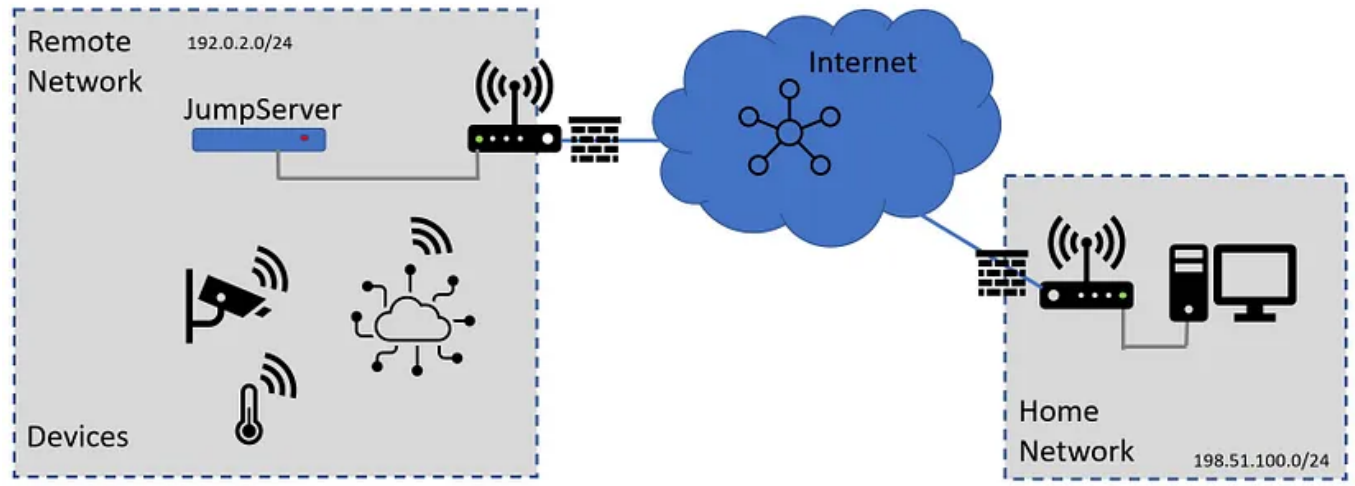
Secure Remote access for IoT on a Raspberry PI

Firstly why would we want to make a jumpserver (also known as a jumphost or jumpbox) to enable access our network. This follows a fairly basic principle of security that we want to allow a very limited entrance to the secured area to minimize the threat surface. In my case I am running a number of different IoT devices that are remote and I want the ability to check and maintain them securely. The IoT devices are behind a typical router which prevents all access from the internet, but that leaves the possibility that they cannot be easily updated or re-configured unless I am on the local network. Yes, some of these tunnel out and have cloud utilities for updating and configuring them, but this does not give me access to everything or the underlying host. Similar principles drive the use of jumpservers in commercial and cloud settings.



Remote access via a jumpserver

Of course, there are alternates but they are either less secure or more complex. For example, you could configure your router with a DMZ that forwards all external connection requests to a certain host or set the router up to allow for the various ports to be forwarded directly to devices inside the network. However, this requires a fairly static set of devices and ports and there is a reasonable probability of making a mistake in the configurations. The security of the individual devices may not be up to the challenge of unfettered remote access. We could address this with a VPN, but that tends to be a complex setup to enable both ends to communicate securely and reliably. Thankfully we have a technology called SSH that is well suited to this role of secure remote access. With a jumpserver we can forward one port from outside the network to one place inside the network that is running a full set of security tools. Once a user is properly authenticated onto the jumpserver they can then access the devices inside the network in the same way a local user would. By having a single configuration allowing access we dramatically reduce the threat surface and possibility of making a mistake.

In this case we are going to use a Raspberry Pi running various types of security software as our jumpserver. We are only going to allow connections from SSH to this server and we will load various tools to restrict the access.

