CONTENTS

RELATED

Initial Server Setup with Ubuntu 12.04

View

How To Install Ruby on Rails on Ubuntu 12.04 LTS (Precise Pangolin) with RVM

View

// Tutorial //

How To Create a Self-Signed SSL Certificate for Apache in Ubuntu 16.04

Pu n April 22, 2016

Apache Security Ubuntu Ubuntu 16.04

COOKIE PREFERENCES

By Justin Ellingwood

Not using Ubuntu 16.04?
Choose a different version or distribution.

Ubuntu 16.04

# 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 learn how to set up a self-signed SSL certificate for use with an Apache web server on an Ubuntu 16.04 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 au

COOKIE PREFERENCES

included with web browsers and operating systems, users cannot use the certificate to validate the identity of your server automatically. As a result, your 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 How To Secure Apache with Let's Encrypt on Ubuntu 16.04 tutorial.

# Prerequisites

Before starting this tutorial, you'll need the following:

- Access to a Ubuntu 16.04 server with a non-**root**, sudo-enabled user. Our Initial Server Setup with Ubuntu 16.04 guide can show you how to create this account.
- You will also need to have Apache installed. You can install Apache using `apt`. First, update the local package index to reflect the latest upstream changes:

```
$ sudo apt update
```
Copy

Then, install the `apache2` package:

```
$ sudo apt install apache2
```
Copy

And finally, if you have a `ufw` firewall set up, open up the `http` and `https` ports:

```
$ sudo ufw allow "Apache Full"
```
Copy

After these steps are complete, be sure you are logged in as your non-**root** user and continue with the tutorial.

# Step 1 – Enabling `mod_ssl`

Before we can use *any* SSL certificates, we first have to enable `mod_ssl`, an Apache module that provides support for SSL encryption.

COOKIE PREFERENCES

Enable `mod_ssl` with the `a2enmod` command:

```
$ sudo a2enmod ssl
```

Copy

Restart Apache to activate the module:

```
$ sudo systemctl restart apache2
```

Copy

The `mod_ssl` module is now enabled and ready for use.

# Step 2 – Create the SSL Certificate

Now that Apache is ready to use encryption, we can move on to generating a new SSL certificate. The certificate will store some basic information about your site, and will be accompanied by a key file that allows the server to securely handle encrypted data.

We can create the SSL key and certificate files with the `openssl` command:

```
$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/
```

Copy

After you enter the command, you will be taken to a prompt where you can enter information about your website. 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.
- **[illegible]s 365**: This option sets the length of time that the certificate will be considered va[illegible] We set it for one year here.

COOKIE PREFERENCES

- **-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`. You need to enter either the hostname you'll use to access the server by, or the public IP of the server. It's important that this field matches whatever you'll put into your browser's address bar to access the site, as a mismatch will cause more security errors.

The entirety of the prompts will look something like this:

```
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.

# Step 3 – Configure Apache to Use SSL

Now that we have our self-signed certificate and key available, we need to update our Apache configuration to use them. On Ubuntu, you can place new Apache configuration files (they must end in `.conf`) into `/etc/apache2/sites-available/` and they will be loaded the next time the Apache process is reloaded or restarted.

For this tutorial we will create a new minimal configuration file. (If you already have an Apac[illegible]`rtualhost>` set up and just need to add SSL to it, you will likely need to copy over the configuration lines that start with `SSL`, and switch the `VirtualHost` port from `80` to `443`. We will take care of port `80` in the next step.)

COOKIE PREFERENCES

Open a new file in the /etc/apache2/sites-available directory:

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

Copy

Paste in the following minimal VirtualHost configuration:

/etc/apache2/sites-available/your_domain_or_ip.conf

```
<VirtualHost *:443>
   ServerName your_domain_or_ip
   DocumentRoot /var/www/your_domain_or_ip

   SSLEngine on
   SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
   SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key
</VirtualHost>
```

Be sure to update the `ServerName` line to however you intend to address your server. This can be a hostname, full domain name, or an IP address. Make sure whatever you choose matches the `Common Name` you chose when making the certificate.

The remaining lines specify a `DocumentRoot` directory to serve files from, and the SSL options needed to point Apache to our newly-created certificate and key.

Now let's create our `DocumentRoot` and put an HTML file in it just for testing purposes:

```
$ sudo mkdir /var/www/your_domain_or_ip
```

Copy

Open a new `index.html` file with your text editor:

```
$ sudo nano /var/www/your_domain_or_ip/index.html
```

Copy

Paste the following into the blank file:

/var/www/your_domain_or_ip/index.html

```
<h1>it worked!</h1>
```

This isn't a full HTML file, of course, but browsers are lenient and it will be enough to verify our configuration.

Save and close the file Next, we need to enable the configuration file with the `a2ensite` tool:

```
$ sudo a2ensite your_domain_or_ip.conf
```
Copy

It will prompt you to restart Apache to activate the configuration, but first, let's test for configuration errors:

```
$ sudo apache2ctl configtest
```
Copy

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 domai
Syntax OK
```

The first line is a message telling you that the `ServerName` directive is not set globally. If you want to get rid of that message, you can set `ServerName` to your server's 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 reload Apache to implement our changes:

```
$ sudo systemctl reload apache2
```
Copy

Now load your site in a browser, being sure to use `https://` at the beginning.

