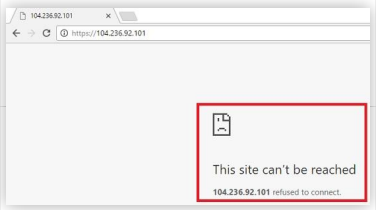
How to Create a Self Signed Certificate on a Linux Server

Posted on Updated on May 17, 2018

Earlier, we'd looked at how to install Apache on a clean Linux install directly from the command line and have it start up on boot. While this works great for normal requests, it doesn't process SSL requests. Meaning that if someone types in "https" instead of "http", you'll get an error that looks like this:



HTTPS is more important than ever, and it will continue to grow in usage until virtually the entire web is encrypted. Google is pushing hard for HTTPS and has promised to provide preferential rankings for secure sites as opposed to insecure ones. Regardless of how you may might feel about this (personally I think Google needs to back off – some sites have no use for HTTPS), it's important to keep up with changing trends and transition your site to SSL. In this tutorial, we'll look at how to configure your site to accept HTTPS requests via self signed certificates.

Keep in mind, that this is for demo purposes only. In reality, you'll need to send your Certificate Signing Request (csr) file to a registered CA so they can verify your site independently. But you can also self sign your requests – meaning that most browsers will display a huge security warning to your users. This can be bypassed if needed, but if you have a production site, keep in mind that you'll have to eventually send your certificate for authentication (for a fee).

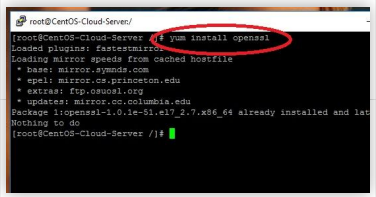
Step 1: Making sure the Right Software is Installed

I was going to write this article using the "genkey" utility via the "crypto-utils" package, but realized that it doesn't work over Putty using Windows as a client. The reason is that it needs to generate random data for creating the key, and it requires mouse or keyboard input over the server console. Unfortunately, this doesn't work over a remote connection via SSH – you have to be directly connected to the server. So despite it having a nice and easy to use GUI, I can't recommend the genkey tool to generate a certificate for your server.

So instead, we're just going to go with "openssl". Chances are that it's already installed on your server, but there's no harm in verifying that you have the latest update files. If you're using CentOS, then type in :

```
yum install openssl
```

With Ubuntu, you'll need to use "apt-get" instead of "yum". Modify this for your own version of Linux. If openssl is already installed, you'll get something like the screenshot below. If not, it'll be installed:



Step 2: Create the Key and Certificate Directory

We'll need a defined location to place our key and certificate files. Well...actually we don't, but it's nice to know where they are in case you have to find them later. Technically you can just keep everything in your root directory, but that's not organized. Since these are dependent on apache, we'll create a directory called "keycertificate" in the apache installation folder. Run the following command:

```
mkdir /etc/httpd/keycertificate
```

Now that we have a place to put our key and certificate, we can finally generate the certificate!

Step 3: Determine your Organization Name or IP Address

When you create your certificate, you'll be asked to provide several details – one of the most important is your domain name or IP address. If you want to submit a certificate request to a CA, then you need to make sure that you provide the main domain name of your site – not an alias for example. For our purposes, you can even use the IP address of your server. As long as you remain consistent with it – because you'll have to enter it in a couple of places.

For this example, I'm using my server IP address.

Step 4: Determining the Level of Encryption

When you generate your private key, you have to determine what level of encryption you want to use. The recommended key length is 2048 bits. Higher security keys like 4096 bits have a slower server response. Technically, you can even use something as low as 512 bits for maximum speed, but browsers these days will display a warning if the certificate is linked to a low security key.

Chrome for example, will reject the request outright without any additional information. Firefox however, displays the following warning:

For testing purposes, I wanted to first use a weak key to see what would happen, and it took me a while to figure out the problem! So if you want modern browsers to recognize your site, make sure you use a key with at least 2048 bit security. Anything less will set off errors.

