How To Create a Self-Signed SSL Certificate for Apache in Ubuntu 18.04

Posted July 5, 2018 158.1k APACHE UBUNTU UBUNTU 18.04 SECURITY

By Brian Boucheron
Become an author

Not using **Ubuntu 18.04**? Choose a different version:

A previous version of this tutorial was written by Justin Ellingwood

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.

ıts without
e system

also assists users in verifying the identity of the sites that they are connecting with.

In this guide, we will show you how to set up a self-signed SSL certificate for use with an Apache web server on Ubuntu 18.04.

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.

A self-signed certificate may be appropriate if you do not have a domain name associated with your server and for instances where an encrypted web interface is not user-facing. If you *do* have a domain name, in many cases it is better to use a CA-signed certificate. You can find out how to set up a free trusted certificate with the Let's Encrypt project here.

Prerequisites

Before you begin, you should have a non-root user configured with `sudo` privileges. You can learn how to set up such a user account by following our Initial Server Setup with Ubuntu 18.04.

You will also need to have the Apache web server installed. If you would like to install an entire LAMP (Linux, Apache, MySQL, PHP) stack on your server, you can follow our guide on setting up LAMP on Ubuntu 18.04. If you just want the Apache web server, skip the steps pertaining to PHP and MySQL.

When you have completed the prerequisites, continue below.

Step 1 – Creating 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.

ommand:

```
rsa:2048 -keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-selfsigned
```

You will be asked a series of questions. Before we go over that, let's take a look at what is happening in the command we are issuing:

- **openssl**: This is the basic command line tool for creating and managing OpenSSL certificates, keys, and other files.
- **req**: This subcommand specifies that we 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. We want to create a new X.509 cert, so we are using this subcommand.
- **-x509**: This further modifies the previous subcommand by telling the utility that we 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 our certificate with a passphrase. We need Apache to be able to read the file, without user intervention, when the server starts up. A passphrase would prevent this from happening because we would have to enter it after every restart.
- **-days 365**: This option sets the length of time that the certificate will be considered valid. We set it for one year here.
- **-newkey rsa:2048**: This specifies that we want to generate a new certificate and a new key at the same time. We did not create the key that is required to sign the certificate in a previous step, so we 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 we are creating.
- **-out**: This tells OpenSSL where to place the certificate that we are creating.

As we stated above, these options will create both a key file and a certificate. We will be asked a few questions about our 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, more likely, your server's public IP address.**

```
Output
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:New York
Locality Name (eg, city) []:New York City
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Bouncy Castles, Inc.
Organizational Unit Name (eg, section) []:Ministry of Water Slides
Common Name (e.g. server FQDN or YOUR name) []:server_IP_address
Email Address []:admin@your_domain.com
```

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

Step 2 – Configuring Apache to Use SSL

We have created our key and certificate files under the `/etc/ssl` directory. Now we just need to modify our Apache configuration to take advantage of these.

We will make a few adjustments to our configuration:

1. We will create a configuration snippet to specify strong default SSL settings.
2. We will modify the included SSL Apache Virtual Host file to point to our generated SSL certificates.
3. (Recommended) We will modify the unencrypted Virtual Host file to automatically redirect requests to the encrypted Virtual Host.

When we are finished, we should have a secure SSL configuration.

Creating an Apache Configuration Snippet with Strong Encryption Settings

First, we will create an Apache configuration snippet to define some SSL settings. This will set Apache up with a strong SSL cipher suite and enable some advanced features that will help keep our server secure. The parameters we will set can be used by any Virtual Hosts enabling SSL.

Create a new snippet in the `/etc/apache2/conf-available` directory. We will name the file `ssl-params.conf` to make its purpose clear:

To set up Apache SSL securely, we will be using the recommendations by Remy van Elst on the Cipherli.st site. This site is designed to provide easy-to-consume encryption settings for popular software.

The suggested settings on the site linked to above 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 the link on the page labelled "Yes, give me a ciphersuite that works with legacy / old software." That list can be substituted for the items copied below.

The choice of which config you use will depend largely on what you need to support. They both will provide great security.

For our purposes, we can copy the provided settings in their entirety. We will just make one small change. We will disable the `Strict-Transport-Security` header (HSTS).

