Creating a Public Web Server on Raspberry Pi

SUNDAY, NOVEMBER 18, 2012

Foreword

I bought a Raspberry Pi a few months ago with the intent to have a toy web server at home. It's cheap (~$30), low-power (~3W), and doesn't need a cooling fan, so it's the ideal toy server that I won't feel bad about running 24/7. It has an ARM processor instead of the more common x86 processor, so you can only install certain OSes on it. Windows, Mac OSX, or most distributions of GNU Linux are normally compiled to create binaries that execute on x86 processors, so these can't be used. The Linux kernel *can* be compiled to create ARM binaries, so certain Linux distributions are compatible with this small computer. The OS that was custom-made for this small computer is called Raspbian, which is what I am using, but you can also install other OSes, including Android, Fedora, and XBMC.

I don't have an HDMI computer display, nor an HDMI adapter, so I will be setting up my web server by sending terminal commands via SSH. Some people may be off-put by the terminal, but I'll try to stay organized so we don't get lost.

I spent a lot of time reading about the details of networking, remote administration, and web server setup and best practices. This project is certainly the entrance to a rabbit hole of other interesting details you don't normally think about on a daily basis. Even though I spent a terrible amount of time, I gained a great deal of satisfaction.

Prerequisites

I performed a bit of setup before starting this project, so you may need to:

- Download and install the Raspbian OS onto an SD card
- Play around a bit on the OS, read the provided introductory Linux documentation

- Install some packages you like using `apt-get`, such as Git
- Connect power to the Raspberry Pi and connect it to your home network
- Connect your laptop via ethernet to same network as Raspberry Pi

Before we can begin, we must connect our Raspbery Pi to our network, as stated in the prerequisites.

Our first step is to ensure we can interface with our Raspberry Pi's OS. I will not be using a keyboard, mouse, or monitor directly connected to the device, instead I will be interfacing with it by sending terminal commands to it by using SSH. The device declares a default hostname of *raspberrypi*, which the local network can use to uniquely identify your device. This is very convenient because we don't have to find the IP address that the router assigned to it, we only have to request `raspberrypi` from the network to send it requests. If you want to change the device's hostname, there are ways to this.

- Connect to your Raspberry Pi with SSH
    - From your laptop, open a terminal and enter `ssh pi@raspberrypi`, where `pi` is the name of the user account you want to use.
    - Enter your password, which is `raspberry` by default.
    - Your current directory is the `pi` user's home directory. Any subsequent commands will be executed on the Raspberry Pi.
    - Type `exit` to quit your SSH session.

Install a Web Server

There are a number of web server applications out there, such as Apache, Nginx, lighttpd, and Jetty. Apache is by far the most popular web server, so you may want to try that, but I'll be using Nginx because I like to feel unique. Actually, I can think of a good reason: Rumors say that Nginx has a small memory footprint. This is a good match for a low-memory computer like the Raspberry Pi.

- Install the Nginx package from the default Raspbian repositories.

    - Just to verify we don't already have Nginx on this device, type `which nginx` into the SSH terminal. It should give us no result.
    - Update our `apt-get` sources by typing `sudo apt-get update`
    - Just to verify that Nginx is in the default repositories, type `apt-cache search nginx`. We should see several results, including the simply-named 'nginx' package.
    - Install the Nginx package by typing `sudo apt-get install nginx`.
- Validate that Nginx is installed
    - Type `service --status-all`. We should see an entry called 'nginx'.

Web Server Security

We just set up an application on our computer that we intend to welcome requests from the rest of the internet, which can be quite dangerous, so let's add some security. Like a good parent, we need to tell our web server to not talk to strangers. Because this is one of my first web servers, I will refer to the web server security guide on Linode for all security advice, mostly because it seems to be well-written.

Because some people make their careers as penetration specialists, I have always been curious about server protection best practices. Some of the protection I will set up may be redundant in a home server situation, but it is never redundant if new knowledge is gained! If I'm missing some important steps, please tell me - I would love to know!

Secure User Accounts

My Raspberry Pi had a default user named 'pi' which I have been using. One concern with web servers is that if a hacker gains control of a web request process, it can run under the same user permissions as the process owner. I want to ensure that Nginx request handling processes be owned by a limited-permissions user.

Before that, we have one more important change: Change the default password of the default user. We will later expose this server's SSH port to the public internet, and an stranger on the internet might try a set of default username/passwords.

