WEB DEVELOPMENT

SETTING UP A SECURE WEBSITE WITH HTTPS AND SPDY SUPPORT UNDER NGINX ON A RASPBERRY PI

SAT 24TH MAY, 2014

Matt Wilcox

This article is a follow-on from my Setting up a (reasonably) secure home web server with Raspberry Pi. This article won't show you the ins-and-outs of configuring a website on nginx, for that see the previous article. This one is about why and how you'd get the very latest nginx installed, and how to serve your site securely over HTTPS/SPDY.

Although this is specifically about the Rapberry Pi, the process ought to work without much modification on other Linux distributions. I've done this on my Debian server and didn't need to do anything different. Your mileage may vary.

Menu

The problem with the Raspbian version of nginx

It works and it's easy to install, but historically the versions of nginx that comes with Linux distributions are quite old – Raspbian in particular ships nginx 1.2.1 (from May 2012) even though the current version is 1.7.0 (from April 2014). Older versions don't support all of the features of the newer releases. For me, especially on a low-power device like the Pi which is potentially going to be used to host sensitive content, I want maximum efficiency and security. By running a more recent version of nginx I get to:

- Know that I'm benefiting from a lot of bug-fixes and other updates in nginx itself.
- Use the most recent and secure cyphers for the encryption process.
- Use SPDY instead of HTTP, for a faster browsing experience.
- Use 'OCSP Stapling', a new feature which helps reduce network round-trips & therefor speed up serving the website.

So, if you like me have decided you want to run a more recent version than what you can have via apt-get, here's a quick 'how to'…

Installing nginx from source files

To get an up-to-date nginx we will be installing nginx from source files: the files the authors of the program use and provide for anyone who wants them. That's different from the Raspbian supplied version, which comes as a 'binary' or 'executable' file, pre-built and pre-configured to work in a way that follows the conventions of the Raspbian distribution. When using source files, we have more work to do because we've got to perform the tasks the distribution maintainers usually do for us in their apt-get version (configuring it, building an executable file, tweaking services, etc).

We can cheat a bit though, and as weird as this sounds, we're starting with the apt-get version. There are a few things the apt-get process sets up for us that are a pain to do manually (or at least that I've not figured out how to do on my own). Things like getting the software to run automatically on boot (useful if your Pi powers down and then up again). So, we'll install nginx from apt-get and then immediately uninstall it. The uninstall process removes the application but leaves a few set-up things intact for us.

Existing nginx users

If you are currently running nginx on your Pi and you're wanting to 'upgrade', back up your nginx configuration files before you uninstall nginx. It might be easiest to just copy your existing /etc/nginx directory to your desktop (or wherever seems good to you), that way you can refer to your old files later on if you need to adjust the new nginx files to match some configuration you have on your old version:

```
sudo cp -r /etc/nginx ~/Desktop/apt-get-nginx
```

Those of you who aren't yet running nginx on your Pi

You need to install it:

```
sudo apt-get install nginx
```

Everyone

Lets uninstall the nginx you have:

```
sudo apt-get remove nginx
```

We're going to compile the newest version of OpenSSL into the newest version of nginx, so that we can choose to use SPDY/HTTPS and not have to rely on the system provided version of OpenSSL (which, like the system provided nginx, may be somewhat out-of-date). Essentially, we bake a copy of OpenSSL into our nginx so it'll all work on its own.

To do this we need to grab the source files for nginx, OpenSSL, and PCRE (PCRE is a pre-requisite of doing this). We then run 'make' (a Unix/Linux program) which builds the programs for us, and installs nginx.

To make life simpler, I've created a small shell script to do most of the work for you. You should be able to follow what it's doing from reading the comments in the code; the script is available as a Gist at Fetch, build, and install the latest nginx with the latest OpenSSL for RaspberryPi:

You should be able to get that onto your Pi by:

```
cd ~

wget
https://gist.githubusercontent.com/MattWilcox/402e2e8aa2e1c132ee24/raw/b88e2a85b31bcca57339c8ed8858f42557c3cf53/build_nginx.sh
```

(You don't have to type all that, just copy and paste it into your Terminal).

You will need to check the name of the most recent source files for nginx, PCRE, and OpenSSL yourself; and change the values under '# names of latest versions of each package' if they are now different! *Do not* put any spaces around the equals sign. They will break the script. You should only need to change the numbers.

Go to the following URLs and just check what the latest versions are:

- https://www.openssl.org/source/
- http://www.pcre.org/
- http://nginx.org/en/download.html

Make this new file executable by running:

```
sudo chmod +x build_nginx.sh
```

You can then run the script:

```
sudo ./build_nginx.sh
```

This will take quite a long time on the Pi – as in 20min or more. Be patient.

