Fast, Scalable And Secure Webhosting For Web Developers

Wim Bervoets
www.fastwebhostingsecrets.com

Fast, Scalable And Secure Web Hosting For Web Developers

Learn to set up your server and website

Wim Bervoets

This book is for sale at http://leanpub.com/fastscalableandsecurewebhostingforwebdevelopers

This version was published on 2019-07-29

Tweet This Book!

Please help Wim Bervoets by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought Fast, scalable and secure webhosting for Web Developers and it's great!

The suggested hashtag for this book is #fastwebhosting.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#fastwebhosting

Contents

HTTPS everywhere

In this chapter we'll first explain the reasons why it is a good idea to secure your site. Then we will show you the exact steps on how to make your site secure, starting from ordering a certificate to testing your https website.

Do you need a secure website?

You can make your site secure by running it over HTTPS, which stands for Hypertext Transfer Protocol Secure. Using https protects the integrity and confidentiality of your user's data.

Here are some reasons why this is important:

- HTTP is an insecure protocol, which means everything that is sent between the browser and your server is in plain text and readable by anyone tapping the internet connection. This could be a government agency (eg. NSA, …) or someone using the same free unencrypted free WIFI hotspot as your user.
- HTTPS on the other hand encrypts the communication between the browser and the server. As such nobody can listen to your users "conversations". A https certificate for a website also proves that users communicate with the intended website and not a fake website run by malicious people.
- Since the summer of 2014, Google has publicly said that having a https site can give a small ranking boost in the search engine results.

It is also vital that you secure all parts of your website. This includes all pages, all resources (images, javascript, css,), all resources hosted on a CDN, …

When you would only use https for eg. A login into a forum or a credit card detail information page, your website is still 'leaking' sensitive information hackers can use.

More in detail this could be a session identifier or cookies which are typically set after a login. The hacker could reuse this information to hijack the users session and being effectively logged in without knowing any password.

In October 2010 the Firesheep plugin for the Firefox browser was released which intercepted unencrypted cookies from Twitter and Facebook, forcing them to go https everywhere.

We also recommend to only offer an https version of your site and redirect any users accessing the http version to the https version. We'll explain how to do this technically in the next sections.

Buying a certificate for your site

When the users browser requests an HTTPS connection to a webpage on your server, the server needs to send back its TLS certificate. This initial exchange to setup a secure connection is called a TLS / SSL handshake.

The browser will do the necessary checks to see if the certificate is still valid, is for the correct site (eg. https://www.myexamplewebsite.com) and more.

To acquire a certificate, you'll need to buy one from a certificate authority. A certificate authority is a company which is trusted by the major browsers to issue valid certificates. Well known names include Comodo, VeriSign and more.

There are a few things you need to know about the different certificate types that exist before you can buy one.

Standard certificate

A standard certificate can be used for a single website domain. Eg. if all your content is hosted on www.mywebsite.com, you could buy a standard certificate which is valid for www.mywebsite.com. Note this doesn't include any subdomains which you may also use.

For example cdn.mywebsite.com is not included. Browsers will issue warnings to the user if you try to use a certificate which is not valid. You could by a second certificate for the subdomain cdn.mywebsite.com to solve this problem.

Wildcard certificate.

A wildcard certificate is still only valid for one top domain (eg. mywebsite.com), but it also supports all subdomains (*.mywebsite.com); hence the name wildcard certificate.

This kind of certificate is usually a little bit more expensive, then a standard certificate. Depending on the price and on the number of subdomains you're going to use you'll need to decide between a standard and wildcard certificate.

Other types of certificates exists (eg. Multidomain), but are usually pretty expensive; so we'll not cover them here.

Public and private key length

Certificates work with public and private keys to encrypt the communications running over https. Anything encrypted with the public key can only be decrypted by someone who has the private key. Anything encrypted with the private key can only be decrypted by the public key.

When a browser and a web server communicate with eachother, the private key needs to remain in a secure location on the web server. The public key is intended to be distributed to the browser, so it is able to decrypt the information which was encrypted with the private key.

To counter brute-force attacks that are trying to acquire the private key in use, the key needs to be big enough. In the past 1024 bit keys were generally created by the certificate authorities. Nowadays you should use 2048 bit keys, because 1024 bit keys have become to weak. (we'll guide you through the technical details later)

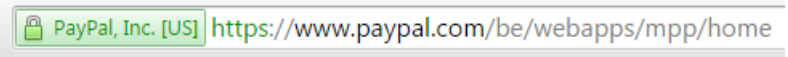
Extended Validation certificates and the green bar in the browsers

Earlier we said that a https certificate for a website proves that users communicate with the intended website and not a fake website run by malicious people.

Of course the Certificate Authority plays a vital role in this: when you order a certificate they should verify you're the owner of the domain name.

With a normal certificate the validation is quicker and less extensive then when you order an EV (Extended Validation) certificate. An EV certificate is more expensive due to the extended manual verification of the site owner.

Browsers do place a lot more trust in an EV certificate. They will display the name of the company inside a green bar in the browsers address bar. For example:



EV Certificate shows a green bar in the browser

An EV certificate could be interesting for an ecommerce site because it gives your user a greater trust in your site which could lead to more sales.

There are some restrictions with EV certificates though: only companies can order an EV certificates, individuals cannot. EV certificates are always for one domain only; there are no wildcard EV certificates at this moment.