Raspberry Pi as a jumpserver

1) Initial installation of the server

Here I am using the latest **Raspbian OS** deployed using the organization's own SD card imager, although this is applicable to any Linux distribution that suits the hardware.

Note that we could use a 'lite' version without a desktop as the unit will be running headless but in this case I am doing a full install with the GUI . The reason for this is that I want to be able to go to a virtual desktop using ssh forwarding and then open a browser on a local device from inside the network. This helps for usability as not all devices in the network that I am accessing support SSH and there are local statistics pages that are nicely displayed in a browser that are not available remotely. For example I am doing ADSB flight tracking and the signal level pages are only available in a local page.

2) Some basic configuration of the software

To make the unit headless we need to enable SSH and for security reasons this is off by default. So we create a blank file called SSH with no file extension and put this in the boot directory of the flash. I also want to enable networking via Wi-Fi so I can work on the server easily during the next few stages. In the case of the rpi we create 'wpa_supplicant.conf' loaded with our Wi-Fi configurations which we then place in the boot area of the CD card. Both can be done before the first boot by re-inserting the flash to the host after writing the image.

This config file is quite flexible and we can add several configurations and a network priority if we have multiple networks, for example to pre-configure the Wi-Fi for the remote network.

```
ctrl_interface=DIR=/var/run/wpa_supplicant
GROUP=netdevcountry=CountryCode
update_config=1

network={
ssid="5GHzSSID"
psk="5GHzpassword"
priority=5
id_str="Home5GHz"
}

network={
ssid="2GHzSSID"
psk="2GHzpassword"
priority=1
id_str="Remote2GHz"
}
# Remote Wi-Fi network details
```

3) Now lets boot the device and connect to it so we can configure it as a jumpserver

First we need to find the device. The easiest way is to check the dhcp server of our router for a new device. If this jumpserver is being used on the network where it is being configured you may want to bind the IP address and MAC so it has a fixed address. Alternately a network utility like nmap can be used to discover it from another Linux server.

```
sudo apt-get install nmap
sudo nmap <networkaddress>
# eg. sudo nmap 198.51.100.* to map the entire subnet
```

Personally I use either Tera Term or Powershell to connect to the device, but any SSH client can be used.

4) Setup some basic software security

Immediately on first login we should change the pi password to something more secure. I will stress here that the password needs to be suited for unattended access. So let's use something pseudorandom generated by a password manager, of at least 16 characters containing lower and upper case letters, numbers and special characters. I use **Lastpass** for this although there are many good alternatives.

We'll also create some new users, an admin user and a regular user with different passwords. I don't recommend removing the pi user at this point as it is a member of many groups including the ones needs to access the gpio, I2C. Let's get everything running then we can consider removing pi.

```
sudo adduser <adminuser>
sudo adduser <regularuser>
```

While we are at it we will change the hostname to something meaningful so when we have multiple terminal windows open we know where in the network we are located. The easiest method being to use raspi-config although we could also edit the hostname and hosts files in /etc.

```
sudo raspi-config
```

And finally lets do a full update as even with a fairly fresh install there are likely a lot of updates required to the downloaded and flashed image. And reboot everything at the end of the process.

```
sudo apt update
sudo apt full-upgrade
```

5) Let's check the setup and change some privileges

We want to add our new admin user to the sudo group so they can exercise privileged commands. Doing things this way will mean the admin user needs to submit a password to run sudo commands.

```
sudo usermod -aG sudo <adminuser>
```

Now we will check the status and see who is in the sudo group and their rights

```
getent group sudo
groups
sudo -l -U <adminuser>
```

It is usually the default that root cannot login via SSH but I like to check this.

```
sudo more /etc/ssh/sshd_config
```

should contain the commented out line

```
#PermitRootLogin prohibit-password
```

At this point we could add the new admin user to all the same groups as pi and then after checking everything, remove the pi user.

