**Rails Local Development over HTTPS using a Self-Signed SSL Certificate**

<https://madeintandem.com/blog/rails-local-development-https-using-self-signed-ssl-certificate>

*last updated August 29, 2017*

I found myself in a peculiar situation recently. I was integrating a Single Sign On workflow with Google as the IdP (read more about it [here](https://madeintandem.com/blog/configuring-rails-app-single-sign-saml-multiple-providers/)). After authenticating on Google, a callback url is provided to handle the response. Google requires that the connection be encrypted (read HTTPS). But, the standard Rails server boots without SSL in development mode. I needed to test the integration locally, so I needed to get my local server secured with SSL. Quite the conundrum!

Let’s look at the server startup logs quickly…

$> rails s

=> Booting Puma

=> Rails 5.1.3 application starting in development on http://localhost:3000

=> Run `rails server -h` for more startup options

Puma starting in single mode...

\* Version 3.9.1 (ruby 2.4.1-p111), codename: Private Caller

\* Min threads: 5, max threads: 5

\* Environment: development

\* Listening on tcp://0.0.0.0:3000

Use Ctrl-C to stop

… and a basic request to localhost:3000

Started GET "/" for 127.0.0.1 at 2017-08-24 11:49:01 -0500

Processing by HomeController#index as HTML

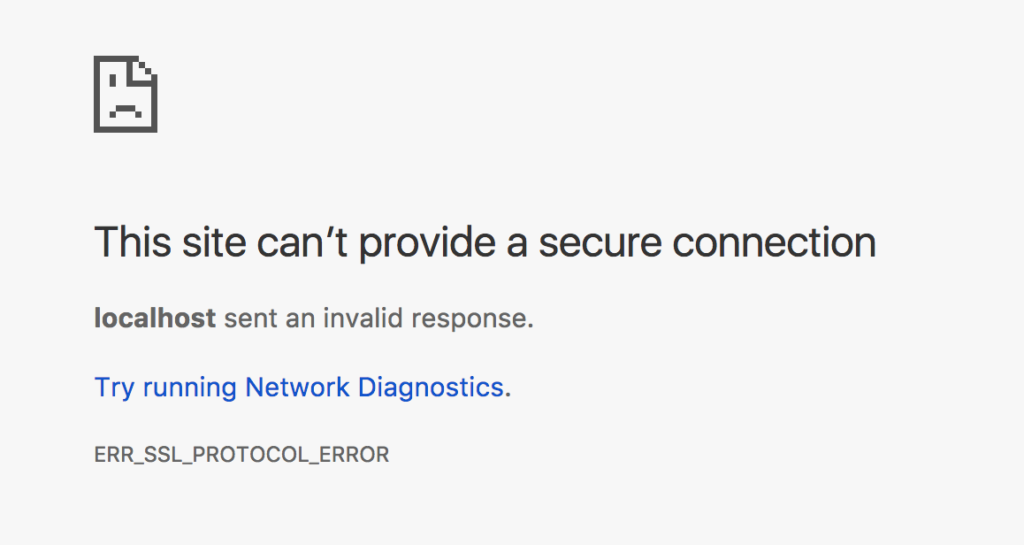
Rendering home/index.html.erb within layouts/application

Rendered home/index.html.erb within layouts/application (0.3ms)

Completed 200 OK in 15ms (Views: 13.5ms | ActiveRecord: 0.0ms)

Good stuff. Notice the output from Puma, specifically this line Listening on tcp://0.0.0.0:3000. We’ll later compare this to the logs when we boot with SSL.

My first idea was to dig into the [Rails configuration options](http://guides.rubyonrails.org/configuring.html#rails-general-configuration). I added config.force\_ssl = true to the bottom of my development.rb config file. After restarting the server and visiting localhost:3000 with Chrome, the https:// was automatically prepended to my url. Chrome was mad:



And the server was mad:

2017-08-24 11:55:39 -0500: HTTP parse error, malformed request (): #<Puma::HttpParserError: Invalid HTTP format, parsing fails.>

After some reading, I started looking into how to generate a self-signed SSL certificate for my localhost server. Turns out that **openssl** has a built in command for exactly this. [This post](https://stackoverflow.com/questions/10175812/how-to-create-a-self-signed-certificate-with-openssl) was extremely helpful to understand the options. Here’s how to generate the cert:

$> openssl req -x509 -sha256 -nodes -newkey rsa:2048 -days 365 -keyout localhost.key -out localhost.crt

You’ll be prompted with some information on country code, email, etc. but you can skip all of the questions. This command will create two new files in the current directory localhost.key and localhost.crt. Put those wherever you want.