Buying the certificate

You can buy a certificate from a Certificate Authority. You should choose a Certificate Authority which is trusted by both current and older browsers/operating systems for maximum compatibility. These include, but are not limited to:

- GlobalSign
- Network Solutions
- Symantec

- Thawte
- Trustwave
- Comodo

You can view daily updated reports of the market shares of the leading Certificate Authorities at http://w3techs.com/technologies/overview/ssl_certificate/all

Because of better pricing we have chosen to buy a certificate from Comodo. They also support generating 2048 bit certificates for better security.

Many companies resell certificates from the above Certificate Authorities. They are the exact same certificates, but come with a reduced price tag. We recommend you to shop around.

One such reseller we recommend and use is the SSLStore which we will use in the example ordering process below.

Generate a Certificate Signing request

When ordering a Certificate from a Certificate Authority you'll need to create a Certificate Signing request. (CSR)

A Certificate Signing request is file with encrypted text that is generated on the server where the certificate will be used on. It contains various details like your organization name, the common name (=domain name), email address, locality and country. It also contains your public key; which the Certificate Authority will put into your certificate.

When we create the Certificate Signing request below we will also generate a private key. The Certificate Signing request will only work with the private key that was generated with it. The private key will be needed for the certificate you'll buy, to work.

Here is how you can create the Certificate Signing request on your server:

```
$ openssl req -nodes -newkey rsa:2048 -sha256  -keyout myprivatekey.key -out certifi\
cate-signing-request.csr
```

Let's explain the openSSL parameters in detail:

- req: activates the part of openssl that deals with certificate requests signing
- -nodes: no des, stores the private key without protecting it with a passphrase. While this is not considered to be best practice, many people do not set a passphrase or later remove it, since services with pass phrase protected keys can not be auto-restarted without typing in the passphrase
- -newkey: generate a new private key
- rsa:2048 1024 is the default bit length of the private key. We will use 2048 bit keys because our Certificate Authority supports this and is required for certificates which expire after October 2013

- -sha256: used by certificate authorities to generate a SHA-2 certificates (which is more secure then SHA-1)
- -keyout myprivatekey.key: store the private key in a file called myprivatekey.key (in PEM format)
- -out certificate-signing-request.csr: store the certificate request in a file called certificate-signing-request.csr

When launching the above command you'll be asked to enter information that will be incorporated into your certificate request.

There are quite a few fields but you can leave some blank. For some fields there will be a default value (displayed in [...] brackets). If you enter '.', the field will be left blank.

- Country Name (2 letter code) [AU]: <2 letter country code> eg. BE for Belgium
- State or Province Name (full name) [Some-State]
- Locality Name (eg. city) []
- Organization Name (eg. company) [Internet Widgits Pty Ltd]: Wim Bervoets
- Organizational Unit Name (eg, section) []:
- Common Name (e.g. server FQDN or YOUR name) []: this is an important setting which we will discuss below.
- Email Address []: email address which will be in the certificate and used by the Certificate Authority to verify your request . Make sure this email is valid & you have access to it. The email address should also match with the email address in the DNS contact emails used for the particular domain you're requesting a certificate for.

The Common Name should be the domain name you're requesting a certificate for. Eg. www.mywebsite.com

This should include the www or the subdomain you're requesting a certificate for.

If you want to order a wildcard certificate which is valid for all subdomains you should specify this with a star; eg. *.mywebsite.com

OpenSSL will now ask you for a few 'extra' attributes to be sent with your certificate request:

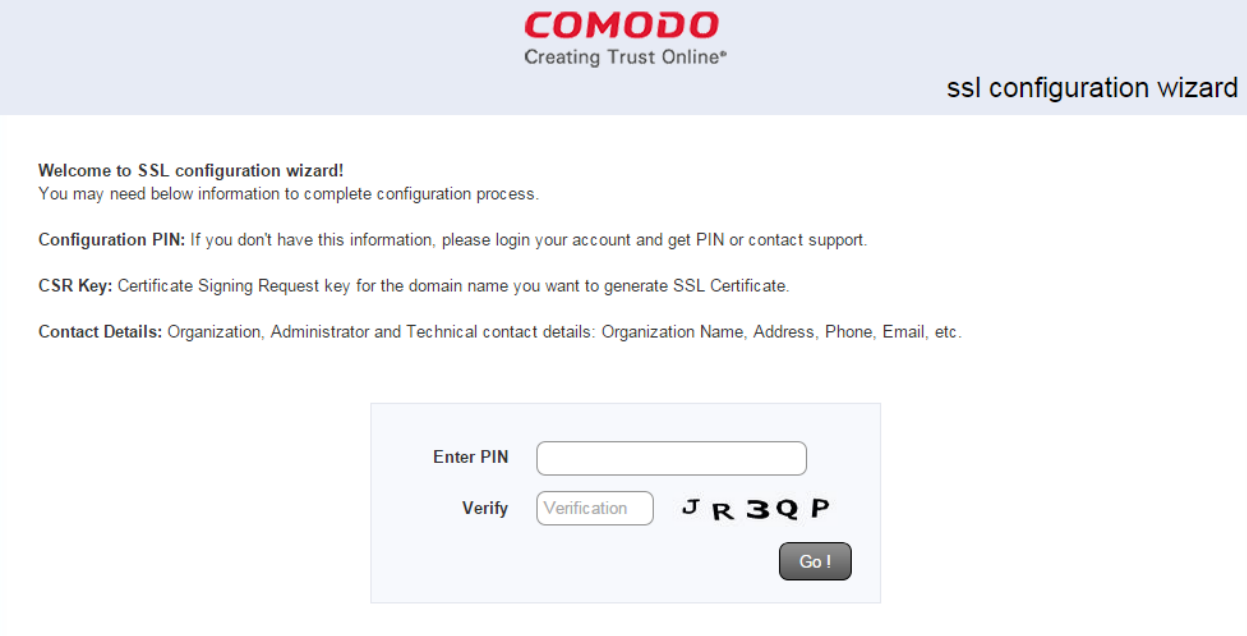
- a challenge password []: leave empty
- an optional company name []: leave empty