- Change the default password of the default user
    - Open an SSH terminal into your Raspberry Pi
    - Type `passwd pi` to change the password for the device's default user account.
    - Enter the existing password, the new password twice, and hit enter to save the change.

Nginx is a master process that spawns worker processes to handle multiple web requests. The Apache community says that the 'www-data' user should handle web requests, so we'll do the same. While the Nginx master process runs as 'root' user, each worker process runs as the 'nobody' user by default. The worker process owner can be customized in the `/etc/nginx/nginx.conf` file.

- Ensure web request processes run as limited-permission user
    - Open an SSH terminal into your Raspberry Pi
    - Enter the following command to check the Nginx default user: `nano /etc/nginx/nginx.conf`.
    - Ensure the first line of this configuration file is `user www-data;`.
- Use SSH key pair authentication instead of passwords
    - On your laptop, open a terminal
    - Generate an SSH key pair by typing `ssh-keygen`

Set up a Firewall

We should also set up a firewall to protect our computer from port scanners and other malicious programs. A firewall is basically a set of rules that limits or blocks incoming or outgoing web requests. After a bit of research, it seems that a tool called iptables is the most popular solutions for this. It is also the solution proposed Linode's server security guide.

The Raspberry Pi firmware version that I have isn't compiled with iptables support, so you may have to upgrade your firmware first. Luckily, it was is simple as a few terminal commands if we use the rpi-update tool that a user named Hexxah has created.

- Upgrade your Raspberry Pi's kernel
    - Open an SSH terminal into your Raspberry Pi.
    - Get the convenient upgrade script by running `sudo wget http://goo.gl/1BOfJ -O /usr/bin/rpi-update && sudo chmod +x /usr/bin/rpi-update`.
    - You may need the ca-certificates package to make a request to GitHub, so run this `sudo apt-get install ca-certificates`.
    - Finally, to upgrade your Raspberry Pi's firmware, run `sudo rpi-update`. This will take ~5 minutes.

Now that our Raspberry Pi has the iptables program, let's set it up.

- Set up a firewall
    - Open an SSH terminal into your Raspberry Pi.
    - Check your default firewall rules by running `sudo iptables -L`.
    - Add firewall rules by creating a file by running `sudo nano /etc/iptables.firewall.rules`.
    - Copy and paste the basic rule set below into this file and save it:

/etc/iptables.firewall.rules

```
*filter

#  Allow all loopback (lo0) traffic and drop all traffic to 127/8 that doesn't use lo0
-A INPUT -i lo -j ACCEPT
-A INPUT -d 127.0.0.0/8 -j REJECT

#  Accept all established inbound connections
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

#  Allow all outbound traffic - you can modify this to only allow certain traffic
-A OUTPUT -j ACCEPT
```

```
#  Allow HTTP and HTTPS connections from anywhere (the normal ports for websites and SSL).
-A INPUT -p tcp --dport 80 -j ACCEPT
-A INPUT -p tcp --dport 443 -j ACCEPT
-A INPUT -p tcp --dport 8080 -j ACCEPT

#  Allow SSH connections
#
#  The -dport number should be the same port number you set in sshd_config
#
-A INPUT -p tcp -m state --state NEW --dport 22 -j ACCEPT

#  Allow ping
-A INPUT -p icmp -j ACCEPT

#  Log iptables denied calls
-A INPUT -m limit --limit 5/min -j LOG --log-prefix "iptables denied: " --log-level 7

#  Drop all other inbound - default deny unless explicitly allowed policy
-A INPUT -j DROP
-A FORWARD -j DROP

COMMIT
```

- Set up a firewall (continued)
    - Load the firewall rules by running `sudo iptables-restore < /etc/iptables.firewall.rules`.
    - Verify the rules have been loaded by running `sudo iptables -L`.
    - Now, to load these firewall rules every time the network adaptor is initialized, make a new file in the network adaptor hooks by running `sudo nano /etc/network/if-pre-up.d/firewall`.

    - Save the following text in this file:

      ```
      #!/bin/sh

      /sbin/iptables-restore < /etc/iptables.firewall.rules
      ```
    - Finally, make this script executable by running `sudo chmod +x /etc/network/if-pre-up.d/firewall`.

Defend Against Brute-force Attacks

One more thing we should be worried about is internet users attempting to access our SSH account by trying a dictionary attack against our password. There is a handy utility called Fail2Ban that monitors your log files for failed login attempts and temporarily blocks offending users.

- Install and configure Fail2Ban
    - Install the Fail2Ban packages by running `sudo apt-get install fail2ban`.
    - You can do different customization, but I just followed the recommendations of these code snippets to add Nginx monitoring to Fail2Ban.