Step 5: Generating the Key and Certificate

Now we come down to it. We're going to perform the following tasks:

1. Generate a key pair
2. Extract the private key
3. Delete the initial key pair
4. Create a certificate signing request
5. Create a self signed certificate

These four steps are carried out by the following four commands:

```
openssl genrsa -des3 -passout pass:x -out serve
openssl rsa -passin pass:x -in server.pass.key
rm server.pass.key
openssl req -new -key /etc/httpd/keycertificate
openssl x509 -req -days 365 -in /etc/httpd/keyc
```

In the above example, replace [server name/IP] with the name you decided on in step 3. The fourth command will ask you a bunch of questions about your organization. Since this is a self signed certificate, you can leave most of the fields blank. However, when it asks you for your "common name", make sure you enter your server's domain/IP address as determined before:

```
[root@CentOS-Cloud-Server keycertificate]# openssl req -new -key /etc
You are about to be asked to enter information that will be incorpora
into your certificate request.
What you are about to enter is what is called a Distinguished Name or
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) [XX]:US
State or Province Name (full name) []:Florida
Locality Name (eg, city) [Default City]:
Organization Name (eg, company) [Default Company Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname)[ :104.236.92.101
Email Address []:

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

The final command will use the CSR to create a self signed certificate and place it in the "keycertificate" folder that we created in Step 2.

Step 6: Configure Apache to use the Certificate and Key

So we have our key and certificate. But by itself this means nothing. Apache still doesn't know that we have these files. So now we have to modify the Apache configuration to tell us where the files are located. Open the following Apache configuration file like this:

```
vi /etc/httpd/conf.d/ssl.conf
```

Or you can use your favorite text editor instead of "vi". Either way, search for these two lines starting with:

```
SSLCertificateFile
```

and

```
SSLCertificateKeyFile
```

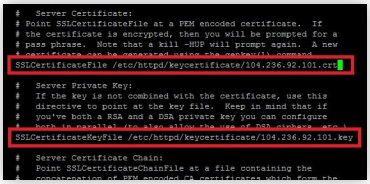
And plug in the paths to the key file and certificate like this:

```
SSLCertificateFile /etc/httpd/keycertificate/[s
```

and

```
SSLCertificateKeyFile /etc/httpd/keycertificate
```

Make sure you use the same domain name or IP address you determined in Step 3. Here's a screenshot to show you what it looks like:



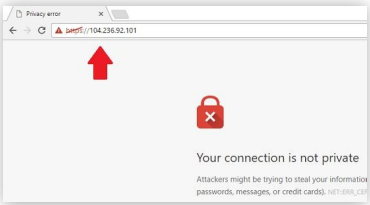
Save the changes to the SSL config file.

Step 7: Restart Apache

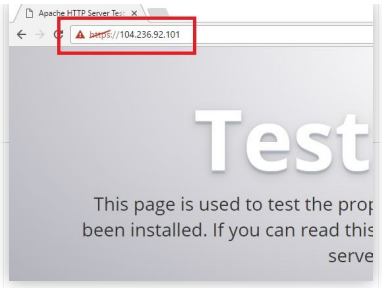
The final step is to restart Apache to make our changes permanent. To do this, type in the following:

```
apachectl restart
```

After it's completed, you're all done! Now open up a browser and visit your site using "https" instead of "http". At the beginning of the tutorial, we saw how it generated an error. This time, you should see something like this:



Chrome recognizes that it's a self signed certificate and generates a warning. If you click "Advanced", you'll be allowed to ignore the error and proceed, and your page should show as usual:



Step 8: Submit your CSR File to a CA

We generated a .csr file where we had to enter all our details, e-mail ID etc. That file can be sent to a Certificate Authority (CA) in order to obtain an authorized certificate that we can place onto our server so that browsers don't display a warning like they do above. CAs charge a fee depending on how

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

Website

Post Comment

About this archive

The content from this archive is provided for reference purposes only and will no longer be updated.