Now we can download the freshly generated csr file and use it when ordering our SSL certificate at the SSLStore.

Ordering a certificate

Let's suppose we want a Comodo Wildcard certificate. Go to https://www.thesslstore.com/wildcardssl-certificates.aspx?aid=52910623 and click on the Add To cart button next to 'Comodo EssentialSSL Wildcard certificate'.

Next you'll be asked for your billing details and credit card information. After completing these steps an email will be sent with a link to the Configure SSL service of Comodo (together with a PIN)



Comodo SSL Configuration wizard

Here you'll also need to provide the Certificate Signing request you have generated in the previous section.

After completing these steps, your domain will be validated by Comodo. Depending on the type of certificate this will take a few hours to one week to complete.

As we didn't choose an Extended Validation certificate, this validation was quick and we soon received a 'Domain Control Validation' email with another validation code for our certificate we requested.

This email was sent to the DNS contacts listed for our domain.

After entering the validation code on the Comodo website, the certificate was emailed to our email address.

The certificate zip file actually contained a few different certificates:

- A root Certificate Authority Certificate.crt

- 2 Intermediate Certificate Authority certificates (COMODORSAAddTrustCA.crt and COMODORSADomainValidationSecureServerCA.crt)
- The certificate for your domain.

You may wonder why there are so many different certificates included and what you need to do with it.

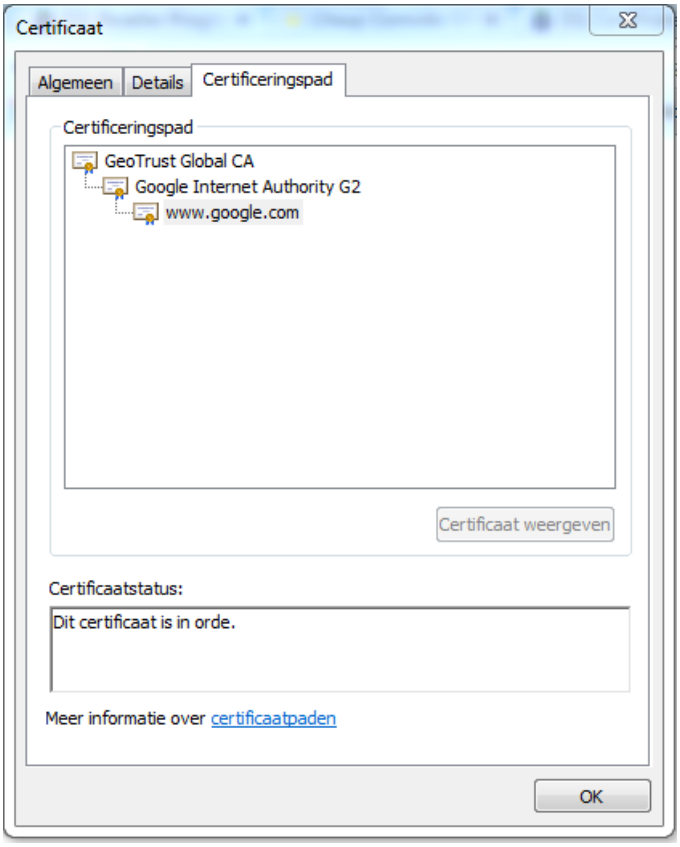
To explain this, we'll need to cover what SSL Certificate chains are.

Browsers and devices connecting to secure sites have a fixed list of Certificate Authorities they trust - the so called root CAs. The other kind of CAs are the intermediate Certificate Authorities.

If the certificate of the intermediate CA is not trusted by the browser or device, the browser will check if the certificate of the intermediate CA was issued by a trusted CA (this goes on until a trusted (root) CA is found).

This chain of SSL certificates from the root CA certificate, over the intermediate CA certificates to the end-user certificate for your domain represents the SSL certificate chain.

You can view the chain in all popular browsers, for example in Chrome you can click on the padlock item of a secure site, choose Connection and then Certificate data to view the chain:



HTTPS Certificate chain

For the Comodo certificate the chain is as follows:

- your domain certificate
- Comodo RSA Domain Validation Secure Server CA
- Comodo RSA Certification authority
- AddTrust External CA Root

In the next section we'll make use of the certificates as we install them in our nginx configuration.

Configuring nginx for SSL

First we will combine our domain certificate with all the intermediary certificates (except the root CA certificate).

