Setting up HTTPS locally can be tricky business. Even if you do manage to wrestle self-signed certificates into submission, you still end up with browser privacy errors. In this article, we'll walk through creating your own Certificate Authority for your local servers so that you can run HTTPS sites locally without issue.

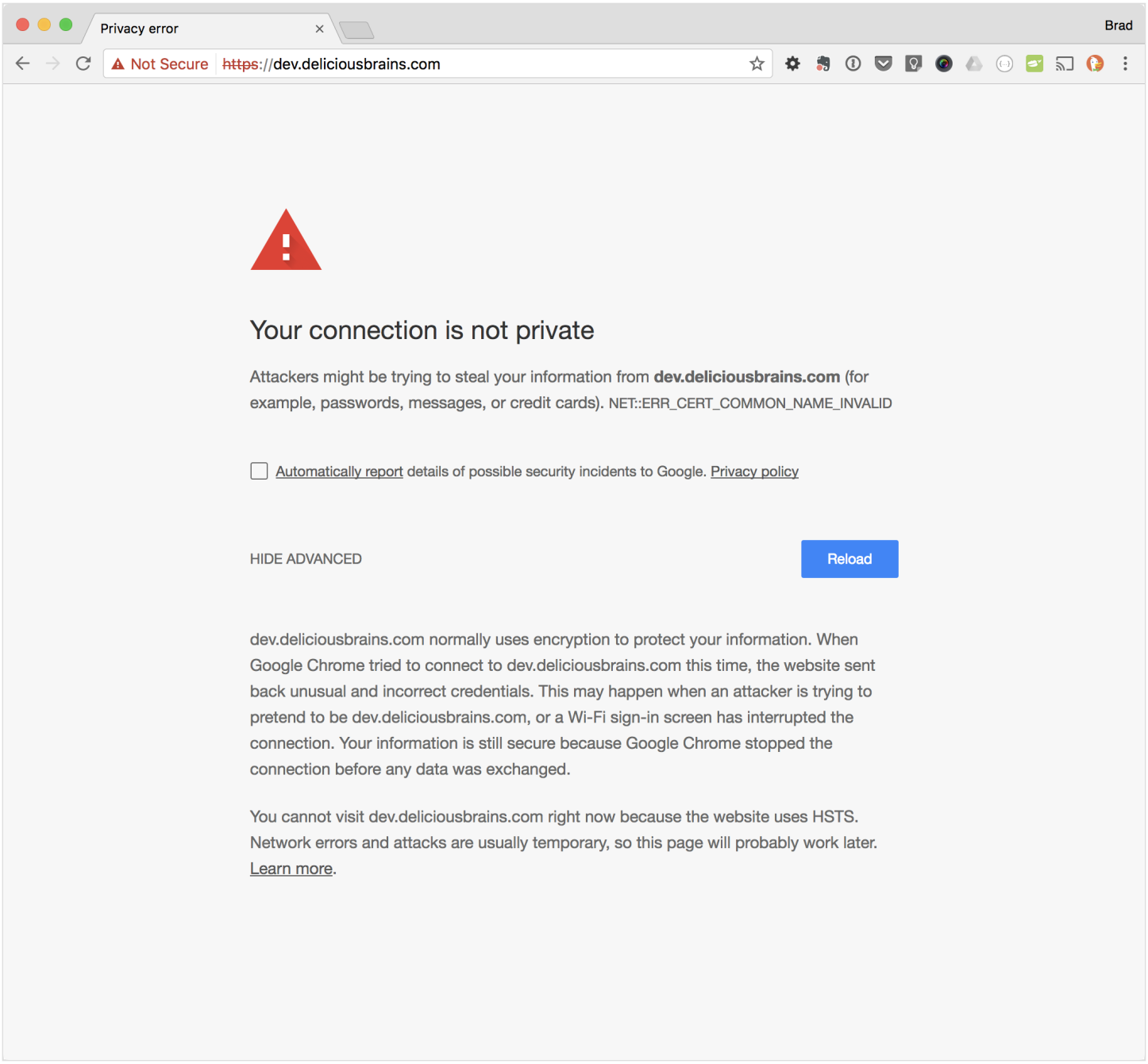
Why HTTPS Locally?