Preloading HSTS provides increased security, but can have far reaching consequences if accidentally enabled or enabled incorrectly. In this guide, we will not enable the settings, but you can modify that if you are sure you understand the implications.

Before deciding, take a moment to read up on HTTP Strict Transport Security, or HSTS, and specifically about the "preload" functionality

Paste the configuration into the `ssl-params.conf` file we opened:

/etc/apache2/conf-available/ssl-params.conf

```
SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
SSLProtocol All -SSLv2 -SSLv3 -TLSv1 -TLSv1.1
SSLHonorCipherOrder On
# Disable preloading HSTS for now.  You can use the commented out header line that includes
# the "preload" directive if you understand the implications.
# Header always set Strict-Transport-Security "max-age=63072000; includeSubDomains; preloac
Header always set X-Frame-Options DENY
Header always set X-Content-Type-Options nosniff
# Requires Apache >= 2.4
SSLCompression off
SSLUseStapling on
```

```
# Requires Apache >= 2.4.11
SSLSessionTickets Off
```

Save and close the file when you are finished.

Modifying the Default Apache SSL Virtual Host File

Next, let's modify `/etc/apache2/sites-available/default-ssl.conf`, the default Apache SSL Virtual Host file. If you are using a different server block file, substitute its name in the commands below.

Before we go any further, let's back up the original SSL Virtual Host file:

```
$ sudo cp /etc/apache2/sites-available/default-ssl.conf /etc/apache2/sites-available/defau
```

Now, open the SSL Virtual Host file to make adjustments:

```
$ sudo nano /etc/apache2/sites-available/default-ssl.conf
```

Inside, with most of the comments removed, the Virtual Host file should look something like this by default:

/etc/apache2/sites-available/default-ssl.conf

```
<IfModule mod_ssl.c>
        <VirtualHost _default_:443>
                ServerAdmin webmaster@localhost

                DocumentRoot /var/www/html

                ErrorLog ${APACHE_LOG_DIR}/error.log
                CustomLog ${APACHE_LOG_DIR}/access.log combined

                SSLEngine on

                SSLCertificateFile      /etc/ssl/certs/ssl-cert-snakeoil.pem
                SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

                <FilesMatch "\.(cgi|shtml|phtml|php)$">
```

```
                <Directory /usr/lib/cgi-bin>
                                SSLOptions +StdEnvVars
                </Directory>

        </VirtualHost>
</IfModule>
```

We will be making some minor adjustments to the file. We will set the normal things we'd want to adjust in a Virtual Host file (ServerAdmin email address, ServerName, etc., and adjust the SSL directives to point to our certificate and key files.

After making these changes, your server block should look similar to this:

/etc/apache2/sites-available/default-ssl.conf

```
<IfModule mod_ssl.c>
        <VirtualHost _default_:443>
                ServerAdmin your_email@example.com
                ServerName server_domain_or_IP

                DocumentRoot /var/www/html

                ErrorLog ${APACHE_LOG_DIR}/error.log
                CustomLog ${APACHE_LOG_DIR}/access.log combined

                SSLEngine on

                SSLCertificateFile      /etc/ssl/certs/apache-selfsigned.crt
                SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key

                <FilesMatch "\.(cgi|shtml|phtml|php)$">
                                SSLOptions +StdEnvVars
                </FilesMatch>
                <Directory /usr/lib/cgi-bin>
                                SSLOptions +StdEnvVars
                </Directory>

        </VirtualHost>
</IfModule>
```

Save and close the file when you are finished.

(Recommended) Modifying the HTTP Host File to Redirect to HTTPS

As it stands now, the server will provide both unencrypted HTTP and encrypted HTTPS traffic. For better security, it is recommended in most cases to redirect HTTP to HTTPS automatically. If you do not want or need this functionality, you can safely skip this section.

To adjust the unencrypted Virtual Host file to redirect all traffic to be SSL encrypted, we can open the `/etc/apache2/sites-available/000-default.conf` file:

```
$ sudo nano /etc/apache2/sites-available/000-default.conf
```

Inside, within the `VirtualHost` configuration blocks, we need to add a `Redirect` directive, pointing all traffic to the SSL version of the site:

/etc/apache2/sites-available/000-default.conf

```
<VirtualHost *:80>
        . . .

        Redirect "/" "https://your_domain_or_IP/"

        . . .
</VirtualHost>
```

Save and close the file when you are finished.