We do this for the following reasons:

- the browser will receive the full certificate chain. (except for the root certificate but the browser already has this one builtin).
- Some browsers will display warnings when they can not find a trusted CA certificate in the chain. This can happen if the chain is not complete.
- Other browsers will try to download the intermediary CA certificates; this is not good for the performance of your website because it slows down setting up a secure connection. If we combine all the certificates and configure nginx properly this will be much faster.

Note: In general a combined SSL certificate with less intermediary CAs will be a little bit better performance wise still.

You can combine the certificates on your server, after you have uploaded all the certificate .crt files with the following command:

```
$ cat <your_domain>.crt COMODORSADomainValidationSecureServerCA.crt COMODORSAAddTrus\
tCA.crt > yourdomain.chained.crt
```

yourdomain.chained.crt can now be configured in nginx:

You'll need to add the following configuration inside a server {...} block in the nginx configuration. Please refer to our Configuring your website domain in nginx section.

```
$ sudo nano /usr/local/nginx/conf/conf.d/mywebsite.com.conf

server {
        server_name www.mywebsite.com;
        # SSL config
        listen <ipv4 address>:443 default_server ssl http2;
        listen [ipv6 address]:443 default_server ssl http2;

        ssl_certificate /usr/local/nginx/conf/<yourdomain.chained.crt>;
        ssl_certificate_key /usr/local/nginx/conf/<yourprivate.key>;
        ...
}
```

In this configuration we tell nginx to listen on an IPv4 and IPv6 address on the default HTTPS port 443. We enable ssl and http2.

HTTP/2 is the next generation standardized HTTP v2 protocol. It is based on the SPDY Google specification which manipulates HTTP traffic, with the goal to reduce web page load latency. It uses compression and prioritizes and multiplexes the transfer of a web page so that only one connection per client is required. (eg. Getting the html, images, stylesheets and javascript files all happens with a connection that is kept open).

You can check an example what kind of performance improvements are possible with HTTP2 on the Akaimai HTTP2 test page

HTTP/2 is best used with TLS (Transport Layer security) encryption (eg. https) for security and better compatibility across proxy servers.

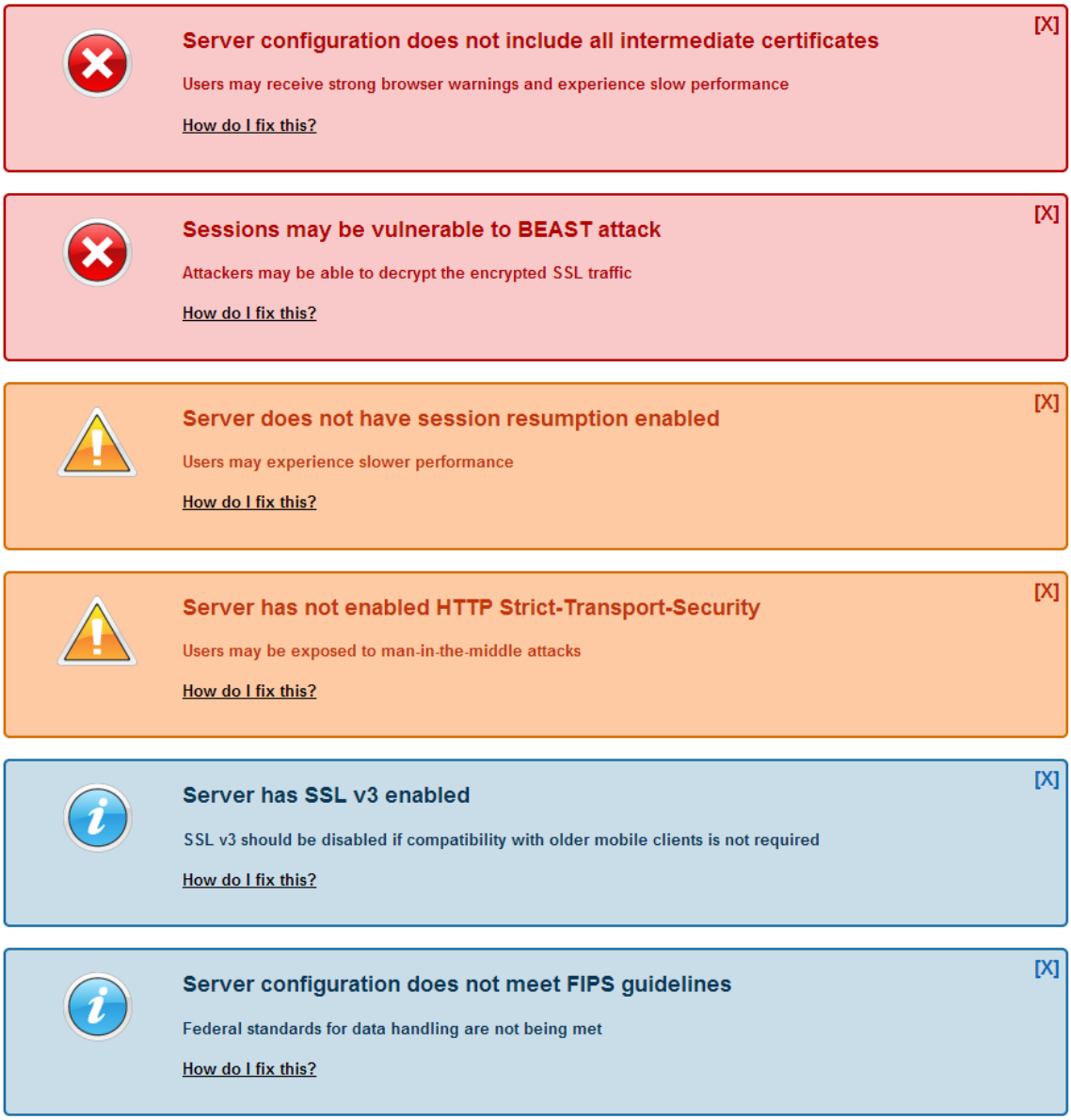
Now restart the nginx server. Your site should now be accessible via https.