6) Now we will add some extra security in the form of a firewall

We could do something complicated here but instead as we only need a simple rule, we are going to use the ufw utility that makes adjustments to the in-built linux iptables firewall.

```
sudo apt-get install ufw
```

This is the one time we need to connect a keyboard and screen as we need to enable ufw, then set it to allow SSH in iptables. You can't set the rules without enabling ufw, which would automatically drop your current session.

```
ufw enable
sudo ufw allow ssh
```

and if we check the status we should see

```
sudo ufw status

Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
```

Of course we could add some other restrictions such as only allowing connections from the WAN router, disabling ping response etc. but for now this should suffice.

7) Now we need to secure the desktop

Here we find a security hole, in that the UI is set to autologin by default in raspbian. Which means if a malicious actor could physically connect to your jumpserver with a screen and keyboard they would find themselves on the GUI. While they can't launch privileged commands that need sudo this is still a big security gap. We can disable this on the desktop or at the command line.

First we will backup the config file for the GUI.

```
sudo cp  /etc/lightdm/lightdm.conf /etc/lightdm/lightdm.conf.bak
```

Then edit it to comment out the line that enables the autologin.

```
#autologin-user=pi
```

Alternately we could change this using raspi-config or in the desktop preferences. While we are there on the desktop we can set the default headless GUI resolution, as this will come in handy when later when we remote into to the Pi desktop. We could also make some changes to hide the valid usernames available on the login screen, but as we are running headless we won't bother with that now as the desktop will only be accessible after we are able to securely connect on SSH. However, just in case we forgot to logout from the display, we should install a screen saver so that the screen times out and a new login is required. And at this point we can enable VNC for the remote desktop.

8) Add Fail2ban

Next we are going to add some extra security, using the utility **fail2ban**. This utility monitors the access logs and restricts the number of login attempts that a user can make in a period which restricts the ability of bots to attack your systems. In case you are wondering, I see multiple bot attempts a week trying to access SSH on my jumpservers.

There is a great tutorial **here** on fail2ban. We can use the default configuration, or make specific rules with increased strictness that suit our purpose. The default has a 10 minute ban which is sufficient to deter attacks while allowing you to get back into the system if you mistype.

```
sudo apt install fail2ban
```

```
sudo service fail2ban restart
```

I prefer to leave the main jail.conf file as-is and create my own jail.local where I place my configurations that override the defaults.

These are found in

```
/etc/fail2ban/
```

and the logs are found here

```
sudo more /var/log/fail2ban.log
```

9) Add 2 Factor Authentication

For the next step we are going to enable 2 Factor Authentication (2FA). This means even with our strong passwords, another authentication method is required before anyone can login. Why is this last? Well we did a lot of configuring and rebooting up to now and this step will make things very slow.

There is an excellent tutorial **here** on the topic so I won't try reproduce this whole area, but the steps are:

· Enable SSH to use challenge response passwords

· Install the 2FA

· Configure the 2FA.

· Enable 2FA for SSH using pluggable authentication modules

```
sudo cp  /etc/ssh/sshd_config /etc/ssh/sshd_config.bak
sudo nano /etc/ssh/sshd_config
```

Change ChallengeResponseAuthentication from the default no to yes. Restart the sshd and make sure you can login from another terminal

```
sudo systemctl restart ssh
```

I like to use google authenticator for my 2FA and I usually enable the authenticator on 2 mobile devices so that I have a backup. For best security enable time based codes and disallow multiple attempts of the same code, don't increase the timeskew and allow for rate limiting.

```
sudo apt install libpam-google-authenticator
google-authenticator
```

Remember to note the emergency codes. The can be found in the hidden file ./.google_authenticator if you forgot to take a note.

Note: you need to do this step of creating a code for each user as once you enable 2FA it is enabled for all users.