Secure SSH

I told my firewall to allow traffic through port 22, which is SSH. This means anybody, not only I, can try to SSH into my Raspberry Pi. To better secure this, we have already took two good steps: 1) changed the password for the default user, and 2) protect from brute-force attacks. But I want to do one more thing.

- Restrict root for SSH
    - `sudo vi /etc/ssh/sshd_config`
    - Change the `PermitRootLogin` line like this:

      ```
      PermitRootLogin without-password
      ```

Update the Server's Software

A best practice for server admins is to ensure all server software is kept up-to-date. This is really easy in a Linux system like this, so there's no excuse. You should do this about once a month, or whenever you think of it.

- Update all installed packages
    - Open an SSH session with your Raspberry Pi.
    - Update your index of available packages and versions by running `sudo apt-get update`.
    - Update your OS's installed softare by running `sudo apt-get upgrade`. This took ~10 min for me.

Request a Page from Your Private Web Server

Now that we have done our due diligence by securing our web server, let's start up Nginx.

- Check the default Nginx config file by running `sudo nano /etc/nginx/sites-enabled/default`.
    - I am fine with the default setup. I just changed it to listen on port 8080.
- Start the server
    - Open an SSH terminal and run this on the Raspberry Pi: `sudo service nginx start`.
- Check the web server
    - The default static web directory is `/usr/share/nginx/www/`.
    - Open a browser on your laptop and navigate to `raspberrypi` in a web browser. You should see the default *index.html* file created by Nginx, which says something like "Welcome to Nginx!".

Request a Page from Your Public Web Server

The final hurdle is to make your Raspberry Pi accessible to the rest of the world. My router is not forwarding requests to the Raspberry Pi, so we will need to do some port forwarding. I added the DD-WRT firmware to my router quite a while ago, so you may need to find a more specific guide for adding port forwarding for your specific router.

Make Server Visible by Public IP Address

- Navigate to your router by IP. I enter 192.168.1.1 into my browser.
- Find the Port Forward settings
    - Add a new entry called *rpi-web*. *FromPort=8080*, *ToPort=8080*, *IpAddress=(Raspberry Pi internal Ip)*

Even with the port forwards, my web server still wasn't accessible by its public IP. With a friend's help, I tried putting it into my router's DMZ, which fixed the problem. It seems the purpose of a DMZ machine is to be the middle ground between your trusted/local network and the enemy/public network. This makes the DMZ machine almost wholly visible to the public internet. You may also want to try this on your router if you are having problems.

- Put your server in your router's DMZ.
    - I have DD-WRT firmware on my router, so your steps may be different.
    - Open the web UI for your router by navigating to its IP address in a web browser.
    - Go to the *NAT/QoS* tab, then the *DMZ* tab.
    - Set *Use DMZ* field to *Enable*.
    - Set the internal IP of your web server in the *DMZ Host IP Address* field.

We now have our Raspberry Pi visible to the rest of the world on ports 80, 443, and 8080. Congratulations. Try sending a request to your web server by its public IP. You can find your public IP by using an internet site like IpChicken. If your external IP address is *123.45.67.890*, then enter *123.45.67.890:8080* into your web browser to get the default Nginx *index.html* page.

Add Dynamic IP Solution: No-IP

My public IP address changes quite often, so it's impossible for me to SSH into my Raspberry Pi from work. To solve this, I wanted a domain name that points to my changing IP address. A technique called Dynamic DNS can help me. This involves

registering a domain name with a third-party and periodically updating my server's registered IP address with a simple app. The third-party I chose to use is called No-IP. The No-IP updater app is built-in to some routers, so you should check there first. My router has this capability, which is filed under a category called DDNS, but it isn't working. So, I want to try installing the app on my Raspberry Pi. Let's give it a try.

- Install the No-IP updater client
    - Download the package into a new directory and unarchive it

      ```
      mkdir ~/downloads
      wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz
      tar vzxf noip-duc-linux.tar.gz
      ```
    - Compile the source code

      ```
      cd noip*
      sudo make
      sudo make install
      ```

During compilation, you will be prompted for your noip.com username and password, as well as a refresh interval (I chose 30)

Conclusion

That was quite an educational project. I have a new appreciation for managed web hosts, because I believe they are responsible for managing the server's security, network visibility, kernel upgrades, etc.

What's next? I want to set up other stuff on this little server, such as GitLab, RedMine, and TiddlyWiki.