SSL Configuration with Openssl:

<https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-nginx-on-centos-7>

SSL Explanation

<https://rkakodker.medium.com/how-to-simple-way-of-generating-wildcard-san-ssl-csrs-for-product-managers-8c25d715d86f>

// Tutorial //

How To Create a Self-Signed SSL Certificate for Nginx on CentOS 7

Published on January 9, 2017 · Updated on April 1, 2022

Security

Nginx

CentOS

Default avatar

By Justin Ellingwood

How To Create a Self-Signed SSL Certificate for Nginx on CentOS 7

Not using CentOS 7?

Choose a different version or distribution.

CentOS 7

Introduction

TLS, or transport layer security, and its predecessor SSL, which stands for secure sockets layer, are web protocols used to wrap normal traffic in a protected, encrypted wrapper.

Using this technology, servers can send traffic safely between the server and clients without the possibility of the messages being intercepted by outside parties. The certificate system also assists users in verifying the identity of the sites that they are connecting with.

In this guide, you will set up a self-signed SSL certificate for use with an Nginx web server on a CentOS 7 server.

Note: A self-signed certificate will encrypt communication between your server and any clients. However, because it is not signed by any of the trusted certificate authorities included with web browsers, users cannot use the certificate to validate the identity of your server automatically. As a result, users will see a security error when visiting your site.

Because of this limitation, self-signed certificates are not appropriate for a production environment serving the public. They are typically used for testing, or for securing non-critical services used by a single user or a small group of users that can establish trust in the certificate’s validity through alternate communication channels.

For a more production-ready certificate solution, check out Let’s Encrypt, a free certificate authority. You can learn how to download and configure a Let’s Encrypt certificate in our setting up Nginx with a Let’s Encrypt certificate on CentOS 7 tutorial.

Prerequisites

To complete this tutorial, you should have the following:

A CentOS server with a non-root user configured with sudo privileges as described in the initial server setup for CentOS 7 tutorial.

Nginx installed on the server, as described in How to Install Nginx on CentOS 7.

When you are ready to get started, log into your server as your sudo user.

Step 1 — Create the SSL Certificate

TLS/SSL works by using a combination of a public certificate and a private key. The SSL key is kept secret on the server. It is used to encrypt content sent to clients. The SSL certificate is publicly shared with anyone requesting the content. It can be used to decrypt the content signed by the associated SSL key.

The /etc/ssl/certs directory, which can be used to hold the public certificate, should already exist on the server. You will need to create an /etc/ssl/private directory as well, to hold the private key file. Since the secrecy of this key is essential for security, it’s important to lock down the permissions to prevent unauthorized access:

sudo mkdir /etc/ssl/private

sudo chmod 700 /etc/ssl/private

Now, you can create a self-signed key and certificate pair with OpenSSL in a single command by typing:

sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/nginx-selfsigned.key -out /etc/ssl/certs/nginx-selfsigned.crt

You will be asked a series of questions. Before going over that, let’s take a look at what is happening in the command:

openssl: This is the basic command line tool for creating and managing OpenSSL certificates, keys, and other files.

req: This subcommand specifies that you want to use X.509 certificate signing request (CSR) management. The “X.509” is a public key infrastructure standard that SSL and TLS adheres to for its key and certificate management. You want to create a new X.509 cert, so you are using this subcommand.

-x509: This further modifies the previous subcommand by telling the utility that you want to make a self-signed certificate instead of generating a certificate signing request, as would normally happen.

-nodes: This tells OpenSSL to skip the option to secure your certificate with a passphrase. You need Nginx to be able to read the file, without user intervention, when the server starts up. A passphrase would prevent this from happening because you would have to enter it after every restart.

-days 365: This option sets the length of time that the certificate will be considered valid. You set it for one year here.

-newkey rsa:2048: This specifies that you want to generate a new certificate and a new key at the same time. You did not create the key that is required to sign the certificate in a previous step, so you need to create it along with the certificate. The rsa:2048 portion tells it to make an RSA key that is 2048 bits long.

-keyout: This line tells OpenSSL where to place the generated private key file that you are creating.

-out: This tells OpenSSL where to place the certificate that you are creating.

As stated above, these options will create both a key file and a certificate. You will be asked a few questions about your server in order to embed the information correctly in the certificate.

Fill out the prompts appropriately. The most important line is the one that requests the Common Name (e.g. server FQDN or YOUR name). You need to enter the domain name associated with your server or your server’s public IP address.

