Create a Self-Signed Certificate for Nginx in 5 Minutes (https://www.humankode.com/ssl/create-a-selfsigned-certificate-for-nginx-in-5-minutes/)

How to Create a Self-Signed SSL/TLS Certificate for Nginx in 5 Minutes

Published by Carlo van Wyk on January 28, 2018

How to Create a Self-Signed SSL/TLS Certificate for Nginx

In this tutorial, I'm going to show you how you can create a self-signed SSL/TLS certificate and use it on Nginx in 5 minutes or less. I'm using Ubuntu for this tutorial, but if you're on Mac OSX you can follow along as the syntax and commands are nearly identical.

Why Create a Self-Signed Certificate?

Self-signed certificates are useful for local development where you want to simulate an HTTPS environment. Take note that self-signed certificates are not meant for production, but they are ideal for localhost development.

An Overview of Creating a Self-Signed Certificate

Before continuing, let's take a step back and look at the steps involved in generating a self-signed certificate for Nginx:

1. Generate a self-signed certificate using OpenSSL
2. Copy the certificate to the certificates folder on Ubuntu
3. Update the Nginx configuration file to load the certificate
4. Copy the certificate's public key to the CA trusted root database to prevent Google Chrome from showing the site as insecure

Additionally, I created a Youtube tutorial that shows how to create a self-signed certificate for Nginx.

Watch the Youtube Tutorial

Step 1: Generate a Self-Signed Certificate using OpenSSL

I'll use OpenSSL to generate the certificate on Ubuntu. OpenSSL is installed on Mac OSX by default and the commands are exactly the same.

OpenSSL will generate 2 files which consist of a private key and a public key. Even though most people refer to an SSL/TLS certificate in the singular sense, it is the combination of the private key and the public key that makes a certificate.

Before running the OpenSSL command to generate a self-signed certificate, I'm going to create a certificate configuration file that will specify the certificate bits and the Subject Alternative Names. The Subject Alt Names are required in Google Chrome 58 and later, and is used to match the domain name and the certificate. If the domain name is not listed in the certificate's Subject Alternative Names list, you'll get a *NET::ERR_CERT_COMMON_NAME_INVALID* error message.

Create the Certificate Configuration File

```
sudo nano localhost.conf
```

```
[req]
default_bits       = 2048
default_keyfile    = localhost.key
distinguished_name = req_distinguished_name
req_extensions     = req_ext
x509_extensions    = v3_ca

[req_distinguished_name]
countryName                 = Country Name (2 letter code)
countryName_default         = US
stateOrProvinceName         = State or Province Name (full name)
stateOrProvinceName_default = New York
localityName                = Locality Name (eg, city)
localityName_default        = Rochester
organizationName            = Organization Name (eg, company)
organizationName_default    = localhost
organizationalUnitName      = organizationalunit
organizationalUnitName_default = Development
commonName                  = Common Name (e.g. server FQDN or YOUR name)
commonName_default          = localhost
commonName_max              = 64

[req_ext]
subjectAltName = @alt_names

[v3_ca]
subjectAltName = @alt_names

[alt_names]
DNS.1   = localhost
DNS.2   = 127.0.0.1
```

Create the Certificate using OpenSSL

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout localhost.ke
```

Step 2: Copy the Certificate Key Pair to the Certificates folder on Ubuntu

Copy the public key to the /etc/ssl/certs directory

```
sudo cp localhost.crt /etc/ssl/certs/localhost.crt
```

Copy the private key to the /etc/ssl/private directory

```
sudo cp localhost.key /etc/ssl/private/localhost.key
```

Step 3: Update the Nginx Configuration File to Load the Certificate Key Pair

