**Name of Title:** Learning Nginx

**Video Name:** 03\_05 Configuring SSL

**Estimated Length:**

**Author Name:** Michael Jenkins

**Chapter\_Section\_Video:**

**Video Objective:**

At the end of this video the learner will be able to create a self signed certificate and configure the site to apply them to all assets.

**Script:**

SLIDE: 03\_05 Configuring HTTPS

Hypertext transport protocol is the basic protocol that web servers use to fulfill requests. HTTPS is the the secure version of that protocol. HTTPS protects the information being sent between the server and the client by encrypting it. This is important for protecting sites that use sensitive information like credit cards or social security numbers.

SLIDE 1

SSL, or Secure Sockets Layer, is the name of the first method developed to encrypt web traffic. Because SSL can be reversed, using SSL has been deprecated and has fallen out of use.

SLIDE 2

Instead, HTTPS now uses TLS, or Transport Layer Security, as the method for encrypting traffic.

SLIDE 3

So even though TLS is used and favored over SSL, you’ll still hear people referring to SSL when they refer to certificates, keys, and so on.

In fact, Nginx provides the HTTP SSL Module for configuring sites for HTTPS.

So let’s create a self signed SSL certificate and then use the directives from the HTTP SSL module to server our site over HTTPS.

I already have our development VM booted up but before we go to the terminal, let’s take a look at our site in the Chrome browser.

OPEN BROWSER:

<http://192.168.0.3>

CLICK ON (i) in on left side of address bar

Since we’re not serving over HTTPS yet, most browsers will let us know by saying something like “This site is not secure.” We’ll take steps now to fix that.

***STOP***

# 03\_06 STARTS HERE - New exercise file folder needed

If your site will be accessible to the general public on the internet, you’ll want to get an SSL certificate from a known certificate authority like let’s encrypt, symantec, or digicert. That will allow your site to be identified and trusted.

For development and testing, we can generate our own SSL certificate. It won’t be trusted, but it will allow us to configure nginx with SSL.

So let’s connect to our development environment and become the root user.

BACK IN THE SHELL

CTRL+L

Sudo su -

The first thing we need to do is create an SSL certificate and key. We’ll be using the openSSL command line tool to do that. You can check for it by typing which openssl and installing it with apt install openssl if its missing.

CTRL+L

Which openssl

Apt install openssl

There are several methods for creating SSL certificates; We’re going to use the simplest one which creates a key and a certificate in one command:

CTRL+L

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

This command creates a key and then immediately uses that key to sign the certificate. We’re prompted for the specifics of the certificate like our physical address, email address, and the name of the site. Since we’re just working with a self signed certificate on a development site, using the defaults is fine. You would want to enter the details specific to your site though, if you are actually putting a site on the internet.

Also, if you want to remove the prompts altogether, you can use the dash batch switch:

CTRL+L

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

In any case, this command places the new certificate in /etc/ssl/certs and the key in /etc/ssl/private.

CTRL+L  
 ls -ltr /etc/ssl/certs/nginx.crt

ls -ltr /etc/ssl/private/nginx.key

The certificate can be public since our server will use it to identify itself over HTTPS. They key needs to stay private since that’s the file that will be used to encrypt the traffic. If the key is exposed, then someone could potentially decypher the traffic sent by the server.

**STOP**

# 03\_07 STARTS HERE - New Exercise File Needed

Now that we have our certificate and key, we need to configure nginx to use them. Let’s edit the site conf file.

CTRL+L

vim /etc/nginx/conf.d/wisdompetmed.local.conf

The first thing we’ll need to do is force all traffic to be served using HTTPS. We’ll do this by creating a new default server that listens on port 80.

server {

listen 80 default\_server;

}

We don’t need a root site for this server because what we’ll do instead is redirect all traffic to HTTPS with the return directive.

Return 301

So if any requests come to port 80, they’ll get a 301 code which will permanently redirect the browser to a different address. So we also need to provide the address. And for that, we’ll use HTTPS:// and the variables that hold the server address and the request URI.

return 301 [https://$server\_addr$request\_uri](about:blank);

This will effectively redirect any traffic that hits the site from port 80 to port 443 which is the port that HTTPS traffic is served over.

So now we need to update our old server block to listen on port 443. We also need to add the ssl directive to tell nginx to encrypt the traffic served on this port. Just because we configure nginx to listen on port 443, doesn’t configure SSL automatically:

listen 443 ssl default\_server;

And we also need to add the ssl\_certificate and ssl\_certificate\_key directives along with the paths to the certificate and key that we created earlier.

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

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

Now we can save the file and test the configuration before loading it:

SAVE FILE

Nginx -t

Systemctl reload nginx

Now let’s reload the page in the browser.