We recommend you to now run an SSL analyzer. You'll get a security score and a detailed report of your SSL configuration:

1. verify the certificate is valid and trusted.
2. inspect the server configuration for protocol support, key exchange support and cipher support.

HTTPS Certificate chain

To get an A+ score the default nginx SSL configuration shown above is not enough. More likely you'll receive one or more of the following warnings:



HTTPS Certificate chain

Let's tune the https configuration in the next section!

Getting an A+ grade on SSLLabs.com

Disabling http access to your server

To make your users use the https version of your site by default, you'll need to redirect all http traffic to the https protocol. Here is an example server nginx configuration which does this:

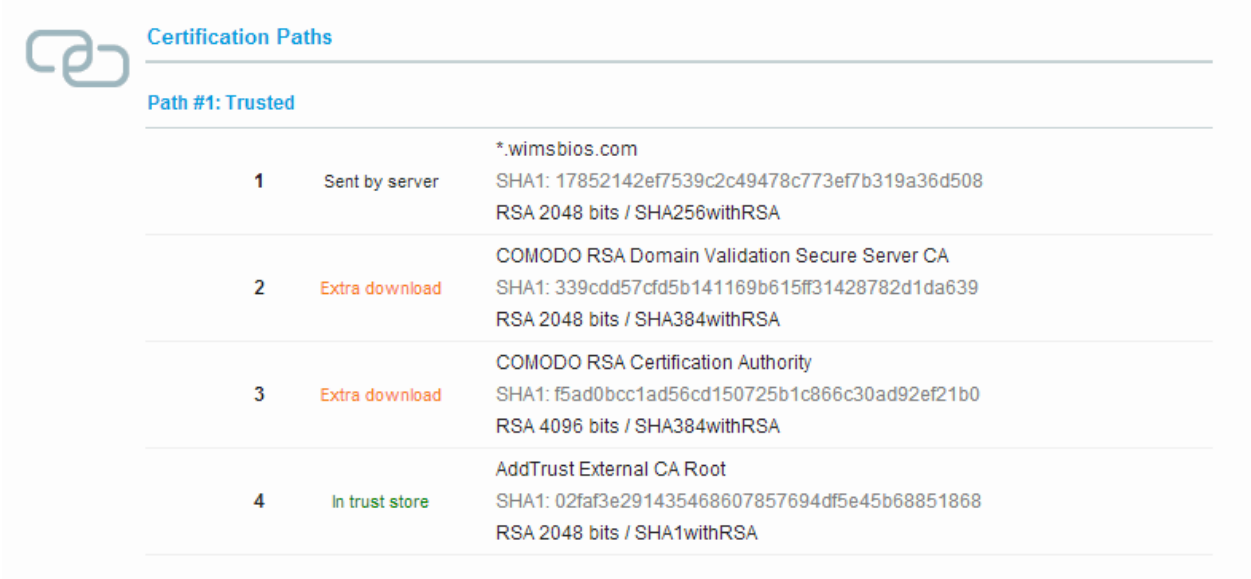
```
server {
        server_name www.yourwebsite.com;
        listen <ip_address>:80; # Listen on the HTTP port
        listen [<ip_address>]:80; # Listen on IPv6 address and HTTP 80 port

        return 301 https://$server_name$request_uri;
}
```

Fixing Server configuration does not include all intermediate certificates

Actually you should not be receiving this error, as we previously combined all the intermediate certificates with our domain certificate. If the SSLLabs test still reports this error, then you should revisit the previous section.

If you don't fix this error users may receive strong browser warnings and experience slow performance.



HTTPS Certificate chain problems

Another tester which analyzes the intermediate certificates is https://www.wormly.com/help/ssl-tests/intermediate-cert-chain

Session can be vulnerable to BEAST / POODLE attacks / SSLv3 is enabled on the server

Sometimes security issues are found in the security protocols or ciphers used for securing websites. Some of these issues get an official name like BEAST or POODLE attacks.

By using the latest version of OpenSSL and properly configuring the nginx SSL settings you can mitigate most of these issues.

