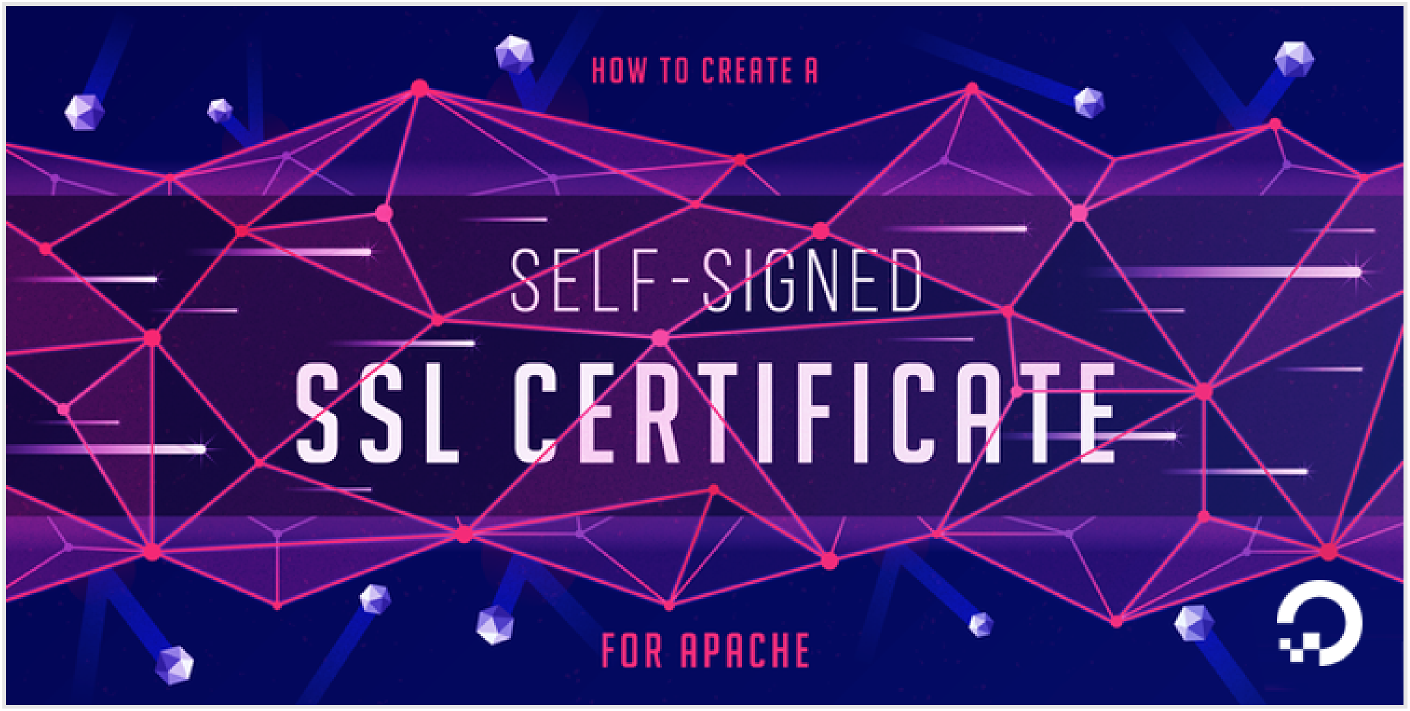
DigitalOcean

Subscribe Share Contents



How To Create a Self-Signed SSL Certificate for Apache in Debian 9

1

Posted September 7, 2018 4.8k APACHE SECURITY DEBIAN 9 DEBIAN

By: Justin Ellingwood By: Brian Boucheron By: Mark Drake

Not using **Debian 9**? Choose a different version:

Introduction

kets layer, are

Using this technology, servers can send traffic safely between servers and clients without the possibility of 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, we will show you how to set up a self-signed SSL certificate for use with an Apache web server on Debian 9.

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 Debian 9.