Step 3 – Adjusting the Firewall

If you have the `ufw` firewall enabled, as recommended by the prerequisite guides, you might need to adjust the settings to allow for SSL traffic. Luckily, Apache registers a few profiles with `ufw` upon installation.

We can see the available profiles by typing:

```
$ sudo ufw app list
```

You should see a list like this:

```
Available applications:
  Apache
  Apache Full
  Apache Secure
  OpenSSH
```

You can see the current setting by typing:

```
$ sudo ufw status
```

If you allowed only regular HTTP traffic earlier, your output might look like this:

```
Output
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Apache                     ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Apache (v6)                ALLOW       Anywhere (v6)
```

To additionally let in HTTPS traffic, we can allow the “Apache Full” profile and then delete the redundant “Apache” profile allowance:

```
$ sudo ufw allow 'Apache Full'
$ sudo ufw delete allow 'Apache'
```

Your status should look like this now:

```
$ sudo ufw status
```

```
Output
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
```

```
OpenSSH (v6)               ALLOW       Anywhere (v6)
Apache Full (v6)           ALLOW       Anywhere (v6)
```

Step 4 – Enabling the Changes in Apache

Now that we've made our changes and adjusted our firewall, we can enable the SSL and headers modules in Apache, enable our SSL-ready Virtual Host, and restart Apache.

We can enable `mod_ssl`, the Apache SSL module, and `mod_headers`, needed by some of the settings in our SSL snippet, with the `a2enmod` command:

```
$ sudo a2enmod ssl
$ sudo a2enmod headers
```

Next, we can enable our SSL Virtual Host with the `a2ensite` command:

```
$ sudo a2ensite default-ssl
```

We will also need to enable our `ssl-params.conf` file, to read in the values we set:

```
$ sudo a2enconf ssl-params
```

At this point, our site and the necessary modules are enabled. We should check to make sure that there are no syntax errors in our files. We can do this by typing:

```
$ sudo apache2ctl configtest
```

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

```
Output
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, us
Syntax OK
```

The first line is just a message telling you that the `ServerName` directive is not set globally. If you want to get rid of that message, and you have a registered domain

name or IP address in `/etc/apache2/apache2.conf`. This is optional as the message will do no harm.

If your output has `Syntax OK` in it, your configuration file has no syntax errors. We can safely restart Apache to implement our changes:

```
$ sudo systemctl restart apache2
```