You should see an error. This is normal for a self-signed certificate! The browser is warning you that it can't verify the identity of the server, because our certificate is not signed by any of its known certificate authorities. For testing purposes and personal use this can be fine. You should be able to click through to **advanced** or **more information** and choose to proceed.

After you do so, your browser will load the `it worked!` message.

N if your browser doesn't connect at all to the server, make sure your connection isn't bei cked by a firewall. If you are using `ufw`, the following commands will open ports `80` and `443`:

```
$ sudo ufw allow "Apache Full"
```

Copy

Next we will add another `VirtualHost` section to our configuration to serve plain HTTP requests and redirect them to HTTPS.

# Step 4 – Redirecting HTTP to HTTPS

Currently, our configuration will only respond to HTTPS requests on port `443`. It is good practice to also respond on port `80`, even if you want to force all traffic to be encrypted. Let's set up a `VirtualHost` to respond to these unencrypted requests and redirect them to HTTPS.

Open the same Apache configuration file we started in previous steps:

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

Copy

At the bottom, create another `VirtualHost` block to match requests on port `80`. Use the `ServerName` directive to again match your domain name or IP address. Then, use `Redirect` to match any requests and send them to the SSL `VirtualHost`. Make sure to include the trailing slash:

/etc/apache2/sites-available/your_domain_or_ip.conf

```
<VirtualHost *:80>
    ServerName your_domain_or_ip
    Redirect / https://your_domain_or_ip/
</VirtualHost>
```

Save and close this file when you are finished, then test your configuration syntax again, and reload Apache:

```
$ sudo apachectl configtest
$ sudo systemctl reload apache2
```

Copy

You can test the new redirect functionality by visiting your site with plain `http://` in front of the address. You should be redirected to `https://` automatically.

# Conclusion

COOKIE PREFERENCES

You have now configured Apache to serve encrypted requests using a self-signed SSL certificate, and to redirect unencrypted HTTP requests to HTTPS.

If you are planning on using SSL for a public website, you should look into purchasing a domain name and using a widely supported certificate authority such as Let's Encrypt.

For more information on using Let's Encrypt with Apache, please read our How To Secure Apache with Let's Encrypt on Ubuntu 16.04 tutorial.

About the authors

Justin Ellingwood Author

Comments

10 Comments

Leave a comment...

This textbox defaults to using `Markdown` to format your answer.

You can type `!ref` in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

Tayskeno • June 28, 2016

Hello,

I am getting this kind of error : Could you help to identify the issue ?

```
AH00526: Syntax error on line 12 of /etc/apache2/conf-enabled/ssl-params
Invalid command 'SSLSessionTickets', perhaps misspelled or defined by a r
Action 'configtest' failed.
The Apache error log may have more information.
```

Reply

Debiprasad • June 23, 2016

Why am I getting the following error?

```
AH00526: Syntax error on line 12 of /etc/apache2/conf-enabled/ssl-params
Invalid command 'SSLSessionTickets', perhaps misspelled or d
```

COOKIE PREFERENCES

```
Action 'configtest' failed.
The Apache error log may have more information.
```

Show replies Reply

TrevorLaneRay • May 3, 2016

Great tutorial; lots of details & easy to follow. Probably should mention we can avoid using self-signed certs, and use Let’sEncrypt CA.

Show replies Reply

ribafs • October 8, 2020

Very good. Thank you.

Reply

ken0a0cbe7482242d334b7229e • June 19, 2019

I got an issue, After applying this certificate. My soapclient is stop working. it is throwing error “could not connect to host”. I try to find but nothing helped me. If you guys have any idea about.

Thanks in advance,

Reply

umar Ghuraiya • June 18, 2019

When you can't install or afford trusted certificates from a certificate authority, you may get by with self-signed certificates. Both trusted, and self-signed certificates are the same and use the same protocols… the only difference is, one is trusted by a third party, and the other is not.

When you're ready, run the commands below to generate the private server key as well as the self-signed SSL/TLS certificate for the chiragpatel.com domain… you'll be using.

Note: chiragpatel.com is my server name

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

After running the commands above, you'll be prompted to answer a few questions about the certificate you're generating… answer them and complete the process.

