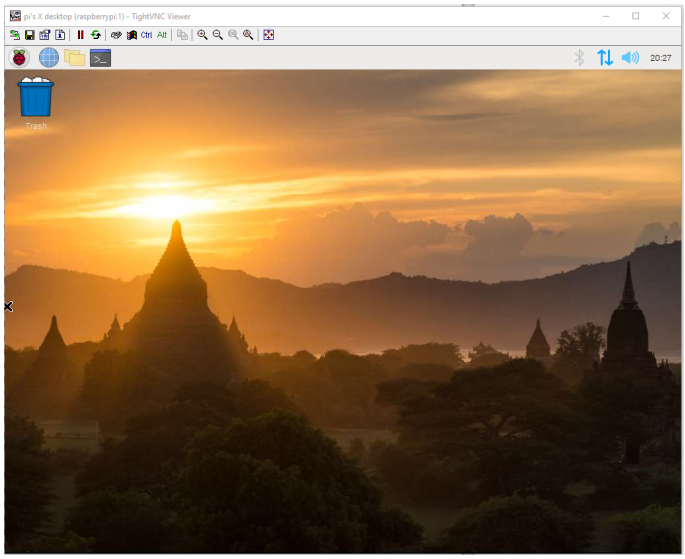

Figure 8: **T**ightVNC Raspberry Pi Desktop

6.2 SSH Client Config for VNC Port Forwarding

We can convert the ssh command line:

```
$ ssh -L 5901:localhost:5901 pi@192.168.0.213
```

into a config file entry named `rpidesktop`:

```
Host rpidesktop
 Hostname 192.168.0.213
 Port 22
 User pi
 LocalForward 5901 localhost:5901
 ExitOnForwardFailure yes
```

We then use it to save some typing:

```
C:\Users\gamez\.ssh>ssh rpidesktop
pi@192.168.0.213's password:
Linux raspberrypi 5.10.63-v7l+ #1459 SMP Wed Oct 6 16:41:57 BST 2021 armv7l
...
```

Start the VNC viewer as before.

You should create a key pair for this connection and perhaps use the SSH agent as well, if you are connecting a lot. An example of creating a key pair is in the later bonus section.

6.3 Starting VNC Server At System Boot

Instead of manually starting the VNC server each time we connect via SSH, we can start the VNC server on the raspberry pi at system reboot by editing the root user's crontab as follows:

```
pi@raspberrypi:~ $ sudo crontab -e
```

Go to the bottom of the file and add this line:

```
@reboot su - pi -c '/usr/bin/vncserver -geometry 1280x720 -localhost -nolisten tcp'
```

After reboot, we see the process:

```
pi@raspberrypi:~ $ ps -uax |grep -i vnc
pi         620  0.3  0.3  46048 14280 ?        S    20:53   0:00 Xtightvnc :1 -desktop X \
 -auth /home/pi/.Xauthority -geometry 1280x720 -depth 24 -rfbwait 120000 \
 -rfbauth /home/pi/.vnc/passwd -rfbport 5901 \
 -fp /usr/share/fonts/X11/misc/,/usr/share/fonts/X11/Type1/,/usr/share/fonts/X11/75dpi/,/usr/share/fonts/X11/100dpi/ \
 -co /etc/X11/rgb -localhost -nolisten tcp
pi         624  0.0  0.0   1940   380 ?        S    20:53   0:00 /bin/sh /home/pi/.vnc/xstartup
```

7 Bonus X Forwarding Configuration for VNC

This particular configuration is not normal. It was created because I couldn't install a VNC viewer client on the Xubuntu linux workstation. The reasons are out of the scope of this presentation, but make for a good rant.

We use X forwarding instead of port forwarding and the X program we run on the `ssh server host` will be the VNC viewer. The process is as follows:

1. Execute the ssh connection command.
2. Start the VNC server in the resulting terminal session, specifying an explicit display number.
3. Start the VNC viewer program in the same terminal session, which will be forwarded over SSH since it is an X program. We use the display number specified for the VNC server.
4. When the VNC viewer exits, we kill the VNC server running on the explicitly specified display.