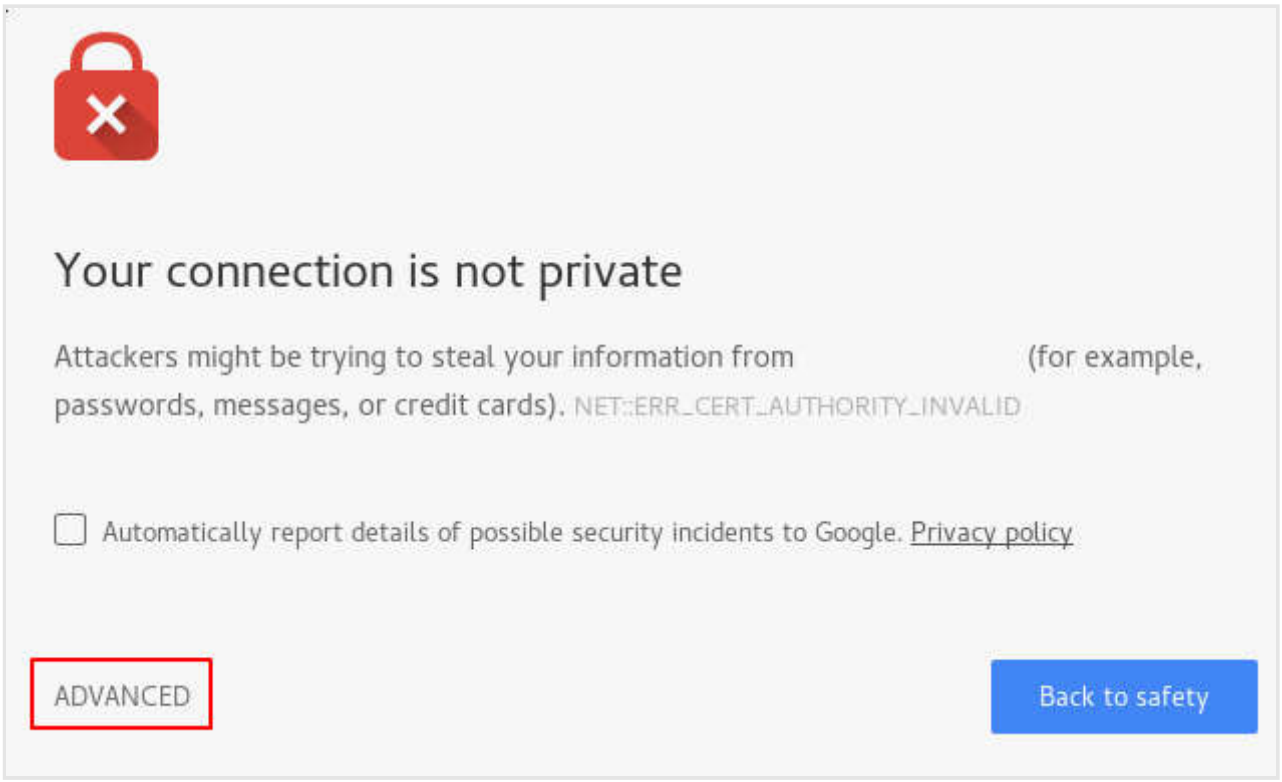
Step 5 – Testing Encryption

Now, we're ready to test our SSL server.