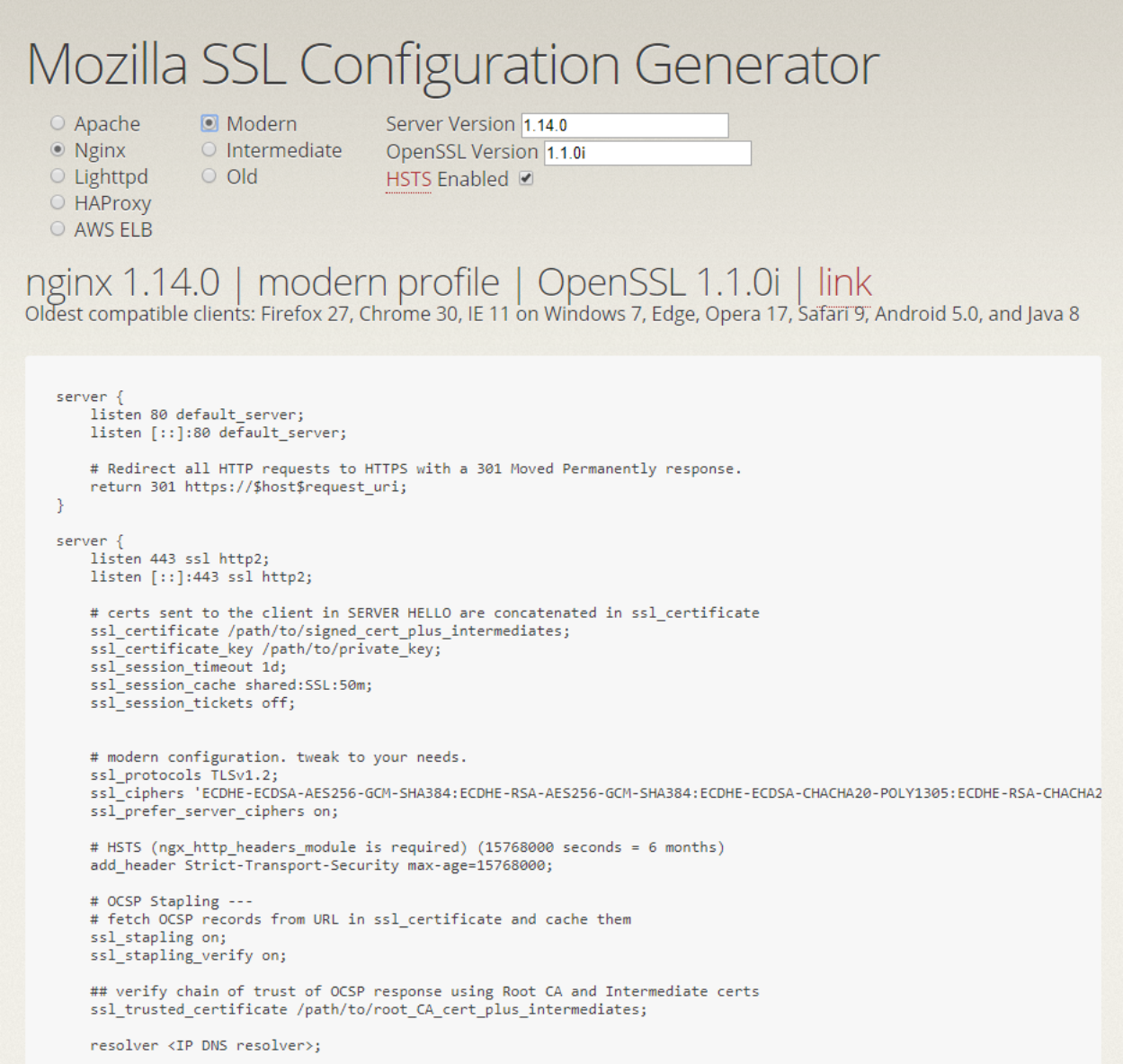
https://wiki.mozilla.org/Security/Server_Side_TLS has an up-to-date list of configuration settings to be used on your server. Actually there are three sets of configurations, a Modern, an Intermediate and Old configuration.

We recommend to at least use the Intermediate or the Modern version as they give you higher levels of security between browsers/clients and your server. The modern version is the most secure, but doesn't work well with very old browsers. To receive an A+ grade for the test you'll need to choose Modern.

Here are the minimum versions supported for the Modern & Intermediate configuration.

- Modern: Firefox 27, Chrome 30, IE 11 on Windows 7, Edge, Opera 17, Safari 9, Android 5.0, Java 8
- Intermediate: Firefox 1, Chrome 1, IE 7, Opera 5, Safari 1, Windows XP IE8, Android 2.3, Java 7

We'll use the online tool at https://mozilla.github.io/server-side-tls/ssl-config-generator/ to generate an 'Modern' configuration.



SSL Config generator

Choose nginx, Modern and fill in the nginx version & OpenSSL version. The nginx configuration generated should be like the one in the screenshot above.

In summary these settings will: * disable SSL3.0 protocol, TLSv1, TLSv1.1 * The SSL Ciphersuites nginx/OpenSSL supports server side are ordered: the most secure at the beginning of the list. This will make sure client/servers will try to use the most secure options they both support. * Specifies that server ciphers should be preferred over client ciphers when using the TLS protocols (to fix BEAST SSL attack) * Enable OSCP stapling (explained in the next chapter)

For Diffie-Hellman based ciphersuites an extra parameter is needed:

```
ssl_dhparam /usr/local/nginx/conf/dhparam.pem;
```

This file can be created by the following OpenSSL command:

```
$ sudo openssl dhparam -out /usr/local/nginx/conf/dhparam.pem 4096
Generating DH parameters, 2048 bit long safe prime, generator 2
This is going to take a long time
.....................................................................................\
.............
```

4096 means the parameter is 4096 bits in size.