You will also need to have the Apache web server installed. If you would like to install an entire LAMP (Linux, Apache, MariaDB, PHP) stack on your server, you can follow our guide on setting up LAMP on Debian 9. If you just want the Apache web server, skip the steps pertaining to PHP and MariaDB.

When you have completed these 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.

We can create a self-signed key and certificate pair with OpenSSL in a single command:

```
e-selfsigned.cr
```

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.**

The entirety of the prompts will look something like this:

```
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:

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

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

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 to this and 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 following 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 inc
# the "preload" directive if you understand the implications.
# Header always set Strict-Transport-Security "max-age=63072000; includeSubDomains; pr
Header always set X-Frame-Options DENY
Header always set X-Content-Type-Options nosniff
# Requires Apache >= 2.4
SSLCompression off
SSLUseStapling on
SSLStaplingCache "shmcb:logs/stapling-cache(150000)"
# Requires Apache >= 2.4.11
SSLSessionTickets Off
```

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/d
```

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 block 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)$">
                                SSLOptions +StdEnvVars
                </FilesMatch>
                <Directory /usr/lib/cgi-bin>
                                SSLOptions +StdEnvVars
```

```
</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. Again, if you're using a different document root, be sure to update the `DocumentRoot` directive.

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

S traffic. For

:ally. 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, 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, 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.

That's all of the configuration changes you need to make to Apache. Next, we will discuss how to update firewall rules with `ufw` to allow encrypted HTTPS traffic to your server.

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. Fortunately, when installed on Debian 9, `ufw` comes loaded with app profiles which you can use to tweak your firewall settings

We can see the available profiles by typing:

```
$ sudo ufw app list
```

You should see a list like this, with the following four profiles near the bottom of the output:

```
Available applications:
. . .
  WWW
  WWW Cache
  WWW Full
  WWW Secure
. . .
```

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
WWW                        ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
WWW (v6)                   ALLOW       Anywhere (v6)
```

To additionally let in HTTPS traffic, allow the "WWW Full" profile and then delete the redundant "WWW" profile allowance:

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

Your status should look like this now:

```
$ sudo ufw status
```

```
Output
```

```
--                          ------      ----
OpenSSH                     ALLOW       Anywhere
WWW Full                    ALLOW       Anywhere
OpenSSH (v6)                ALLOW       Anywhere (v6)
WWW Full (v6)               ALLOW       Anywhere (v6)
```

With your firewall configured to allow HTTPS traffic, you can move on to the next step where we'll go over how to enable a few modules and configuration files to allow SSL to function properly.

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 then restart Apache to put these changes into effect.

Enable `mod_ssl`, the Apache SSL module, and `mod_headers`, which is needed by some of the settings in our SSL snippet, with the `a2enmod` command:

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

Next, enable your SSL Virtual Host with the `a2ensite` command:

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

You will also need to enable your `ssl-params.conf` file, to read in the values you've set:

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

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

```
$ sudo apache2ctl configtest
```

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

```
Syntax OK
```

As long as your output has `Syntax OK` in it, then your configuration file has no syntax errors and you can safely restart Apache to implement the changes:

```
$ sudo systemctl restart apache2
```

With that, your self-signed SSL certificate is all set. You can now test that your server is correctly encrypting its traffic.