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

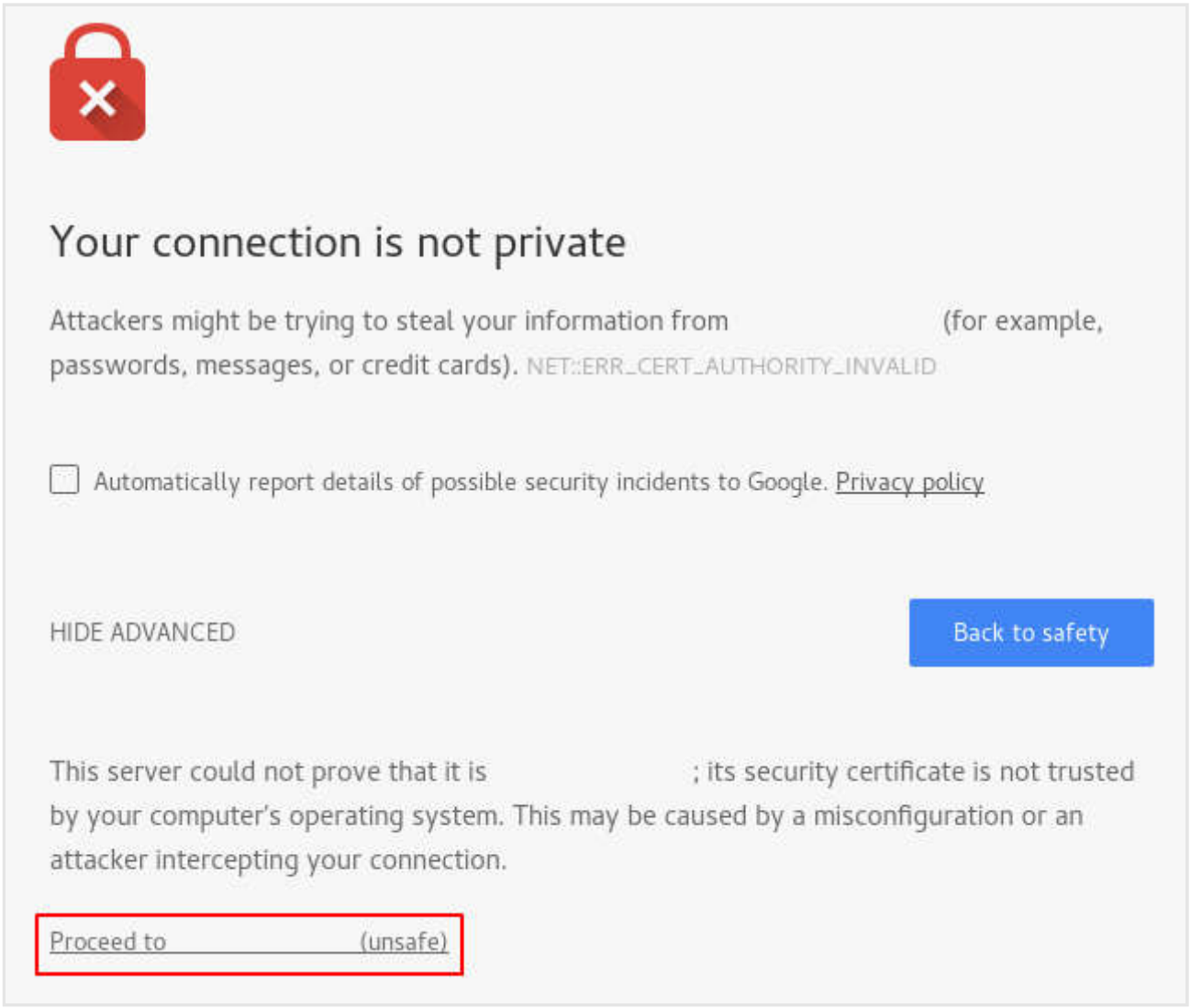
```
https://server_domain_or_IP
```

Because the certificate we 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:



f our
NCED"

and then the link provided to proceed to your host anyways:



You should be taken to your site. If you look in the browser address bar, you will see a lock with an "x" over it. In this case, this just means that the certificate cannot be validated. It is still encrypting your connection.

If you configured Apache to redirect HTTP 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.

Step 6 – Changing to a Permanent Redirect

If your redirect worked correctly and you are sure you want to allow only encrypted traffic, you should modify the unencrypted Apache Virtual Host again to make the redirect permanent.

Open your server block configuration file again:

```
$ sudo nano /etc/apache2/sites-available/000-default.conf
```

Find the `Redirect` line we added earlier. Add `permanent` to that line, which changes the redirect from a 302 temporary redirect to a 301 permanent redirect:

/etc/apache2/sites-available/000-default.conf

```
<VirtualHost *:80>
        . . .

        Redirect permanent "/" "https://your_domain_or_IP/"

        . . .
</VirtualHost>
```

Save and close the file.

Check your configuration for syntax errors:

```
$ sudo apache2ctl configtest
```

When you're ready, restart Apache to make the redirect permanent:

```
$ sudo systemctl restart apache2
```

Conclusion

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

Was this helpful? Yes No

Report an issue

Related

TUTORIAL

How to Use Ansible to Install and Set Up Apache on Ubuntu 18.04

Ansible offers a simple architecture that doesn't require special software t...

TUTORIAL

How To Migrate Redis Data with Replication on Ubuntu 18.04

Replication is the practice of regularly copying data from one database to...

TUTORIAL

How To Install the OpenLiteSpeed Web Server on Ubuntu 18.04

OpenLiteSpeed is an optimized open source web server that can be...

CHEATSHEET

How To Manage Sorted Sets in Redis

In Redis, sorted sets are a data type similar to sets in that both are non repeating groups of strings. The...

Still looking for an answer?

Ask a question

Search for more help

18 Comments

Leave a comment...

Sign In to Comment

danke *August 1, 2018*

0 Thx for the turorial,

but it only works for me with firefox 61(64bit), chrome 68 and edge 42 won't work.

i don't use a domain, only ip

failure: DLG*FLAGS*INVALID_CA

> danke *August 2, 2018*
>
> 0 i used the go skript on this site for the generation of the certificate with an ip and it works now
>
> https://docs.minio.io/docs/how-to-secure-access-to-minio-server-with-tls.html

Persisto *August 23, 2018*

0 Very refreshing to see step by step guidance in Linux which is up-to-date, complete and does not make any assumptions about the knowledge of the user. (It worked!)

marcogomesr *October 25, 2018*

0 HEY!!!
I got ubuntu 16.04.4 on lightsail... would this tutorial work ?...

Thanks

> superxingzheng *December 10, 2018*
>
> 0 it works for me on 16.04

ImFree *October 29, 2018*

0 how to change not secure to secure

Mechanic *December 8, 2018*

0 I used this guide since ubuntu 16.04 and even in ubuntu 18.04, but in my new install of ubuntu 18.04, i follwed the same steps but no error in following the steps, but while accessing the site from browser (firefox or opera), the following error is thrown:

In Opera:

This site can't provide a secure connection
<ip address> uses an unsupported protocol.