Why not just use regular HTTP locally? Because if your production site is HTTPS-only and you're developing locally on regular HTTP, your dev and production environments are not as similar as they could be. For example, my dev environment for this site (deliciousbrains.com) runs as an Ubuntu server in a VMware virtual machine (VM) on his Mac. The production site is an Ubuntu server running on Linode with an almost identical configuration.

You definitely want your dev environment to mirror production as closely as possible.
When it doesn't, you invite more issues showing up in production that didn't show up in dev. Running HTTP when your production site is HTTPS-only is definitely an unnecessary risk.

However, trying to get an SSL certificate working with your local server kind of sucks if you're not using a tool that handles it for you like Valet.

If you've ever tried to run an HTTPS site locally, you've probably seen something like the following in Chrome:



The workaround *used to be* creating a self-signed certificate and using that. MAMP Pro does this for you and was my go-to for years. Unfortunately, that's no longer possible. The modern approach is to become your own Certificate Authority (CA)!

How It Works

To request an SSL certificate from a CA like Verisign or GoDaddy, you send them a Certificate Signing Request (CSR), and they give you a certificate in return that they signed using their root certificate and private key. All browsers have a copy (or access a copy from the operating system) of Verisign's root certificate, so the browser can verify that your certificate was signed by a trusted CA.

That's why when you generate a self-signed certificate the browser doesn't trust it. It's self-signed. It hasn't been signed by a CA. But we can generate our own root certificate and private key. We then add the root certificate to all the devices we own just once, and then all certificates that we generate and sign will be inherently trusted.

Becoming a (tiny) Certificate Authority

It's kind of ridiculous how easy it is to generate the files needed to become a certificate authority. It only takes two commands. First, we generate our private key:

```
openssl genrsa -des3 -out myCA.key 2048
```

You will be prompted for a passphrase, which I recommend not skipping and keeping safe. The pass phrase will prevent anyone who gets your private key from generating a root certificate of their own. Output should look like this:

```
Generating RSA private key, 2048 bit long modulus
.....................................................................+++
.....................................+++
e is 65537 (0x10001)
Enter pass phrase for myCA.key:
Verifying - Enter pass phrase for myCA.key:
```

Then we generate a root certificate:

```
openssl req -x509 -new -nodes -key myCA.key -sha256 -days 1825 -out myCA.pem
```

You will be prompted for the passphrase of your private key (that you just chose) and a bunch of questions. The answers to those questions aren't that important. They show up when looking at the certificate, which you will almost never do. I suggest making the Common Name something that you'll recognize as your root certificate in a list of other certificates. That's really the only thing that matters.

```
Enter pass phrase for myCA.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]:CA
State or Province Name (full name) [Some-State]:Nova Scotia
Locality Name (eg, city) []:Truro
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Delicious Brains Inc
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:Delicious Brains
Email Address []:noreply@deliciousbrains.com
```

You should now have two files: myCA.key (your private key) and myCA.pem (your root certificate).

🎉 Congratulations, you're now a CA. Sort of.

To become a real CA, you need to get your root certificate on all the devices in the world. Let's start with the ones you own.

Installing Your Root Certificate

We need to add the root certificate to any laptops, desktops, tablets, and phones that will be accessing your HTTPS sites. This can be a bit of a pain, but the good news is that we only have to do it once. Once our root certificate is on each device, it will be good until it expires.