What do we do with these files? I found out from the [Rails server docs](http://guides.rubyonrails.org/command_line.html#rails-server) that the -b options binds the server to a specific IP. Restarting the server with a binding:

$> rails s -b 'ssl://localhost:3000?key=path/to/file/localhost.key&cert=path/to/file/localhost.crt'

=> Booting Puma

=> Rails 5.1.3 application starting in development on http://ssl://localhost:3000?key=localhost.key&cert=localhost.crt:3000

=> Run `rails server -h` for more startup options

Puma starting in single mode...

\* Version 3.9.1 (ruby 2.4.1-p111), codename: Private Caller

\* Min threads: 5, max threads: 5

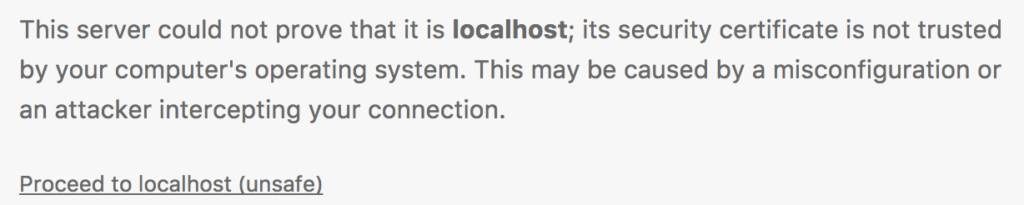
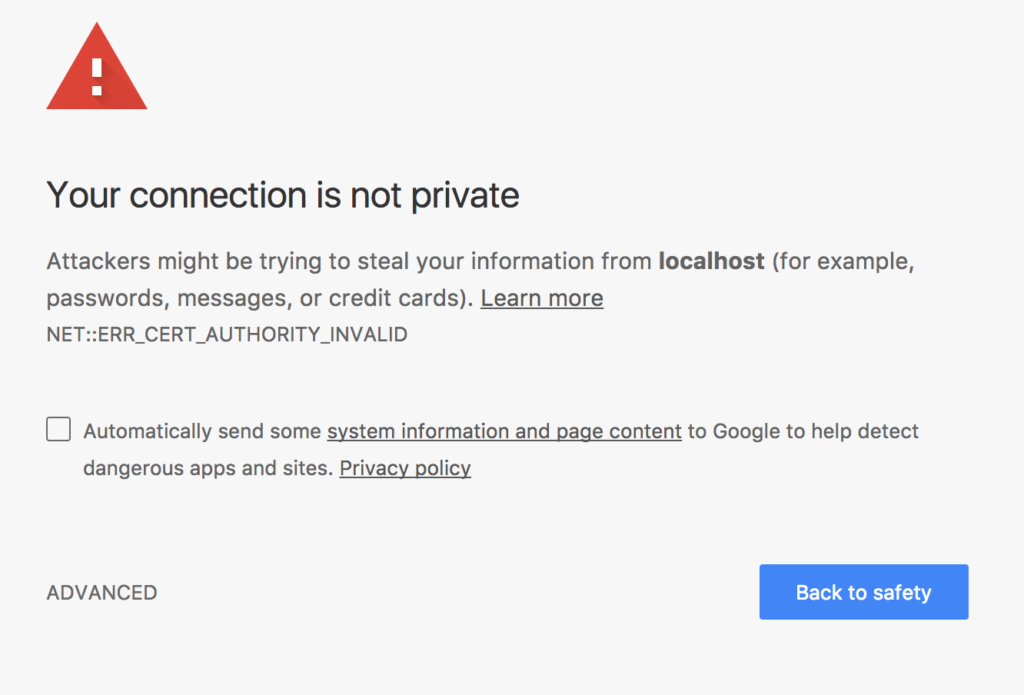
\* Environment: development

\* Listening on ssl://localhost:3000?key=localhost.key&cert=localhost.crt

Use Ctrl-C to stop

Notice the line Listening on ssl://localhost:3000?key=localhost.key&cert=localhost.crt. It appears that our local server is now secured with over HTTPS!

Visiting https://localhost:3000

https://madeintandem.com/wp-content/uploads/2017/08/Screen-Shot-2017-08-24-at-11.41.42-AM.png

Lots of **red** but almost there! The final step was to simply click *ADVANCED*and to tell Chrome to trust the server by clicking *Proceed to localhost (unsafe)*.

Rails provides so many nice configuration options to make the development experience similar to the production experience. Running the server locally over HTTPS was a unique challenge.