If you did not have nginx installed previously, you're now set to go, you just need to start nginx (see the next section). You can set up a website in the normal way – if you don't know 'the normal way', try following my Setting up a (reasonably) secure home web server with Raspberry Pi tutorial, but skip the setting up the Pi and installation of nginx parts.

If you had nginx installed previously everything should just work for you now. However it might not. If not it's likely that the new version of nginx might be reading your existing config files expecting some form of new option, or be seeing an option in your configuration files that's no longer valid. You could tinker about and google until you know what's different, or you could 'revert' your /etc/nginx directory to the default contents nginx last compiled with, and then manually re-configure those while referencing your old files. To do that:

```
sudo mv /etc/nginx /etc/nginx-broken
sudo mv /etc/nginx-default /etc/nginx
```

Refer to the files in your nginx-broken directory and amend the ones in /etc/nginx.

Starting nginx

```
sudo service nginx start
```

You could also use stop, reload, or restart. You'll want to use reload and/or restart as you play about with configuring nginx. I'm not going into any further detail on setting up a basic website here – try my older article if you need to get that information.

If all you wanted was a recent nginx you're all done. If you're interested in being able to use SPDY/HTTPS, that's coming up next…

Serving a site via HTTPS/SPDY

SPDY is a protocol created by Google, which is intended to replace HTTPS (and will provide the foundation for HTTP2). HTTPS and SPDY are both encrypted methods of communication, designed to ensure that people can't eavesdrop on the back-and-forth messages between your server and someone's browser. That's why online shops use them. They're also intended to give some level of assurance that the server is actually owned by the person/company it claims to be.

This works because browsers (IE, Firefox, Chrome, etc) have a list of trusted 'Certificate Authorities' – companies that sell SSL Certificates. If a website presents an SSL Certificate which has been 'signed' by one of those trusted companies the browser checks with the issuing Certificate Authority to see it the Certificate is still valid, and all being well the browser trusts the certificate and lets the user onto the site.

The Certificate Authorities are charged with checking that the SSL certificate buyer is who they say they are and does in fact control the domain they say they do. Because of that, getting a website using HTTPS/SPDY up and running is more involved than using regular HTTP. Depending on what you want, it may cost you money, and it may involve an identity check including a call from a real live person to your own phone number. There are a few options though…

Self Signed Certificate

If you're doing something on a private site with only yourself as the intended audience, you could use a 'self-signed certificate', which is where you act as your own Certificate Authority and 'sign' your own SSL certificate. Technically this will work, but that's no good for public sites; anyone connecting to your site will receive a full-screen warning that the SSL Certificate is untrusted (because you're not a trusted Certificate Authority browsers won't trust any certificate you sign). If you want to go down this route, follow this tutorial on how to create a self-signed certificate.

Basic SSL Certificate

If you're only looking to secure one domain for public access (i.e., your main domain such as mydomain.com) then you can get yourself a free 'Class One' certificate from https://startssl.com, who are a Certificate Authority that all modern browsers trust. These free certificates are valid for one year, and are renewable. Class One certificates are intended to 'provide modest assurances and are meant to secure personal web sites, public forums or web mail'. 'Class One' isn't, as far as I'm aware, any official terminology – it's just something StartSSL use to distinguish the different SSL Certificate products they sell. Class One certificates don't go through as rigorous checks as the higher classes; only enough to ensure you own the domain you're securing.

A more complete SSL Certificate

If you're getting more serious and want a certificate that means people can trust you are who you say you are you'll need a Class Two certificate, which are 'digital certificates ideal for authentication, Business to Business and Business to Consumer transactions, protection of electronic mail and signing of object code and macros'. A Class Two certificate costs money because the signing authority (StartSSL) are required to check your identity, which requires human intervention – you'll have to provide high resolution copies of identity documents, and talk to people on a phone. Another nice thing about a Class 2 certificate is you can create a 'wildcard', and use it for any sub-domain you might have in addition to your main domain.

Getting an SSL Certificate

I use and would recommend StartSSL as a cheap/free way to get a good SSL Certificate, though there are other Certificate Authorities that might suit your particular needs better. If you're going to use StartSSL for a free 1 year Class One certificate go to https://www.startssl.com/?app=12 and click 'sign up'. Just follow the process it guides you through. I think there's a fair difference in the process from company to company, and although I don't find StartSSL's website too clear to use, it works and it's free.

Configuring nginx to use your SSL Certificate

Once you have completed signing up for an SSL Certificate you should be in possession of three files;

- yourdomain.key
- yourdomain.pem
- ca-bundle.pem

The first two are ones you've made during the process and are for your domain, the third is the Certificate Authority's own certificate, which we'll use to enable OCSP Stapling. If you've used StartSSL that file is available from 'Toolbox' and 'StartCom CA Certificates' and is called 'Server Certificate Bundle with CRLs'.