Finally we enable the 2FA. I recommend staying logged in as one user, and checking the operation from another user as if it is incorrect it is not easy to login.

```
sudo cp /etc/pam.d/sshd /etc/pam.d/sshd.bak
sudo nano /etc/pam.d/sshd
```

To enable add the following after the @include common-auth statement

```
# 2FA
auth required pam_google_authenticator.so
```

And restart sshd. Then make sure to check that you can login from another terminal window.

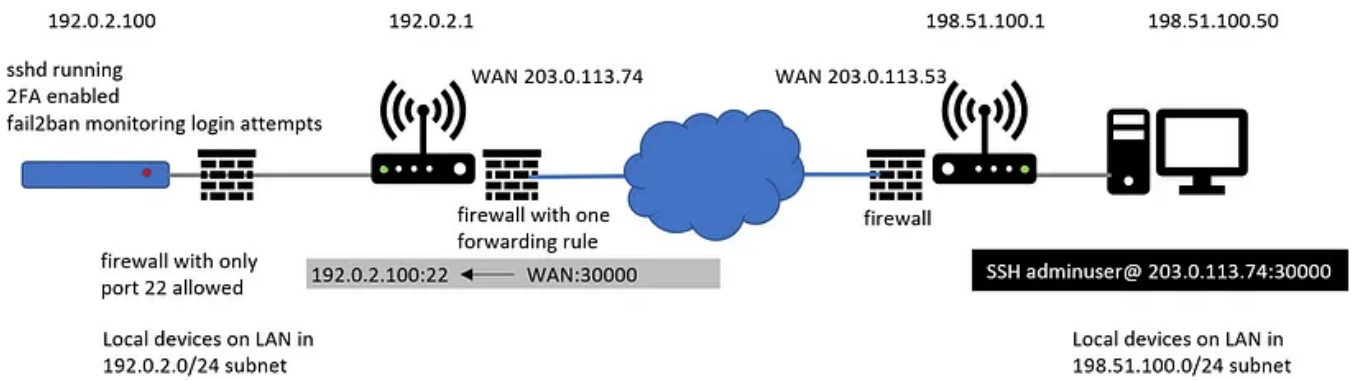
Of course we could have used certificates to authenticate the end user, but that adds another layer of complexity that requires us to use only certain devices to connect to the jumpserver and I want to be able to access it from my tablet and desktop. So for convenience I am not using certificates, although for higher security areas I would use them.

It is worth noting that security always needs to take into account usability as otherwise the end user will bypass it. A classic area was the use of strict password rotation that led to users reusing the same short password with an incrementing number at the end.

10) Configure the router to connect to the jumpserver

And now we can enable access. As we stated at the start the idea is to restrict the attack surface as much as possible. So we are firstly going to configure dhcp on our router to bind an IP address to the mac address of the jumpserver. That way it always gets the same IP address. It is preferable to configure dhcp to do this rather than give a static address as then there is no possibility of a conflict where another devices gets the same address.

Then we will take a single port from the WAN and forward it to the SSH port on the jumpserver. When doing this it is best to pick a high port number that is not used by any other services. This also constrains the possible attackers to those bothering to do a full port scan rather than those scanning the first 1000 ports. Note that I did not bother changing the port on the jumpserver from the default 22 as the server is inside the network and the SSH port can only be reached from the port forwarded by the router.



Of course we could make this even more restrictive and use a smaller subnet for the jumpserver and vlans to separate out the individual networks. We could also restrict the ability of the server to access the internet from inside the local network, And we could load an IDS.

So there we have a fully configured jumpserver with a minimal attack surface as it is only accessible via SSH with 2FA. We can now access devices on the local network from a remote location. In future articles I'll look at other things that can be done to secure this network.

If you liked this and found it useful, please follow me as it will encourage me to continue writing further articles. And feel free to subscribe.

https://davewpark.medium.com/subscribe

Security Raspberry Pi IoT Remote Linux

Follow

Written by Dave Park