# Generating a 2048 bit RSA private key ...+++ ... +++ writing new private key to 'mydomain.key'

# You are about to be asked to enter information that will be incorporated into your certificate request. What you are about to enter is what is called a Distinguished Name or a DN. There are quite a few fields but you can leave some blank For some fields there will be a default value, If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:IN State or Province Name (full name) [Some-State]:Gujarat Locality Name (eg, city) []:Vadodara Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example Company Organizational Unit Name (eg, section) []:SSL Unit Common Name (e.g. server FQDN or YOUR name) []:chiragpatel.com Email Address []:*<email removed by mod>*

Please enter the following 'extra' attributes to be sent with your certificate request A challenge password []: LEAVE BLANK An optional company name []:

COOKIE PREFERENCES

Reply

Rick Graves • May 13, 2019

I followed the steps on “How To Serve Django Applications with Apache and mod_wsgi on Ubuntu 16.04”

That directed me here. I followed the steps here. The https part was working, but I was getting the Apache2 default “It works!” page, not my Django app. I finally got my Django app to show up by moving the directory and WSGI lines from 000-default.conf to default-ssl.conf

Reply

djos • January 7, 2019

Hi, you can check up my video where I show in 10 minutes or so how to setup a free multidomain ssl certificate with certbot on ubuntu.

https://youtu.be/Obti3LjGR7o

Happy to contribute to the DO community,

All the best,

José from France

Reply

peter8ba74a39bb7bb7e0b846d • November 14, 2018

A remark on ssl-params.conf.

r using wordpress you got problems with X-frame which is used by some pl s. It cost me a couple of hours to find out how to correct this.

COOKIE PREFERENCES

Change the line:

- Header always set X-Frame-Options DENY*

in *Header set X-Frame-Options: “sameorigin”*

or *Header always set X-Frame-Options "allow-from https://<wordpress site>/*

Btw A great tutorial. Thank you for that.

Reply

mallimatla • November 7, 2018

Hello, Thanks for making this documentation. I’m able to do the Self Signed SSL to Apache and able to access like https://123.456.12.43, it is showing tomcat server. But https is not working when i try along with port or application like https://123.456.12.43:8080 or https://123.456.12.43:8080/sample-application/ We have placed our application in the /sample-application directory.

Appreciate if any quick help or suggestions. Thank you very much.

Reply

Load More Comments

COOKIE PREFERENCES

Popular Topics

Ubuntu

Linux Basics

JavaScript

Python

MySQL

Docker

Kubernetes

All tutorials →

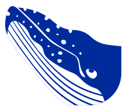
Free Managed Hosting →

Congratulations on unlocking the whale ambience easter egg! Click the whale button in the bottom left of your screen to toggle some ambient whale noises while you read.

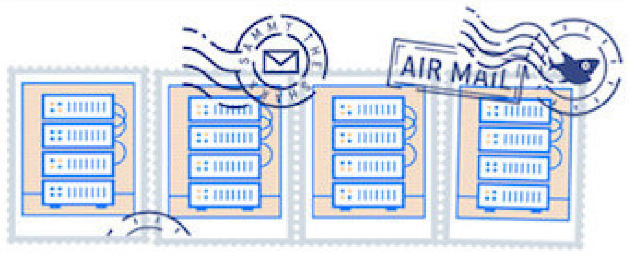
Thank you to the Glacier Bay National Park & Preserve and Merrick079 for the sounds behind this easter egg.

Interested in whales, protecting them, and their connection to helping prevent climate change? We recommend checking out the Whale and Dolphin Conservation.

Reset easter egg to be discovered again / Permanently dismiss and hide easter egg



COOKIE PREFERENCES



Get our biweekly newsletter

Sign up for Infrastructure as a Newsletter.

Sign up →

Hollie's Hub for Good

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

Learn more →

Become a contributor

You get paid; we donate to tech nonprofits.

Learn more →

Featured on Community

COOKIE PREFERENCES

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

DigitalOcean Products

Cloudways Virtual Machines Managed Databases Managed Kubernetes
Block Storage Object Storage Marketplace VPC Load Balancers

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 →

Company	Products	Community	Solutions	Contact
About	Products Overview	Tutorials	Website Hosting	Support
Leadership	Droplets	Q&A	VPS Hosting	Sales
Blog	Kubernetes	CSS-Tricks	Web & Mobile Apps	Report Abuse
Careers	App Platform	Write for DOnations	Game Development	System Status
Customers				

COOKIE PREFERENCES

Partners

Channel Partners

Referral Program

Affiliate Program

Press

Legal

Security

Investor Relations

DO Impact

Functions

Cloudways

Managed Databases

Spaces

Marketplace

Load Balancers

Block Storage

Tools & Integrations

API

Pricing

Documentation

Release Notes

Uptime

Currents Research

Hatch Startup Program

deploy by DigitalOcean

Shop Swag

Research Program

Open Source

Code of Conduct

Newsletter Signup

Meetups

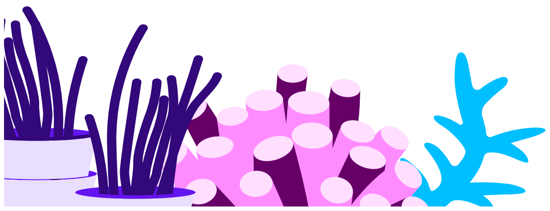
Streaming

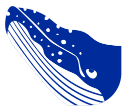
VPN

SaaS Platforms

Cloud Hosting for Blockchain

Startup Resources





COOKIE PREFERENCES