Create a folder on your Pi to put these files into:

```
sudo mkdir /etc/nginx/my_ssl_certs
```

You need to get all of those files onto your pi and in that folder. If you've registered for the SSL Cert directly on your Pi you can just download them in the normal way. If you've been registering via a different machine and are using the Pi through Terminal only, you're going to need to download the files onto the machine you've been using for registering and then use scp to move them from that machine onto your pi, e.g., On the machine you've downloaded the files onto…

```
scp yourdomain.key yourdomain.pem ca-bundle.pem pi@IPOFYOURPI:~/
```

Replace 'IPOFYOURPI' with the IP address to your Pi and if your pi isn't running as the user 'pi' change that too.

Now, ssh onto your Pi and move those files into the directory we want. We didn't copy them directly there as the pi user shouldn't have permission to write files into /etc/nginx.

On your Pi:

```
sudo mv ~/yourdomain.key ~/yourdomain.pem ~/ca-bundle.pem
/etc/nginx/my_ssl_certs
```

Last thing, we need to create one more file for use when Perfect Forward Secrecy algorithms are used (which we absolutely want).

Again, on the pi, and inside the /etc/nginx/my_ssl_certs directory;

```
sudo openssl dhparam -rand - 4096 >> dhparam.pem
```

When you run this, the program means what it says; it's going to take a long time, almost 24hrs for a 4096bit value. If you want, change that to 2048 instead and it'll lop the time off considerably (but be less secure). Now might be a good time to read a few chapters of a decent book (I can recommend anything by Brandon Sanderson, if you're interested). Or you could go learn why big prime numbers make things secure (that's what this is doing; making a huge prime number).

With all files now on your pi and in the /etc/nginx/my_ssl_certs directory you can use them to secure your site!

In the nginx file that defines your site (/etc/nginx/sites-available/yoursite) you'll need to set the following (adjusting to your needs, this is a basic set-up which forbids HTTP and enforces HTTPS, using SPDY if SPDY is available on the browser that's trying to connect):

```
server {
# REDIRECT HTTP TO HTTPS
        server_name yourdomain.com;
        add_header  Strict-Transport-Security max-age=15768000;
        return 301  https://yourdomain.com$request_uri;
}

server {
# SET THE SITE UP FOR SSL

        listen 443 ssl spdy;

        # Get SSL setup
        ssl                 on;
        ssl_certificate
/etc/nginx/my_ssl_certs/yourdomain.pem;
        ssl_certificate_key
/etc/nginx/my_ssl_certs/yourdomain.key;
        ssl_dhparam         /etc/nginx/my_ssl_certs/dhparam.pem;
        ssl_session_timeout 5m;
        ssl_protocols       SSLv3 TLSv1 TLSv1.1 TLSv1.2;
```

```
        ssl_ciphers         'ECDHE-RSA-AES128-GCM-SHA256:ECDHE-
ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128-GCM-
SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-
AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-
SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-
SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-
SHA:DHE-RSA-AES256-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:ECDHE-
RSA-RC4-SHA:ECDHE-ECDSA-RC4-SHA:AES128:AES256:RC4-
SHA:HIGH:!aNULL:!eNULL:!EXPORT:!DES:!3DES:!MD5:!PSK';
        ssl_prefer_server_ciphers on;
        ssl_session_cache         shared:SSL:50m;

        # Enable HSTS ( see
http://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security )
        add_header Strict-Transport-Security max-age=15768000;

        # OCSP Stapling
        ssl_stapling            on;
        ssl_stapling_verify     on;
        ssl_trusted_certificate /etc/nginx/my_ssl_certs/ca-
bundle.pem;
        resolver                8.8.8.8;

### NOW CONFIGURE YOUR SITE BELOW, e.g,

        server_name yourdomain.com;
        root        /path/to/your/website/files;
        index       index.html index.htm;

        access_log /path/to/your/website/logs/access.log;
        error_log  /path/to/your/website/logs/error.log;

        error_page 404 /404.html;

        location / {
                try_files $uri $uri/;
        }
```

Be sure to replace the list of ssl_ciphers from this example with whatever the latest recommended cipher-suite from Mozilla is – choosing good ciphers in a good order is complex; you can render your server far less secure by doing this bit wrong – so just grab what the experts have compiled and use that.

Restart nginx as follows:

```
sudo service nginx restart
```

Your site should now be serving over HTTPS. To test it, run your domain through https://www.ssllabs.com/ssltest/

References

This post is essentially a note for myself if I ever need to do this stuff again, as I hadn't found a single resource that had all of that info in one place. I've basically cobbled all of the above together after a lot of playing about on my own sites over the last few months, while referring to the wisdom of others:

- https://wiki.mozilla.org/Security/Server_Side_TLS
- http://elinux.org/RPi_Nginx_Webserver#Nginx_Installation_from_Source_Code
- https://jve.linuxwall.info/blog/index.php?post/2013/10/12/A-grade-SSL/TLS-with-Nginx-and-StartSSL