ERR*SSL*VERSION*OR*CIPHER_MISMATCH
null

Unsupported protocol
The client and server don't support a common SSL protocol version or cipher suite.

In Firefox:

An error occurred during a connection to <ip address>. Cannot communicate securely with peer: no common encryption algorithm(s). Error code: SSL*ERROR*NO*CYPHER*OVERLAP

lisa50469 *December 15, 2018*

0 Just created this Droplet, and following the instructions I get the following error:

– Unit apache2.service has begun starting up.
Dec 15 16:47:33 ubuntu-s-1vcpu-1gb-sfo2-01 apachectl[1305]: AH00526: Syntax error on line 1 of /etc/apache2/conf-enabled/ssl-params.conf:
Dec 15 16:47:33 ubuntu-s-1vcpu-1gb-sfo2-01 apachectl[1305]: Invalid command 'SSLCipherSuite', perhaps misspelled or defined by a module not included in the server
Dec 15 16:47:33 ubuntu-s-1vcpu-1gb-sfo2-01 apachectl[1305]: Action 'start' failed.
Dec 15 16:47:33 ubuntu-s-1vcpu-1gb-sfo2-01 apachectl[1305]: The Apache error log may have more information.
Dec 15 16:47:33 ubuntu-s-1vcpu-1gb-sfo2-01 systemd[1]: apache2.service: Control process exited, code=exited status=1
Dec 15 16:47:33 ubuntu-s-1vcpu-1gb-sfo2-01 systemd[1]: apache2.service: Failed with result 'exit-code'.
Dec 15 16:47:33 ubuntu-s-1vcpu-1gb-sfo2-01 systemd[1]: Failed to start The Apache HTTP Server.

SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
SSLProtocol All -SSLv2 -SSLv3 -TLSv1 -TLSv1.1
SSLHonorCipherOrder On

Disable preloading HSTS for now. You can use the commented out header line that includes
the “preload” directive if you understand the implications.
Header always set Strict-Transport-Security “max-age=63072000; includeSubDomains; preload”
Header always set X-Frame-Options DENY
Header always set X-Content-Type-Options nosniff

Requires Apache >= 2.4

SSLStaplingCache “shmcb:logs/stapling-cache(150000)”

Requires Apache >= 2.4.11

SSLSessionTickets Off

fadyeffat *December 17, 2018*

1 I followed the steps but it gives me the apache page not my website

datasci *January 4, 2019*

0 for those wanting to turn on directory privacy (follow this guide: https://www.digitalocean.com/community/tutorials/how-to-set-up-password-authentication-with-apache-on-ubuntu-16-04)

In order for you to then be able to redirect HTTP to HTTPS AND have the authentication enabled, add this block of code into Step 2, under /etc/apache2/sites-available/default-ssl.conf

```
<Directory /var/www/html>
                    AuthType Basic
                    AuthName "Restricted Content"
                    AuthUserFile /etc/apache2/.htpasswd
                    Require valid-user
</Directory>
```

So, the top of your default-ssl.conf will look like this,

```
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        <Directory /var/www/html>
                    AuthType Basic
                    AuthName "Restricted Content"
                    AuthUserFile /etc/apache2/.htpasswd
                    Require valid-user
            </Directory>

        ServerAdmin webmaster@localhost
.....................rest of file code..............
```

```
Redirect permanent "/" "https://your_domain_or_IP"
```

I was able to achieve HTTP redirect to HTTPS with directory privacy enabled. You must add that block of code above to default-ssl.conf (similar to how you do it in the 'how-to-set-up-password-authentication-with-apache-on-ubuntu-16-04' tutorial, and you must complete that tutorial as well to even enable password authentication on Apache).

Hope this helps someone!

mnizam *February 8, 2019*

0

I successfully completed all of the steps until Step 4, but can't connect to http://FQDN-HOSTNAME or https://FQDN-HOSTNAME where FQDN-HOSTNAME is my fully qualified domain name for my web server.