The SSH client connection command will look like the usual X forwarding command:

```
$ ssh -Y pi@10.13.0.53
```

Once connected to the `ssh server host`, we issue a command to start the VNC server:

```
raspberrypi$ tightvncserver :1 -localhost -nolisten tcp
```

The added option `:1` instructs the VNC server to use the explicit display numbered 1. The usual localhost and tcp listening options are included. This command will automatically go into the background, leaving us back at the `ssh server host` command prompt.

Since the VNC server prints that it is listening on `raspberrypi:1` we tell the VNC viewer software to use the explicit display `:1`.

```
raspberrypi$ xtightvncviewer :1
```

There will be a password prompt in the terminal window for the VNC server and the resulting VNC viewer window will be displayed on the `ssh client host's` display.

When we're done administering the system and close the VNC viewer X program on our `ssh client host's` display, the above command will terminate and return the command prompt on the `ssh server host`.

We can now shut down the VNC server using the `-kill` command line option:

```
raspberrypi$ tightvncserver -kill :1
```

7.1 Using a Shell Script for Automation

We can create a shell script to contain the three commands we used on the `ssh server host` to automate the process of running the VNC server / viewer.

On the `ssh server host`, in the `pi` user's home directory, create a `bin` directory to hold the automation script.

```
raspberrypi$ cd
raspberrypi:~ $ mkdir bin
raspberrypi:~ $ cd bin
raspberrypi:~/bin $
```

Create a shell script with the following contents and place it in the `bin/` directory:

```
raspberrypi:~/bin $ cat ssh-vnc.sh
#!/bin/sh

TVNC=/usr/bin/tightvncserver
XTVNC=/usr/bin/xtightvncviewer

${TVNC} :1 -localhost -nolisten tcp
${XTVNC} :1
${TVNC} -kill :1
```

Make it executable:

```
raspberrypi:~/bin $ chmod +x ssh-vnc.sh
```

Now when we want to administer the raspberry pi server we can use this command line on the `ssh client host` (our Xubuntu workstation):

```
mv@think ~/.ssh $ ssh -Y pi@10.13.0.53 /home/pi/bin/ssh-vnc.sh
```

The only difference in this invocation is that the VNC server password prompt is displayed onscreen rather than in the terminal window. When we exit the viewer, the VNC server will be terminated automatically.

7.2 Forced Commands and Keys

This is a convenience configuration that uses the shell script from the previous section along with a key pair and the forced command option. As above, we want to start the VNC server, run the X client that we are forwarding, in this case, the VNC viewer, and then close the VNC server when the client exits, i.e. the user is done.

To do this, we create a key pair to use as SSH authentication. This pair is created on the `ssh client host`. We protect it with a passphrase and give it a descriptive name and a comment.

```
$ cd .ssh
mv@think ~/.ssh $ ssh-keygen -t ecdsa -f pi_from_think_to_pihole \
  -C "Pi from think to pihole VNC"
Passphrase:
Confirm passphrase:
```

The comment `-C "..."` is stored in the public key file and is recommended to help remind future you what the key pair in the `authorized_keys` file is for.

Copy our newly created key to the `ssh server host`, i.e. the `pihole` (10.13.0.53) system. Note that `ssh-copy-id` requires password authentication if there are no keys already present on the target host.

```
mv@think ~/.ssh $ ssh-copy-id -i pi_from_think_to_pihole pi@10.13.0.53
Password:
```

The public key portion of the key pair should now be on the `ssh server host` in the pi user's ∼`/.ssh/authorized_keys` file.