```
sudo nano /etc/nginx/sites-available/default
```

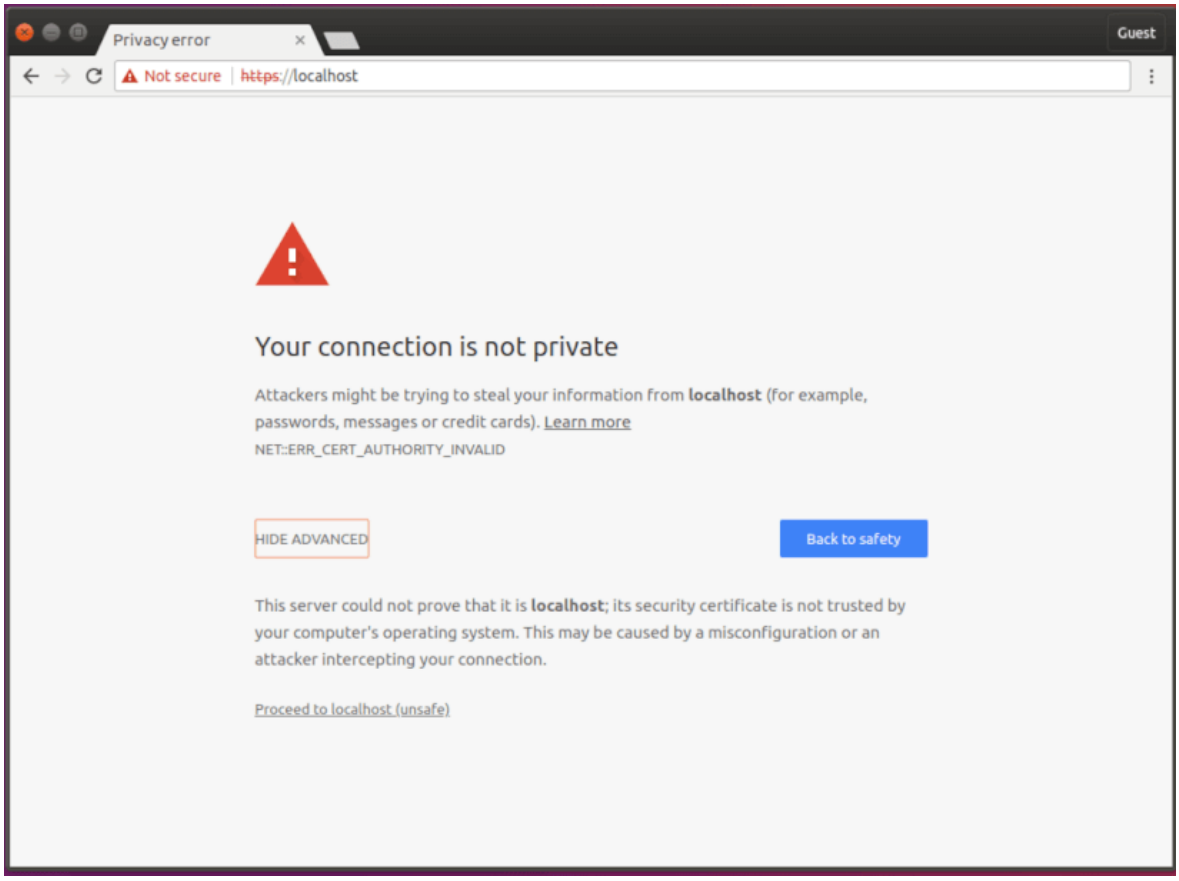
```
server {
        listen 80;
        listen 443 ssl http2;
        listen [::]:443 ssl http2;
        server_name localhost;

        ssl_certificate /etc/ssl/certs/localhost.crt;
        ssl_certificate_key /etc/ssl/private/localhost.key;

        ssl_protocols TLSv1.2 TLSv1.1 TLSv1;

        root /var/www/html;

        index index.html index.nginx-debian.html;
}
```

Reload the Nginx configuration changes