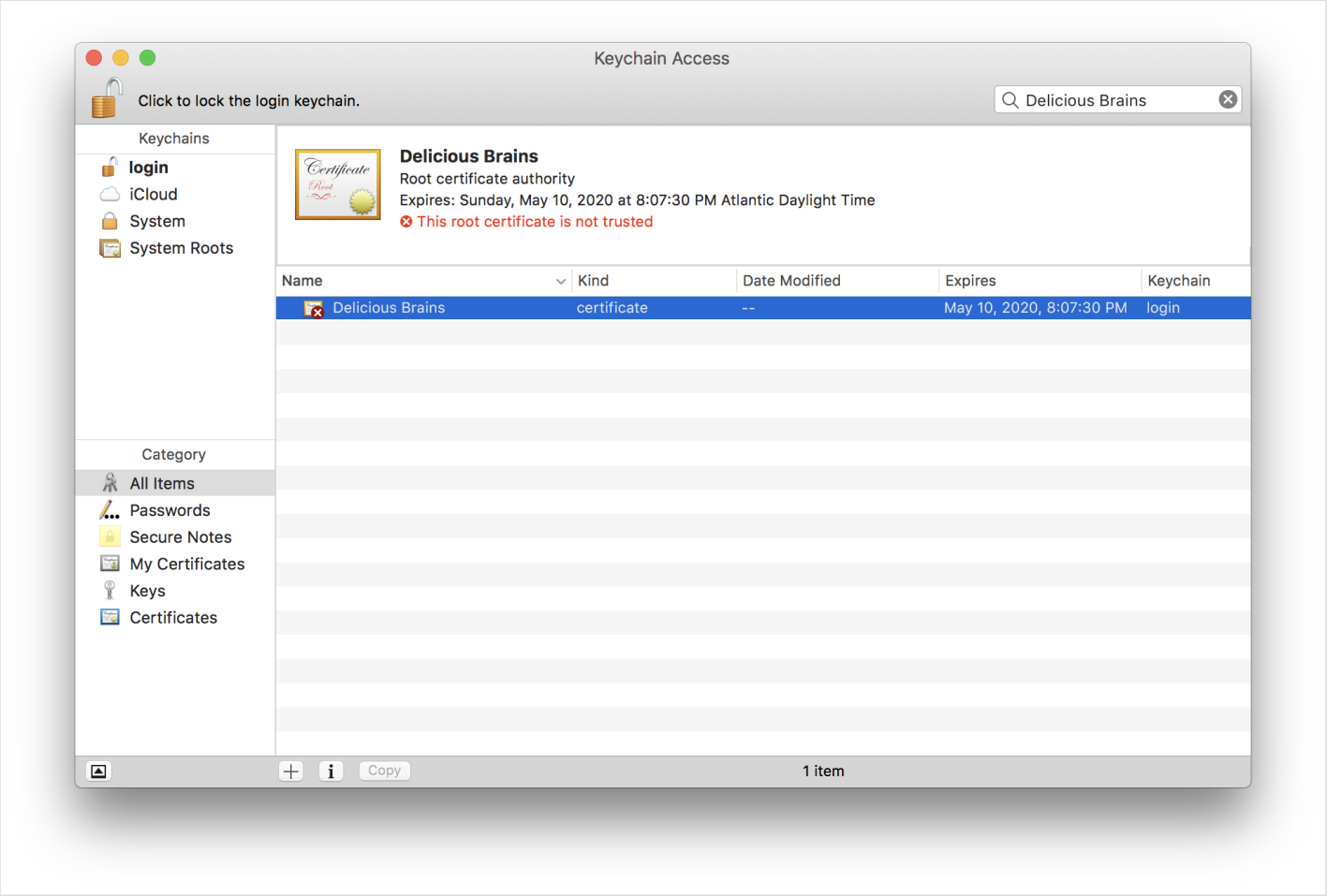
Adding the Root Certificate to macOS Keychain

Via the CLI

```
sudo security add-trusted-cert -d -r trustRoot -k "/Library/Keychains/System.keychain" myCA.pem
```