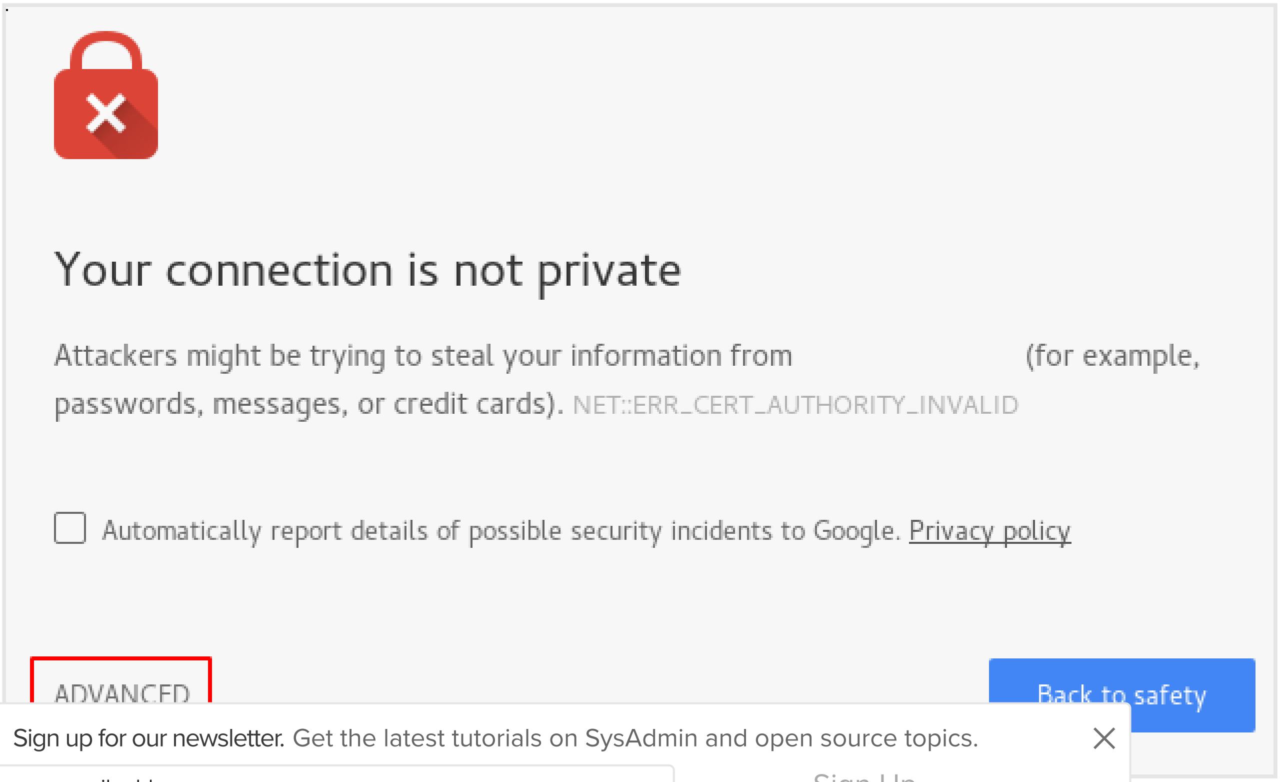
Step 5 — Testing Encryption

You're now ready to test your 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 you 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:

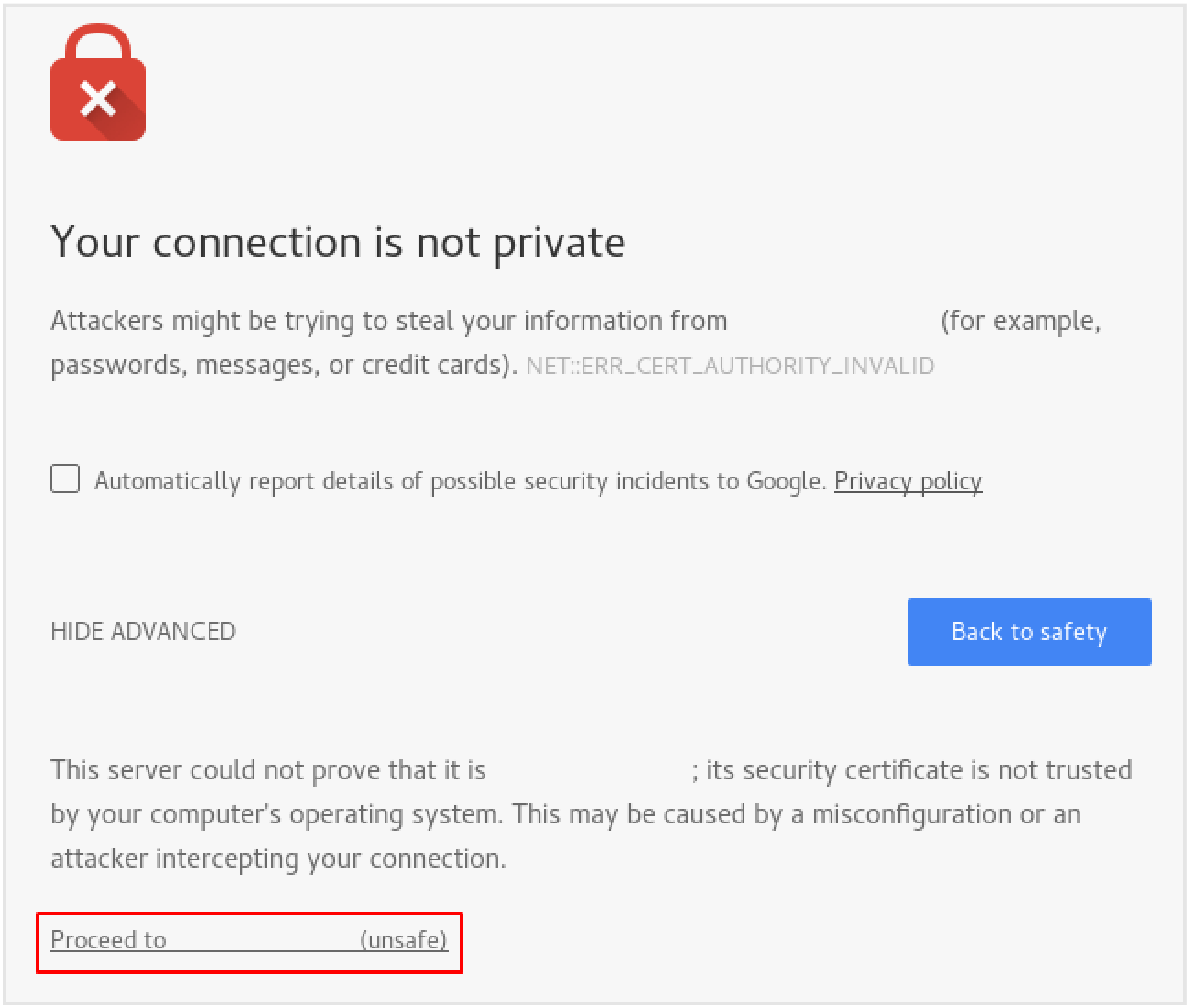


Your connection is not private

Attackers might be trying to steal your information from (for example, passwords, messages, or credit cards). NET::ERR_CERT_AUTHORITY_INVALID

☐ Automatically report details of possible security incidents to Google. Privacy policy

This is expected and normal. We are only interested in the encryption aspect of our certificate, not the third party validation of our host's authenticity. Click **ADVANCED** and then the link provided to proceed to your host anyways:



Your connection is not private

Attackers might be trying to steal your information from (for example, passwords, messages, or credit cards). NET::ERR_CERT_AUTHORITY_INVALID

☐ Automatically report details of possible security incidents to Google. Privacy policy

This server could not prove that it is ; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

Proceed to (unsafe)

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 or another similar “not secure” notice. 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. However, the redirect

HTTPS

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
```

If this command doesn't report any syntax errors, restart Apache:

```
$ sudo systemctl restart apache2
```

This will make the redirect permanent, and your site will only serve traffic over HTTPS.

Conclusion

You have configured your Apache server to use strong encryption for client connections. This will traffic.

By: Justin Ellingwood By: Brian Boucheron By: Mark Drake

Upvote (1) Subscribe Share

Related Tutorials

How To Set Up Apache with a Free Signed SSL Certificate on a VPS

How To Secure Nginx with NAXSI on Ubuntu 16.04

How To Install and Configure pgAdmin 4 in Server Mode

How To Install Webmin on Debian 9

How To Set Up an OpenVPN Server on Debian 9

0 Comments

Log In to Comment

Community Tutorials Questions Projects Tags Newsletter RSS

Distros & One-Click Apps Terms, Privacy, & Copyright Security Report a Bug Write for DOnations Shop

Sign up for our newsletter. Get the latest tutorials on SysAdmin and open source topics.

Enter your email address

Sign Up