Now we used the `command="..." authorized_keys` file directive to force the use of the key pair to a specific command:

```
pi@raspberrypi ~/.ssh $ cat authorized_keys
command="/home/pi/bin/ssh-vnc.sh" ecdsa-sha2-nistp256 AAAA....
lkasjdlkjd= Pi from think to pihole VNC
```

We can now use the new key as our identity when connecting. Use the `-i file` option to `ssh` to tell it to use the private key found in the file, rather than prompting for a password. It is important to note that the prompt when using an identity file is for the passphrase protecting the private key / identity file.

```
mv@think ~/.ssh $ ssh -i pi_from_think_to_pihole pi@10.13.0.53
Key Passphrase:
```

You should be prompted for the key passphrase and then be prompted for the VNC server password.

7.3 Alternate SSH Client Config File

It is useful to store the extra identity parameters for our `ssh` commands in an SSH client config file. We'll create a new one called `config.keys` and have it contain the following:

```
Host piholex
 Hostname 10.13.0.53
 Port 22
 User pi
 ForwardX11 yes
 ForwardX11Trusted yes
 ExitOnForwardFailure yes
 IdentityFile  ~/.ssh/pi_from_think_to_pihole
```

Use it:

```
mv@think ~/.ssh $ ssh -F config.keys piholex
Key Passphrase:
```

Once again, the VNC viewer program is displayed (after the authentication prompt).

7.4 SSH Agent

We can start the `ssh-agent` program so that it will store our private keys in memory and handle SSH connections without repeatedly prompting for private key passphrases. Normally an SSH agent should be started at login, before any window managers are started so as to allow a single agent to be used by all applications/terminals. We needn't set that up now. For our local terminal session, we can start the agent and set the environment as follows:

```
mv@think ~/.ssh $ eval `ssh-agent -s`
```

We can add our key manually to the running agent:

```
mv@think ~/.ssh $ ssh-add pi_from_think_to_pihole
Key Passphrase:
```

Now when we use the key to connect to the server, the agent will automatically verify the connection without prompting us for a passphrase.

```
mv@think ~/.ssh $ ssh -F config.keys piholex
 --- VNC Viewer Displays ---
```

To remove the requirement to run `ssh-add` we can place an option in the SSH client config file called `AddKeysToAgent yes`:

```
Host piholex
 Hostname 10.13.0.53
 Port 22
 User pi
 ForwardX11 yes
 ForwardX11Trusted yes
 ExitOnForwardFailure yes
 IdentityFile  ~/.ssh/pi_from_think_to_pihole
 AddKeysToAgent yes
```

Now the first time after we've started the agent, when we use an `ssh` client command that uses an identity file, we will be prompted for the passphrase that one time, but subsequent connections will not require the passphrase. We don't need to run `ssh-add` either.

8 Appendix - VNC Viewer Installation on Windows

Get tightvnc here: https://www.tightvnc.com/download.php

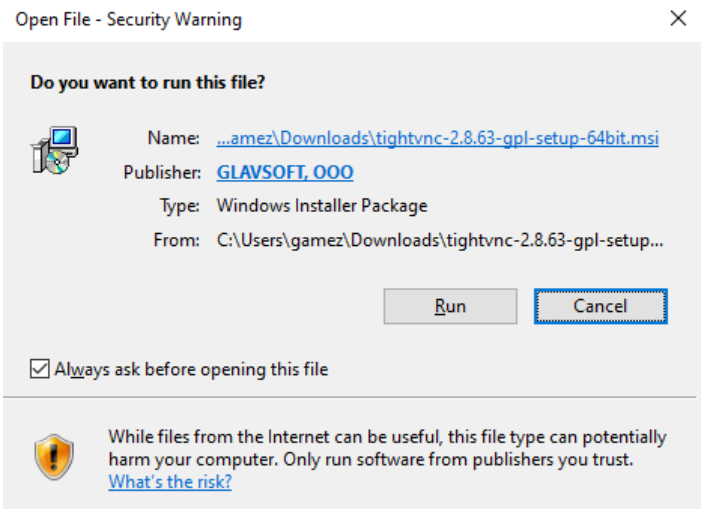

Figure 9: **T**ightVNC Software Vendor Security Notification

Select Run and we are greeted with the welcome message:

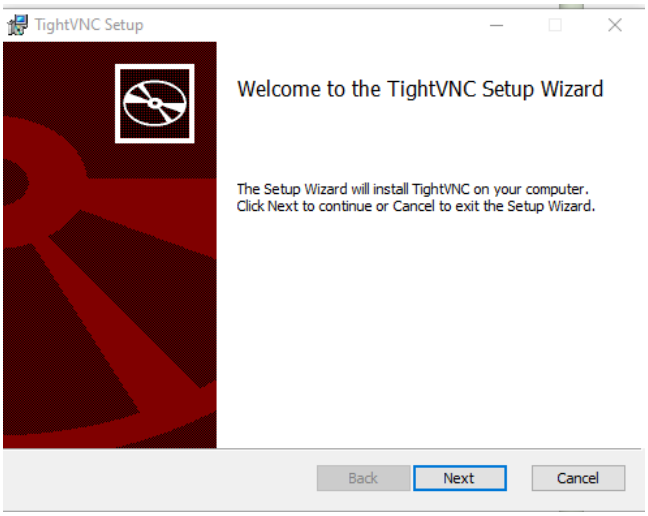

Figure 10: **T**ightVNC Installation Welcome

A license agreement section is presented, which you will agree with.

The next step during the installation gives us the option to do a custom install.

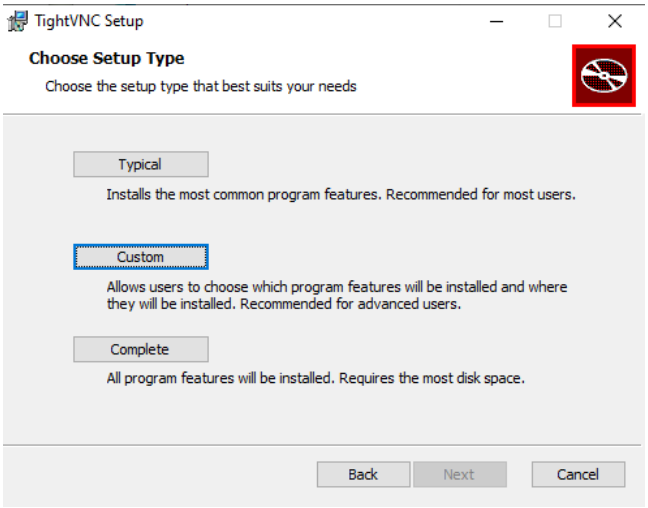

Figure 11: **T**ightVNC Setup Options - choose custom

We choose custom so we can turn off the installation of the VNC server on the Windows PC. There is no need for it, so don't install it. We only need the VNC viewer program.

Make sure your installation options have the red X on the TightVNC Server item, like this:

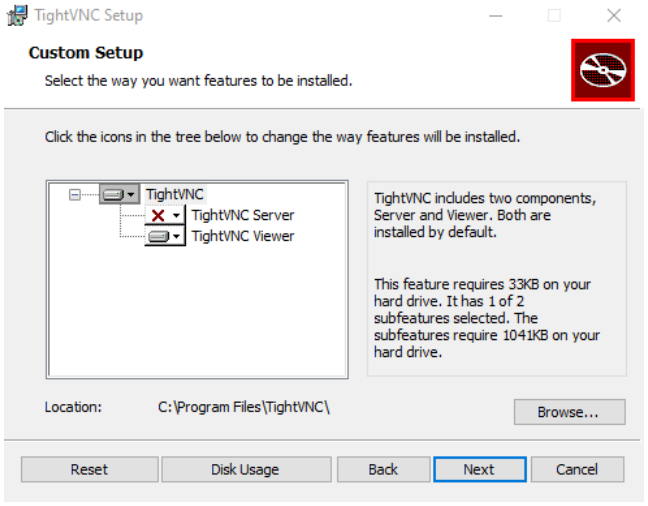

Figure 12: **T**ightVNC Setup Options - No Server Component

Select additional tasks and follow the rest of the prompts (images not shown).