All browsers are more or less giving the same error:

This site can't provide a secure connection FQDN-HOSTNAME sent an invalid response. Try running Windows Network Diagnostics.

ERR*SSL*PROTOCOL_ERROR

Can anyone please point me in the right direction, and help troubleshoot this issue?

$ openssl s_client -connect FQDN-HOSTNAME:443 -CApath /etc/ssl/certs
CONNECTED(00000003)

139951192613312:error:1408F10B:SSL routines:ssl3*get*record:wrong version number:../ssl/record/ssl3_record.c:252:

no peer certificate available

No client certificate CA names sent

SSL handshake has read 5 bytes and written 176 bytes

Verification: OK

New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE

Protocol : TLSv1.2
Cipher : 0000
Session-ID:
Session-ID-ctx:
Master-Key:
PSK identity: None
PSK identity hint: None
SRP username: None
Start Time: 1549634693
Timeout : 7200 (sec)
Verify return code: 0 (ok)

Extended master secret: no

$ curl -verbose https://FQDN-HOSTNAME

- Rebuilt URL to: https://FQDN-HOSTNAME/
- Trying 24.197.220.137...
- TCP_NODELAY set
- Connected to FQDN-HOSTNAME (24.197.220.137) port 443 (#0)
- ALPN, offering h2
- ALPN, offering http/1.1
- successfully set certificate verify locations:
- CAfile: /etc/ssl/certs/ca-certificates.crt CApath: /etc/ssl/certs
- TLSv1.2 (OUT), TLS handshake, Client hello (1):
- error:1408F10B:SSL routines:ssl3*get*record:wrong version number
- stopped the pause stream!
- Closing connection 0 curl: (35) error:1408F10B:SSL routines:ssl3*get*record:wrong version number

martiusfox *April 26, 2019*

0 excellent guide, composed carefully... many thanks!

Hi
I found that in my Apache configuration (somewhere) I already had “Header always set X-Frame-Options SAMEORIGIN”. It clashed with the similar header in *ssl-params.conf*.
So I removed that line and framed web-based application work as they should! Eg. PhpPgAdmin

loxx927 *July 11, 2019*

0 Thanks for the excellent tutorial, this was really helpful. I managed to successfully complete the tutorial (with a minor tweak), but excuse my ignorance when it comes to certs...

In both Firefox and Chrome, each browser reports that the page is not secure. So is it possible to use this method to take it all the way to the green lock, or because of the nature of the certs being self signed, they'll never really be truly “secure” and not require the user to click through all the add exception boxes?

IE, you couldn't / wouldn't use this in a production environment, correct?

filichigo *July 31, 2019*

1 DONT WORK!!! Can't undo thith fucked bullsheet!!!

wrajith *August 7, 2019*

0 Thank you for the detailed tutorial. Everything worked as expected.

kmcmullin11 *August 22, 2019*

0 great now i cant even connect to my own damn website. The fuck is this shit

spmssdangeros *October 17, 2019*

0 Hi I am having trouble with this.

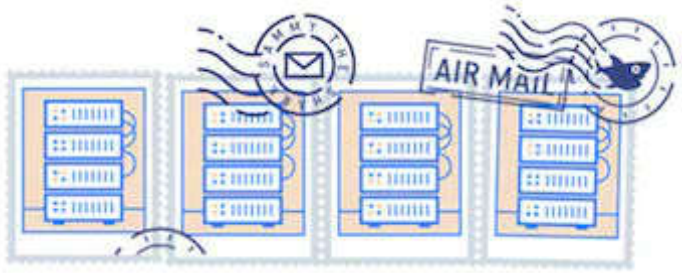
root@pms:/home/spm# sudo service apache2 restart

- Restarting web server apache2 [fail]
- The apache2 configtest failed. Output of config test was: AH00526: Syntax error on line 15 of /etc/apache2/conf-enabled/ssl-params.conf: Invalid command

the server configuration Action 'configtest' failed. The Apache error log may have more information.

Can someone please help me i only follow the guide

Find a DigitalOcean Meetup near you.



GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.

Featured on Community Kubernetes Course Learn Python 3 Machine Learning in Python Getting started with Go Intro to Kubernetes

DigitalOcean Products Droplets Managed Databases Managed Kubernetes Spaces Object Storage Marketplace

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

Learn More