OCSP Stapling

OCSP stands for Online Certificate Status Protocol. Let's explain the context a bit.

Certificates issued by a Certificate Authority can be revoked by the CA. For example because the customer lost their private key or was stolen, or the domain was transferred to a new owner.

The Online Certificate Status Protocol (OCSP) is one method for obtaining certificate revocation information. When presented with a certificate, the browser asks the issuing CA if there are any problems with it. If the certificate is fine, the CA can respond with a signed assertion that the certificate is still valid. If it has been revoked, however, the CA can say so by the same mechanism.

OCSP has a few drawbacks:

- it slows down new HTTPS connections. When the browser encounters a new certificate, it has to make an additional request to a server operated by the CA.
- Additionally, if the browser cannot connect to the CA, it must choose between two undesirable options: ** It can terminate the connection on the assumption that something is wrong, which decreases usability. ** It can continue the connection, which defeats the purpose of doing this kind of revocation checking.

OCSP stapling solves these problems by having the site itself periodically ask the CA for a signed assertion of status and sending that statement in the handshake at the beginning of new HTTPS connections.

To enable OCSP stapling in nginx; add the following options:

```
ssl_stapling on;
ssl_stapling_verify on;
ssl_trusted_certificate /usr/local/nginx/conf/ca.root.crt;
```

The ssl_trusted_certificate file should only contain the root Certificate Authority certificate. In our case, we created this file like this:

```
cat AddTrustExternalCARoot.crt > ca.root.crt
```

When nginx asks for the revocation status of your certificate, it'll ask the CA this in a secure manner using the root CA certificate (ca.root.crt in our case).

To validate OSCP stapling is working run the following command:

```
$ openssl s_client -connect www.<yourwebsite>.com:443 -tls1 -tlsextdebug -status < /\
dev/null| grep OCSP
```

It should give back:

```
OCSP Response Data:
    OCSP Response Status: successful (0x0)
    Response Type: Basic OCSP Response
```

when it is working.

"OCSP response: no response sent" means it is not active yet.

You may need to rerun this command a few times if you just recently started nginx.

If OCSP is not working correctly nginx will also issue the following warning in its error log file (/var/log/nginx/error.log)

```
2015/12/12 04:47:03 [error] 1472#0: OCSP_basic_verify() failed (SSL: error:27069065:\
OCSP routines:OCSP_basic_verify:certificate verify error:Verify error:unable to get \
issuer certificate) while requesting certificate status, responder: gv.symcd.com
```

Implementing HTTP Strict Transport Security

Suppose a user types in the URL of your website in a browser without any https or http protocol specified. Then the browser will likely choose to load the site via http (the unsecure version). Even if you have configured your server to redirect all http requests to https, the user may will talk to the non-encrypted version of the site before being redirected.

This opens up the potential for a man-in-the-middle attack, where the redirect could be exploited to direct a user to a malicious site instead of the secure version of the original page.

The HTTP Strict Transport Security feature lets a website inform the browser it should never try to load the site using HTTP, and that it should automatically convert all attempts to access the site using HTTP to HTTPs requests.

In your nginx configuration you'll need to add the following line:

```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains";
```

This adds an HTTP Strict-Transport-Security header, specifiying that all subdomains should also be run on https and that the browser should not try the http version for one year.

Optimizing SSL

To further optimize the SSL performance of nginx we can enable some caches.

```
ssl_session_cache shared:SSL:50m;
ssl_session_timeout 1d;
```

The ssl_session_cache will create a shared cache between all the nginx worker processes. We have reserved 50MB for storing SSL sessions (for 1 day). According to the nginx documentation 1MB can store about 4000 sessions. You can reduce or increase the size of the cache based on the traffic you're expecting.

Enabling Certificate Authority Authorization (CAA)

Over a hundred certificate authorities (CAs) have the power to issue certificates which vouch for the identity of your website. Certificate Authority Authorization (CAA) is a way for you to whitelist the CAs you actually use so you can minimize your risk from security vulnerabilities in all the others.

As of September 8, 2017, all certificate authorities are required to respect your CAA policy.

To whitelist the CA you are using, a special kind of DNS Record was created: CAA record - short for Certificate Authority Authorization record.

In the following example we will add a CAA record for Comodo (Sectigo). To know what values to enter into DNSMadeEasy, our DNS provider, you can use the CAA record generator at SSLMate

```
example.com.        CAA        0 issue "sectigo.com"
```

To generate a CAA Record in DNSMadeEasy:

- Add a CAA Record ** Name: <empty> ** Provider: Comodo ** Type: issue ** Value: "comodoca.com" ** Issuer Critical: 0 ** TTL: 1800

Enabling SSL on a CDN

When serving your site over https, you need to make sure that all resources used by your HTML are also served via HTTPS. (eg. Images, javascript, stylesheets).

When you're using a CDN to host your resources, you'll need to configure the SSL settings in your CDN Account.

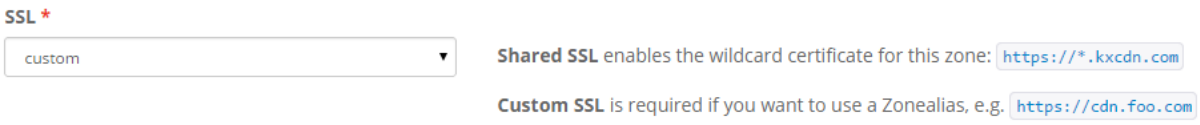
We're going to show you how you can enable HTTPS on a KeyCDN server. The process will be similar for eg. MaxCDN.