The newly installed VNC viewer can be seen in the start menu:

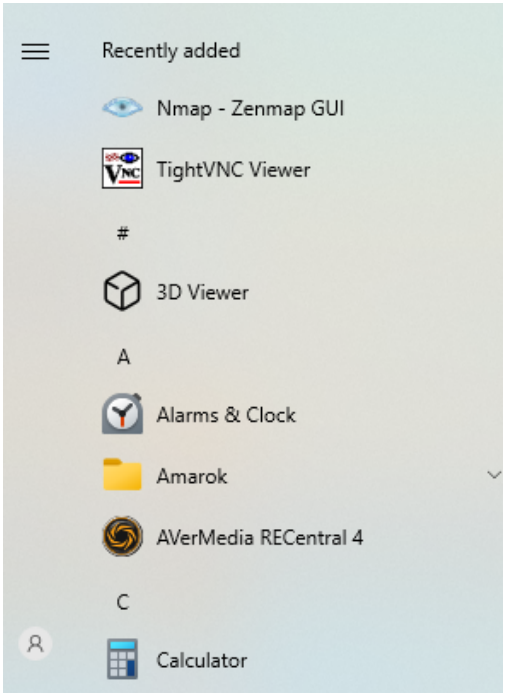

Figure 13: **T**ightVNC Viewer Start Menu Item

9 Appendix - VNC Software Installation on Raspberry Pi

On the Raspberry Pi, install `tightvncserver`, but first remove `realvnc-vnc-server` and `realvnc-vnc-viewer` (if it is also installed).

Removal:

```
root@raspberrypi:~# apt remove realvnc-vnc-server
...
The following packages will be REMOVED:
  realvnc-vnc-server
0 upgraded, 0 newly installed, 1 to remove and 128 not upgraded.
After this operation, 37.2 MB disk space will be freed.
Do you want to continue? [Y/n]
...

root@raspberrypi:~# apt remove realvnc-vnc-viewer
...
The following packages will be REMOVED:
```

```
  realvnc-vnc-viewer
0 upgraded, 0 newly installed, 1 to remove and 128 not upgraded.
After this operation, 6,725 kB disk space will be freed.
Do you want to continue? [Y/n]
...
```

Install:

```
root@raspberrypi:~# apt install tightvncserver
...
The following additional packages will be installed:
  xfonts-base
Suggested packages:
  tightvnc-java
The following NEW packages will be installed:
  tightvncserver xfonts-base
0 upgraded, 2 newly installed, 0 to remove and 128 not upgraded.
Need to get 6,448 kB of archives.
After this operation, 8,758 kB of additional disk space will be used.
Do you want to continue? [Y/n]
...
Setting up tightvncserver (1:1.3.9-9+deb10u1) ...
update-alternatives: using /usr/bin/tightvncserver to provide /usr/bin/vncserver (vncserver) in auto mode
update-alternatives: using /usr/bin/Xtightvnc to provide /usr/bin/Xvnc (Xvnc) in auto mode
update-alternatives: using /usr/bin/tightvncpasswd to provide /usr/bin/vncpasswd (vncpasswd) in auto mode
...
```

It is important to note that a quick first run of the `vncserver` program is required to set the VNC session password.

We can also add the viewer software to demonstrate using it as an X program.

```
root@raspberrypi:~# apt install xtightvncviewer
...
The following NEW packages will be installed:
  xtightvncviewer
0 upgraded, 1 newly installed, 0 to remove and 128 not upgraded.
...
Setting up xtightvncviewer (1:1.3.9-9+deb10u1) ...
update-alternatives: using /usr/bin/xtightvncviewer to provide /usr/bin/vncviewer (vncviewer) in auto mode
Processing triggers for man-db (2.8.5-2) ...
```

10 Appendix - VNC Server Network Security

It is advisable to be aware of VNC network socket behaviour. The following section shows how to observe and disable network features of the VNC server sofware.