GO BACK TO BROWSER

RELOAD PAGE

SHOULD SEE “Your connection is not private” page

We’re seeing a “Your connection is not private” message from Chrome now. This is a good thing! It means our SSL certificate is in place and our site is serving HTTPS properly.

Chrome uses page to tell us that it can’t identify the site is actually who it says it is. Since we signed the certificate ourselves, we could be anyone. So chrome and other browsers will give this nice warning to let us know to keep an eye out for anything that might be suspicious. Other browsers like Firefox for example, even give you an option for storing exceptions to sites that we know are OK. In this case, we know its our own site, so we can proceed.

CLICK ADVANCED

CLICK PROCEED TO SITE…

And now we can check the address bar to see that our site is using HTTPS. Chrome still says our connection is not secure.

CLICK NOT SECURE

But the difference here is that we have a certificate in place now.

HOVER OVER “certificate”

And if we click here where it says certificate, we can see the certificate that Chrome has processed for our site.

CLICK DETAILS AND SCROLL DOWN TO ISSUER

We didn’t enter any specifics for this key so these values are just defaults from our Open SSL configuration.

If this certificate was actually issued by a certificate authority like Let’s Encrypt, Symantec, or Digicert, we would see their information here instead. And if that was the case, our site would be trusted by the browser and we would see a green lock icon in the address bar instead of this red triangle.

The good thing about our configuration now though, is that all content -- even the images -- are being sent over HTTPS. So now if we go to the appointments page:

<https://192.168.0.3/appointments/>

We get prompted for a password and the password is sent from our browser to the server over HTTPS. Don’t forget that our configuration for this location also uses allow and deny directives to keep requests limited to our network. So with SSL encryption in place, limiting access by IP address, and password protection, this page is probably the most secure page on the site.

This also demonstrates how all of these methods can be layered to improve the security of the sites we serve with nginx.

For more information on configuring SSL for nginx, take a look at the documentation for the HTTP SSL module on nginx.org.

EDITOR: FLY IN LINK TO <http://nginx.org/en/docs/http/ngx_http_ssl_module.html>

**Conclusion:**

Type your conclusion statement here.

**Script and Media:**

Break the script up into parts and align it with any media (slides, web, CLI, etc.)

| **Part** | **Script** | **Media** |
| --- | --- | --- |
|  |  |  |

**Exercise Files:**

**Basement:**

[**https://certsimple.com/blog/ssl-or-tls**](https://certsimple.com/blog/ssl-or-tls)

Why people who know better still say 'SSL'

HTTPS is a protocol used to provide security over the Internet. HTTPS guarantees that users are talking to the server they expect, and that nobody else can intercept or change the content they're seeing in transit.

<https://www.feistyduck.com/library/openssl-cookbook/>

This free book provides complete coverage of OpenSSL installation, configuration, and key and certificate management. The appendix includes *SSL/TLS Deployment Best Practices*, a concise guide to designing and deploying secure web sites and applications.

Generate the Key and Cert

openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout nginx-selfsigned.key -out nginx-selfsigned.crt -batch

<http://nginx.org/en/docs/http/ngx_http_ssl_module.html>

<http://nginx.org/en/docs/http/configuring_https_servers.html>

server {  
 listen 80 default\_server;  
 listen [::]:80 default\_server;  
 server\_name server\_domain\_or\_IP;  
 return 302 https://$server\_name$request\_uri;  
}  
  
server {  
  
 # SSL configuration  
  
 listen 443 ssl http2 default\_server;  
 listen [::]:443 ssl http2 default\_server;  
 Sslconfig ….

<https://en.wikipedia.org/wiki/Transport_Layer_Security>

**Transport Layer Security** (**TLS**) – and its predecessor, **Secure Sockets Layer** (**SSL**), which is now deprecated by the [Internet Engineering Task Force](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) [[1]](https://en.wikipedia.org/wiki/Transport_Layer_Security#cite_note-1) (IETF) – are [cryptographic protocols](https://en.wikipedia.org/wiki/Cryptographic_protocol) that provide [communications security](https://en.wikipedia.org/wiki/Communications_security) over a [computer network](https://en.wikipedia.org/wiki/Computer_network).

HTTPS makes sure that users are talking to the server they expect, and that no one else can change the content from the site while its in being served.

If you’re following along with the exercise files, you can use the Vagrantfile in the folder for this lesson. It will boot the VM and install nginx and all supporting software and data for our demo site.

If you're not using the exercise files, you can follow along with a VM running Ubuntu 18.04 LTS. You’ll need root access and you’ll need to install nginx.