For setting up a CDN, take a look at our Chapter Using a CDN.

- Go to KeyCDN and login to your account.
- Click on Zones and click on the Manage button -> Edit for the zone you want to configure.
- Click on Show Advanced features

The settings we need to configure are:

- SSL
- Custom SSL certficate
- Custom SSL Private key
- Force SSL

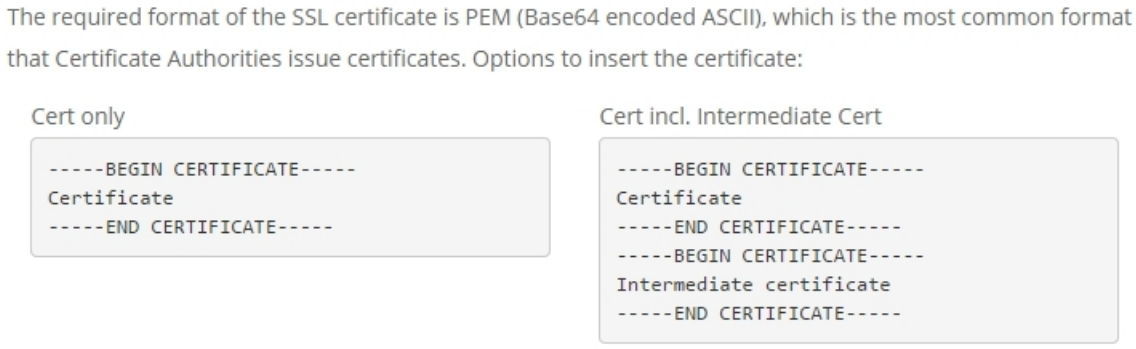


Enabling SSL on a CDN

As we want to configure https://cdn.<yourdomain.com>, we choose the Custom SSL option.

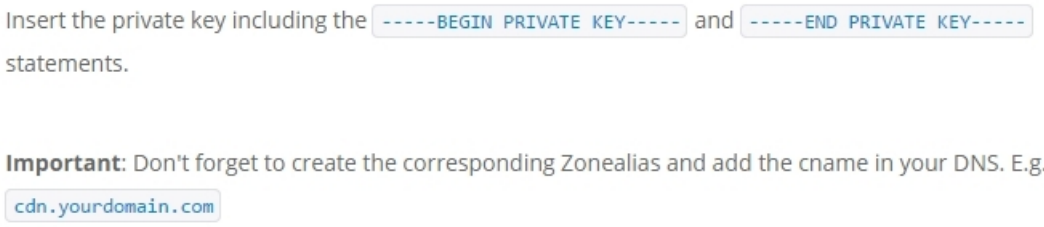
In the Custom SSL Certificate, we need to include our domain certificate and the intermediate CA certificates.

You should copy the text from our chained certificate file at /usr/local/nginx/conf/<yourdomain.chained.crt>. Below you can see the exact syntax to use.



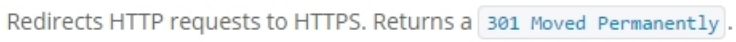
Custom SSL certificate on a CDN

You'll also need to provide your private key in the Custom SSL Private Key section. This key is available at /usr/local/nginx/conf/<yourprivate.key>



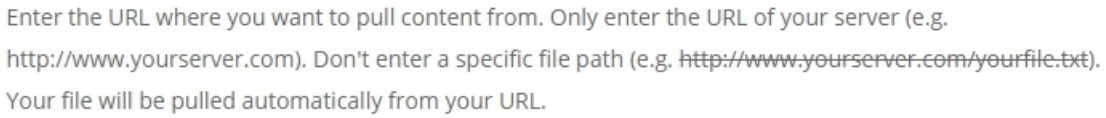
Custom SSL private key

Lastly enable the setting to redirect cdn.<yourwebsite.com> requests to https:



Redirect CDN URLs to https

Make sure to use a https URL for your Origin URL too (eg. https://www.yourwebsite.com)

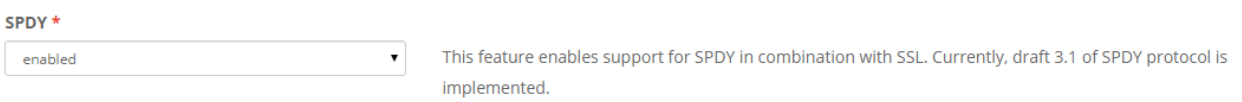


Use a https URL for your Origin URL

Please note that most CDNs that support SSL implement it via Server Name Indication which means multiple certificates can be presented to the browser on 1 single IP address. This reduces their need for dedicated IP addresses per customer which lowers the cost significantly. The only (small) downlside of SNI is that it isn't supported by IE6 on Windows XP, meaning those users will see a certificate warning.

Enabling SPDY or HTTP/2 on a CDN

As we have enabled https on our CDN, we can now also enable the Google SPDY protocol or HTTP/2 which will speed up the https communications significantly.



Enabling SPDY on a CDN