What network sockets are being used on the pi (only relevant ones are shown)?

```
pi@raspberrypi:~ $ sudo netstat -lupnt
```

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      556/sshd
tcp        0      0 127.0.0.1:5901          0.0.0.0:*               LISTEN      948/Xtightvnc
tcp6       0      0 :::22                   :::*                    LISTEN      556/sshd
...
```

Our VNC server, that we started on the pi manually with the command seen above, is shown in the third tcp line:

```
tcp        0      0 127.0.0.1:5901          0.0.0.0:*               LISTEN      948/Xtightvnc
```

Its listening socket is: 127.0.0.1:5901

We also have the SSH server listening on all IPv4 interfaces (0.0.0.0:22) and IPv6 interfaces (:::22).

Let's do an Nmap scan of the pi host (get nmap from https://nmap.org/download.html):

```
C:\Users\gamez>nmap 192.168.0.213
Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-21 19:50 Pacific Standard Time
Nmap scan report for 192.168.0.213
Host is up (0.00098s latency).
Not shown: 999 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: E4:5F:01:5D:49:19 (Raspberry Pi Trading)

Nmap done: 1 IP address (1 host up) scanned in 0.24 seconds
```

Compare this line to the listening sockets displayed when running the `vncserver` command without the `-localhost` and `-nolisten tcp` command line options.

First, close the previously started VNC server:

```
pi@raspberrypi:~ $ vncserver -kill :1
Killng Xtightvnc process ID 948
```

The :1 is the display number (the default) and corresponds to TCP port 5901.

Restart without options:

```
pi@raspberrypi:~ $ vncserver

New 'X' desktop is raspberrypi:1
```

```
Starting applications specified in /home/pi/.vnc/xstartup
Log file is /home/pi/.vnc/reaspberrypi:1.log

pi@raspberrypi:~ $
```

What do the network sockets look like now?

```
pi@raspberrypi:~ $ sudo netstat -lupnt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      556/sshd
tcp        0      0 0.0.0.0:5901            0.0.0.0:*               LISTEN      6180/Xtightvnc
tcp        0      0 0.0.0.0:6001            0.0.0.0:*               LISTEN      6180/Xtightvnc
tcp6       0      0 :::22                   :::*                    LISTEN      556/sshd
```

There are two significant changes to the list of listening sockets.

1. The previous address:port socket entry of 127.0.0.1:5901 was bound only to the localhost address (127.0.0.1), but is now bound to all interfaces on the system (designated by the address 0.0.0.0).
2. There is a new listening socket entry 0.0.0.0:6001, which is an X server protocol socket. There is now an X server listening on all interfaces on the network. This is usually undesirable.

An nmap scan shows:

```
C:\Users\gamez>nmap 192.168.0.213
Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-21 19:49 Pacific Standard Time
Nmap scan report for 192.168.0.213
Host is up (0.00039s latency).
Not shown: 997 closed tcp ports (reset)
PORT     STATE SERVICE
22/tcp   open  ssh
5901/tcp open  vnc-1
6001/tcp open  X11:1
MAC Address: E4:5F:01:5D:49:19 (Raspberry Pi Trading)

Nmap done: 1 IP address (1 host up) scanned in 0.37 seconds
```

We can see why the two options `-localhost` and `-nolisten tcp` are so important. Basically, there are now three ways to attack the pi:

1. by connecting to the SSH server (low risk, well protected using key pairs).
2. by connecting to the VNC server and trying passwords over and over.
3. by connecting to the X server and attempting exploits and so on.

11 Appendix - Useful Manual Pages

`ssh` - SSH client

`ssh_config` - SSH client configuration

`sshd_config` - SSH server configuration

VNC server

Xvnc program (for -localhost)

Xserver program (for -nolisten tcp)