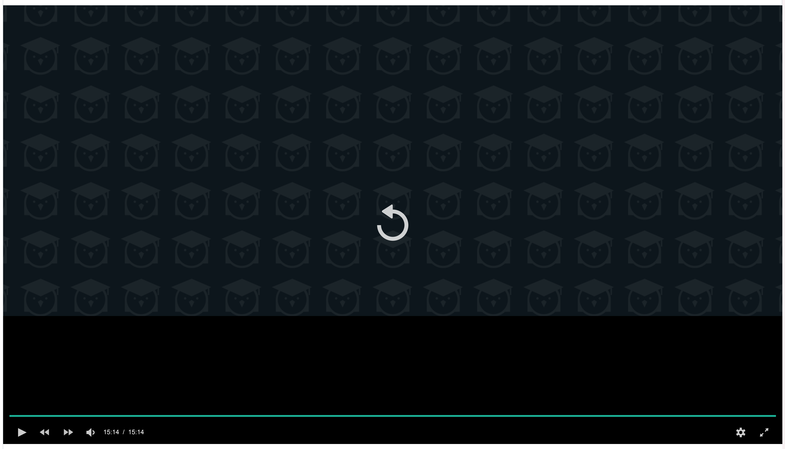
Lecture: Improving SSL Configuration



Now that we have an understanding of the main features of NGINX, we're going to dig a little deeper into some of the specific aspects of running NGINX. In this lesson, we'll dig deeper into improving the SSL setup for our server that is using a self-signed certificate.

Note: the commands in this video are run as the `root` user.

Documentation For This Video

- NGINX `ssl` module
- Mozilla's Server Side TLS Documentation

Improving SSL Configuration with SSL Session Caching

Up to this point, when we've worked with SSL in our servers, we've done the bare minimum by using `ssl_certificate` and `ssl_certificate_key`. There are a number of other settings that we can and should set to improve the overall configuration. These improvements are better for security, but can also improve our server performance by alleviating repetitive work.

To begin, let's add session caching so that fewer SSL handshakes need to be made. The initial handshake is the most taxing part of the process, and if we can avoid repeating it then we'll improve our overall performance. Let's make some changes to our `/etc/nginx/conf.d/default.conf` file to add SSL session caching.

/etc/nginx/conf.d/default.conf (partial)

```
    ssl_certificate /etc/nginx/ssl/public.pem;
    ssl_certificate_key /etc/nginx/ssl/private.key;

ssl_session_timeout 1d;
ssl_session_cache shared:SSL:50m;
ssl_session_tickets off;</code></pre>
```

These three new directives do the following:

- ssl_session_timeout – set's how long a session can persist
- ssl_session_cache – set's the number of sessions to cache. The shared value indicates that it can be used by all NGINX processes. One megabyte of space can hold about 4,000 sessions
- ssl_sessions_tickets – SSL Tickets are an idea where encrypted session information is stored on the client instead of the server. We're opting to not use tickets

Setting SSL Cipher and Protocols

Depending on the clients that will be interacting with a service, the configuration of the TLS protocols and ciphers can be important. For our purposes, we're expecting that our clients are using modern web browsers (at least IE 11, FireFox 27, Chrome 30, Safari 9, or Android 8). Knowing that we expect "modern" clients we'll set some values based on suggestions from Mozilla.

/etc/nginx/conf.d/default.conf (partial)

```
    ssl_certificate /etc/nginx/ssl/public.pem;
    ssl_certificate_key /etc/nginx/ssl/private.key;

ssl_session_timeout 1d;
ssl_session_cache shared:SSL:50m;
ssl_session_tickets off;

ssl_protocols TLSv1.2;
ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256';
ssl_prefer_server_ciphers on;</code></pre>
```

Note: You can read about these suggestions and more about backward compatibility here.

We know that all of our clients can work with TLS version 1.2 so we're going to require it using ssl_protocols. Skipping over the ciphers for a moment, we're setting ssl_prefer_server_ciphers so that we utilize the list of ciphers that we specify rather than using client ciphers. This configuration is so that our server is in control of the SSL/TLS experience, so we're keeping as much as we can under the server's umbrella. The ssl_ciphers list is quite long, the ordering is important, and we're not (at least I'm not) cryptography experts. This list is taken directly from Mozilla as recommended.

HSTS and OCSP Stapling

We only have a few more things to change. The first is adding support for HSTS. HSTS is a way to have NGINX tell the client that it should never interact with our domain using HTTP, and we add this using a header. If we were not ready for our domain/IP address to only receive HTTPS traffic then we should *NOT* add this header until we are:

/etc/nginx/conf.d/default.conf (partial)

```
    ssl_certificate /etc/nginx/ssl/public.pem;
    ssl_certificate_key /etc/nginx/ssl/private.key;

ssl_session_timeout 1d;
ssl_session_cache shared:SSL:50m;
ssl_session_tickets off;

ssl_protocols TLSv1.2;
ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256';
ssl_prefer_server_ciphers on;

# 15768000 is roughly 6 months
add_header Strict-Transport-Security max-age=15768000;</code></pre>
```

Now any browser that receives a response from our server will refuse to send traffic over HTTP to our domain for the duration of max-age.

Our last change relates to OCSP or "Online Certificate Status Protocol". Without OCSP stapling, the client will need to make a separate request to verify the validity of a certificate from the issuer, but with OSCP stapling, the server can make that request and cache it for awhile to return that information with the initial handshake. Let's add OCSP Stapling:

/etc/nginx/conf.d/default.conf (partial)

```
    ssl_certificate /etc/nginx/ssl/public.pem;
    ssl_certificate_key /etc/nginx/ssl/private.key;

ssl_session_timeout 1d;
ssl_session_cache shared:SSL:50m;
ssl_session_tickets off;

ssl_protocols TLSv1.2;
ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256';
ssl_prefer_server_ciphers on;

# 15768000 is roughly 6 months
add_header Strict-Transport-Security max-age=15768000;

# OCSP Stapling
ssl_stapling on;
ssl_stapling_verify on;</code></pre>
```

If we check our configuration using nginx -t, we'll see that we have a warning related to OCSP stapling:

```
nginx: [warn] "ssl_stapling" ignored, issuer certificate not found for certificate "/etc/nginx/ssl/public.pem"
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Since we're using a self-signed certificate, the stapling is ineffective. In a setting where we're using a certificate from an actual certificate authority, we would also want to include the following directive with a downloaded root CA certificate:

/etc/nginx/conf.d/default.conf (partial)

```
    ssl_trusted_certificate /path/to/ROOT_CA_CERT;
```

When we discuss using Let's Encrypt, we'll generate a valid and verifiable certificate. At that point, we'll be able to revisit OCSP stapling.