The entirety of the prompts will look something like this:

Output

Country Name (2 letter code) [XX]:US

State or Province Name (full name) []:Example

Locality Name (eg, city) [Default City]:Example

Organization Name (eg, company) [Default Company Ltd]:Example Inc

Organizational Unit Name (eg, section) []:Example Dept

Common Name (eg, your name or your server's hostname) []:your\_domain\_or\_ip

Email Address []:webmaster@example.com

Both of the files you created will be placed in the appropriate subdirectories of the /etc/ssl directory.

As you are using OpenSSL, you should also create a strong Diffie-Hellman group, which is used in negotiating Perfect Forward Secrecy with clients.

You can do this by typing:

sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048

This may take a few minutes, but when it’s done you will have a strong DH group at /etc/ssl/certs/dhparam.pem that you can use in your configuration.

Step 2 — Configure Nginx to Use SSL

The default Nginx configuration in CentOS is fairly unstructured, with the default HTTP server block living within the main configuration file. Nginx will check for files ending in .conf in the /etc/nginx/conf.d directory for additional configuration.

You will create a new file in this directory to configure a server block that serves content using the certificate files you generated. You can then optionally configure the default server block to redirect HTTP requests to HTTPS.

Create the TLS/SSL Server Block

Create and open a file called ssl.conf in the /etc/nginx/conf.d directory:

sudo vi /etc/nginx/conf.d/ssl.conf

Inside, begin by opening a server block. By default, TLS/SSL connections use port 443, so that should be your listen port. The server\_name should be set to the server’s domain name or IP address that you used as the Common Name when generating your certificate. Next, use the ssl\_certificate, ssl\_certificate\_key, and ssl\_dhparam directives to set the location of the SSL files you generated:

/etc/nginx/conf.d/ssl.conf

server {

listen 443 http2 ssl;

listen [::]:443 http2 ssl;

server\_name your\_server\_ip;

ssl\_certificate /etc/ssl/certs/nginx-selfsigned.crt;

ssl\_certificate\_key /etc/ssl/private/nginx-selfsigned.key;

ssl\_dhparam /etc/ssl/certs/dhparam.pem;

}

Next, you will add some additional SSL options that will increase your site’s security. The options you will use are recommendations from Cipherlist.eu. This site is designed to provide easy-to-consume encryption settings for popular software.

Note: The default suggested settings on Cipherlist.eu offer strong security. Sometimes, this comes at the cost of greater client compatibility. If you need to support older clients, there is an alternative list that can be accessed by clicking on the link labeled “Yes, give me a ciphersuite that works with legacy / old software.”

The compatibility list can be used instead of the default suggestions in the configuration above between the two comment blocks. The choice of which config you use will depend largely on what you need to support.

There are a few pieces of the configuration that you may wish to modify. First, you can add your preferred DNS resolver for upstream requests to the resolver directive. You used Google’s for this guide, but you can change this if you have other preferences.

Finally, you should take a moment to read up on HTTP Strict Transport Security, or HSTS, and specifically about the “preload” functionality. Preloading HSTS provides increased security, but can have far reaching consequences if accidentally enabled or enabled incorrectly. In this guide, you will not preload the settings, but you can modify that if you are sure you understand the implications.

/etc/nginx/conf.d/ssl.conf

server {

listen 443 http2 ssl;

listen [::]:443 http2 ssl;

server\_name your\_server\_ip;

ssl\_certificate /etc/ssl/certs/nginx-selfsigned.crt;

ssl\_certificate\_key /etc/ssl/private/nginx-selfsigned.key;

ssl\_dhparam /etc/ssl/certs/dhparam.pem;

########################################################################

# from https://cipherlist.eu/ #

########################################################################

ssl\_protocols TLSv1.3;# Requires nginx >= 1.13.0 else use TLSv1.2

ssl\_prefer\_server\_ciphers on;

ssl\_ciphers EECDH+AESGCM:EDH+AESGCM;

ssl\_ecdh\_curve secp384r1; # Requires nginx >= 1.1.0

ssl\_session\_timeout 10m;

ssl\_session\_cache shared:SSL:10m;

ssl\_session\_tickets off; # Requires nginx >= 1.5.9

ssl\_stapling on; # Requires nginx >= 1.3.7

ssl\_stapling\_verify on; # Requires nginx => 1.3.7

resolver 8.8.8.8 8.8.4.4 valid=300s;

resolver\_timeout 5s;

# Disable preloading HSTS for now. You can use the commented out header line that includes

# the "preload" directive if you understand the implications.

#add\_header Strict-Transport-Security "max-age=63072000; includeSubDomains; preload";

add\_header X-Frame-Options DENY;

add\_header X-Content-Type-Options nosniff;

add\_header X-XSS-Protection "1; mode=block";

##################################

# END https://cipherlist.eu/ BLOCK #

##################################

}

Because you are using a self-signed certificate, the SSL stapling will not be used. Nginx will simply output a warning, disable stapling for your self-signed cert, and continue to operate correctly.