```
sudo service nginx reload
```

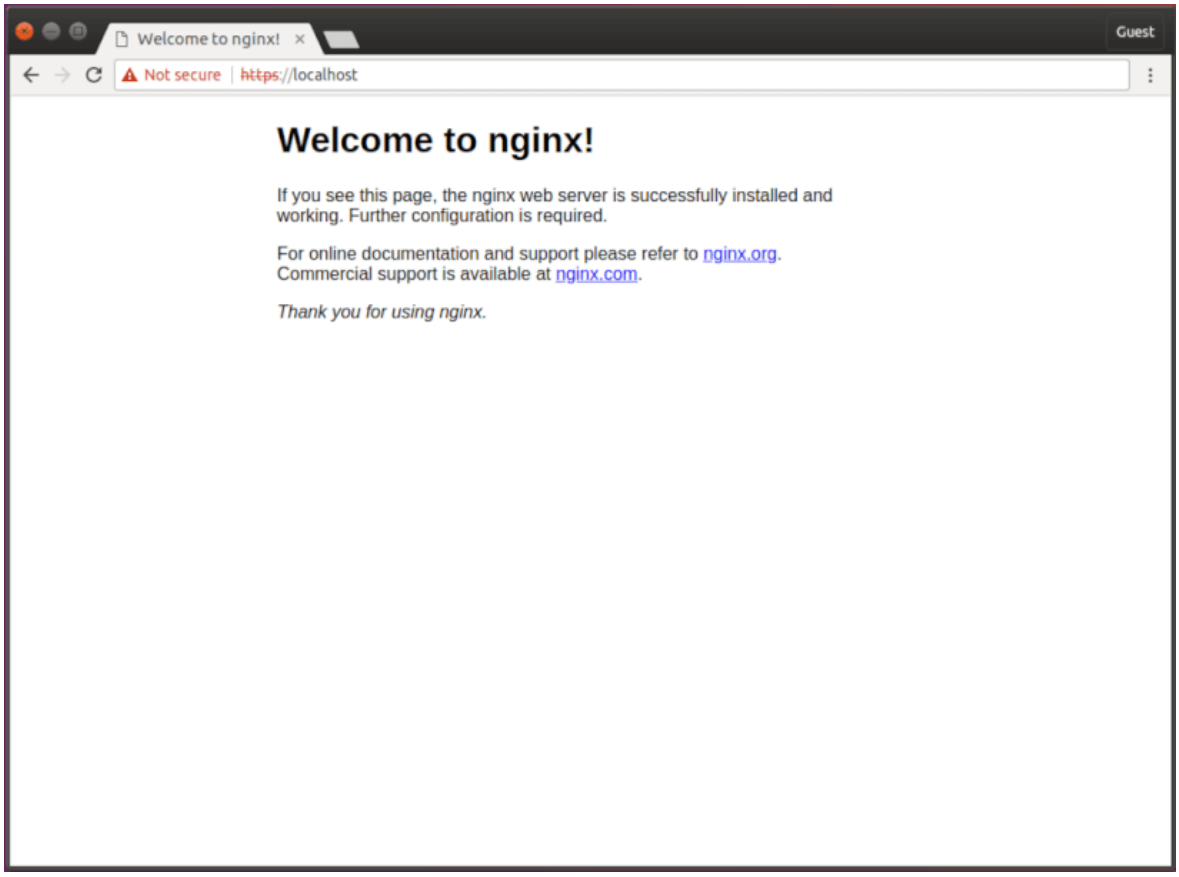
Open up the Google Chrome to Verify that Nginx Loads the Site Over HTTP and HTTPS

Since I haven't added the self-signed certificate to Chrome's CA Root store, Chrome shows the site as insecure.



Google Chrome Shows the Site as Not Secure

Click proceed to Localhost to verify that Nginx is correctly configured



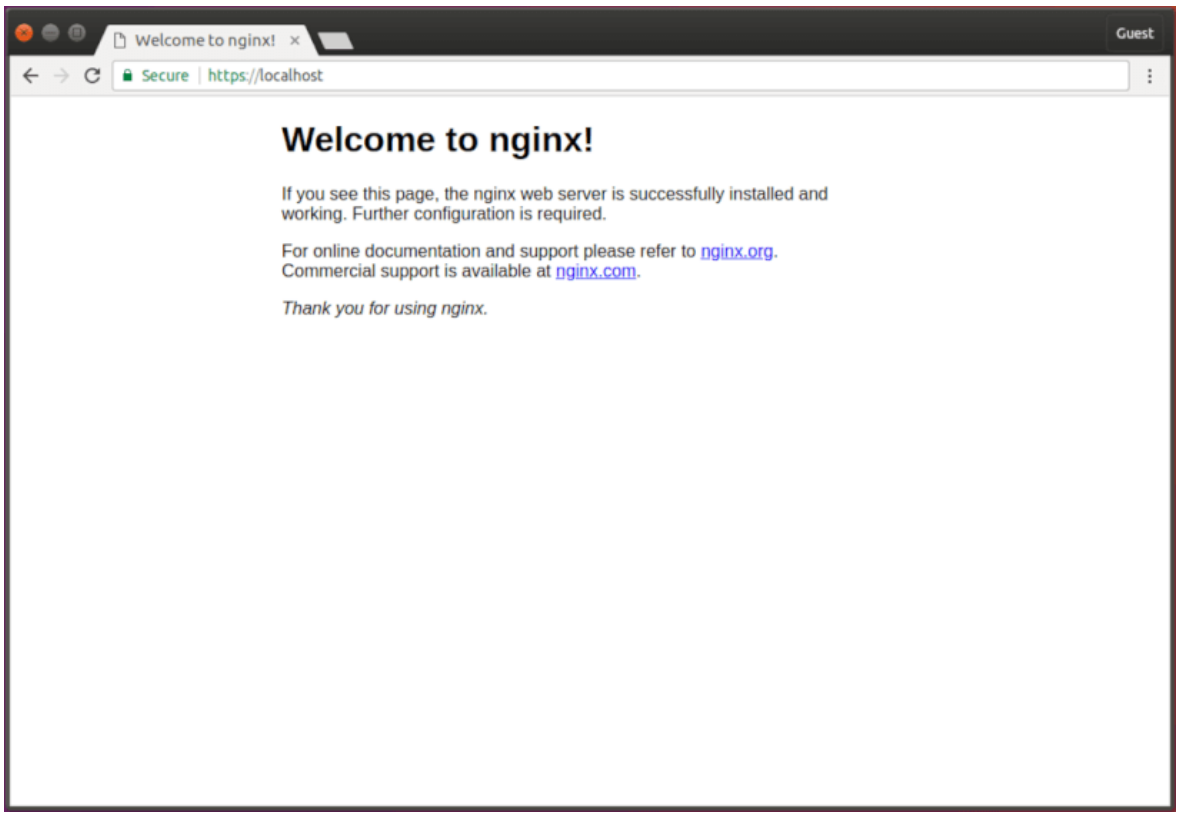
Nginx is Serving the Self-Signed Certificates, But Google Chrome is Showing the Site as Not Secure

Step 4: Configure Chrome to Trust the Certificate and to Show the Site as Secure

Add the certificate to the trusted CA root store

```
certutil -d sql:$HOME/.pki/nssdb -A -t "P,," -n "localhost" -i localhost.crt
```

Close all the Google Chrome windows and reopen. Chrome is now showing the site as secure.



Google Chrome Shows the Site as Secure

Related Posts

Develop Locally with HTTPS, Self-Signed Certificates and ASP.NET Core (https://www.humankode.com/asp-net-core/develop-locally-with-https-self-signed-certificates-and-asp-net-core/)

How to Set Up Free SSL Certificates from Let's Encrypt using Docker and Nginx (https://www.humankode.com/ssl/how-to-set-up-free-ssl-certificates-from-lets-encrypt-using-docker-and-nginx/)

connect

Youtube (https://www.youtube.com/channel/UC0pcYdOK_0k8WwoX77roJ8Q)

GitHub (https://www.github.com/thecarlo)

LinkedIn (https://www.linkedin.com/in/carlovan)