Finally, add the rest of the Nginx configuration for your site. This will differ depending on your needs. You will just copy some of the directives used in the default location block for your example, which will set the document root and some error pages:

/etc/nginx/conf.d/ssl.conf

server {

listen 443 http2 ssl;

listen [::]:443 http2 ssl;

server\_name your\_server\_ip;

ssl\_certificate /etc/ssl/certs/nginx-selfsigned.crt;

ssl\_certificate\_key /etc/ssl/private/nginx-selfsigned.key;

ssl\_dhparam /etc/ssl/certs/dhparam.pem;

########################################################################

# from https://cipherlist.eu/ #

########################################################################

. . .

##################################

# END https://cipherlist.eu/ BLOCK #

##################################

root /usr/share/nginx/html;

location / {

}

error\_page 404 /404.html;

location = /404.html {

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

}

}

When you are finished, save and exit. This configures Nginx to use your generated SSL certificate to encrypt traffic. The SSL options specified ensure that only the most secure protocols and ciphers will be used. Note that this example configuration simply serves the default Nginx page, so you may want to modify it to meet your needs.

Create a Redirect from HTTP to HTTPS (Optional)

With your current configuration, Nginx responds with encrypted content for requests on port 443, but responds with unencrypted content for requests on port 80. This means that your site offers encryption, but does not enforce its usage. This may be fine for some use cases, but it is usually better to require encryption. This is especially important when confidential data like passwords may be transferred between the browser and the server.

Thankfully, the default Nginx configuration file allows us to easily add directives to the default port 80 server block. You can do this by inserting this at the beginning of ssl.conf:

/etc/nginx/conf.d/ssl.conf

server {

listen 80;

listen [::]:80;

server\_name your\_server\_ip;

return 301 https://$host$request\_uri;

}

. . .

Save and close the file when you are finished. This configures the HTTP on port 80 (default) server block to redirect incoming requests to the HTTPS server block you configured.

Step 3 — Enable the Changes in Nginx

Now that you’ve made your changes, you can restart Nginx to implement your new configuration.

First, you should check to make sure that there are no syntax errors in the configuration files. You can do this by typing:

sudo nginx -t

If everything is successful, you will get a result that looks like this:

Output

nginx: [warn] "ssl\_stapling" ignored, issuer certificate not found

nginx: the configuration file /etc/nginx/nginx.conf syntax is ok

nginx: configuration file /etc/nginx/nginx.conf test is successful

Notice the warning in the beginning. As noted earlier, this particular setting throws a warning since a self-signed certificate can’t use SSL stapling. This is expected and your server can still encrypt connections correctly.

If your output matches the above, your configuration file has no syntax errors. You can safely restart Nginx to implement your changes:

sudo systemctl restart nginx

The Nginx process will be restarted, implementing the SSL settings you configured.

Step 4 — Test Encryption

Now, you’re ready to test your SSL server.

Open your web browser and type https:// followed by your server’s domain name or IP into the address bar:

https://server\_domain\_or\_IP

Because the certificate you created isn’t signed by one of your browser’s trusted certificate authorities, you will likely see a scary looking warning like the one below:

Nginx self-signed cert warning

This is expected and normal. You are only interested in the encryption aspect of your certificate, not the third party validation of your host’s authenticity. Click “ADVANCED” and then the link provided to proceed to your host anyway:

Nginx self-signed override

You should be taken to your site. If you look in the browser address bar, you will see some indication of partial security. This might be a lock with an “x” over it or a triangle with an exclamation point. In this case, this just means that the certificate cannot be validated. It is still encrypting your connection.

If you configured Nginx to redirect HTTP requests to HTTPS, you can also check whether the redirect functions correctly:

http://server\_domain\_or\_IP

If this results in the same icon, this means that your redirect worked correctly.

Conclusion

You have configured your Nginx server to use strong encryption for client connections. This will allow you to serve requests securely, and will prevent outside parties from reading your traffic.