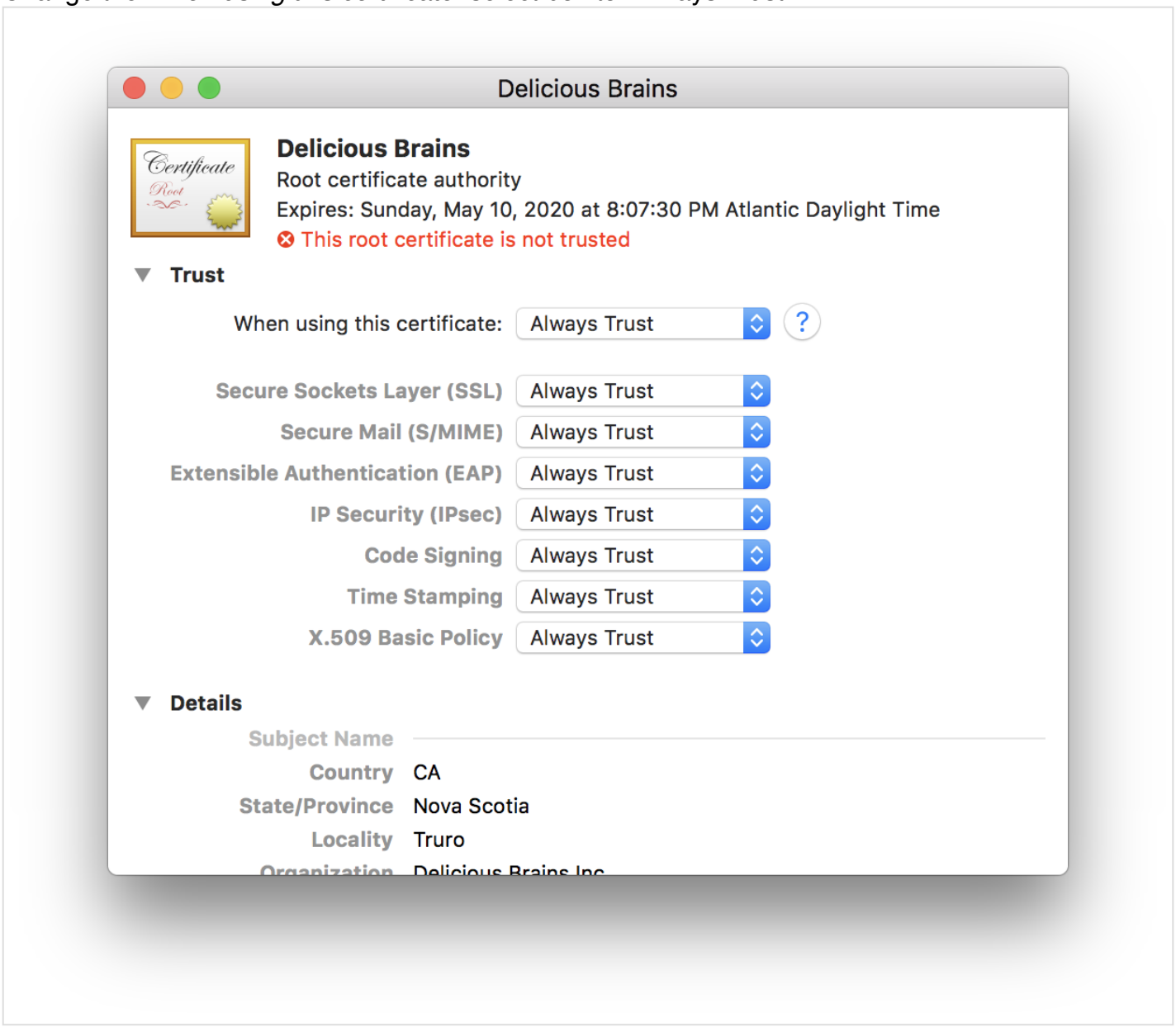
Via the UI

1. Open the macOS Keychain app
2. Go to *File > Import Items…*
3. Select your private key file (i.e. myCA.pem)
4. Search for whatever you answered as the *Common Name* name above



5. Double click on your root certificate in the list
6. Expand the *Trust* section

7. Change the *When using this certificate:* select box to “Always Trust”

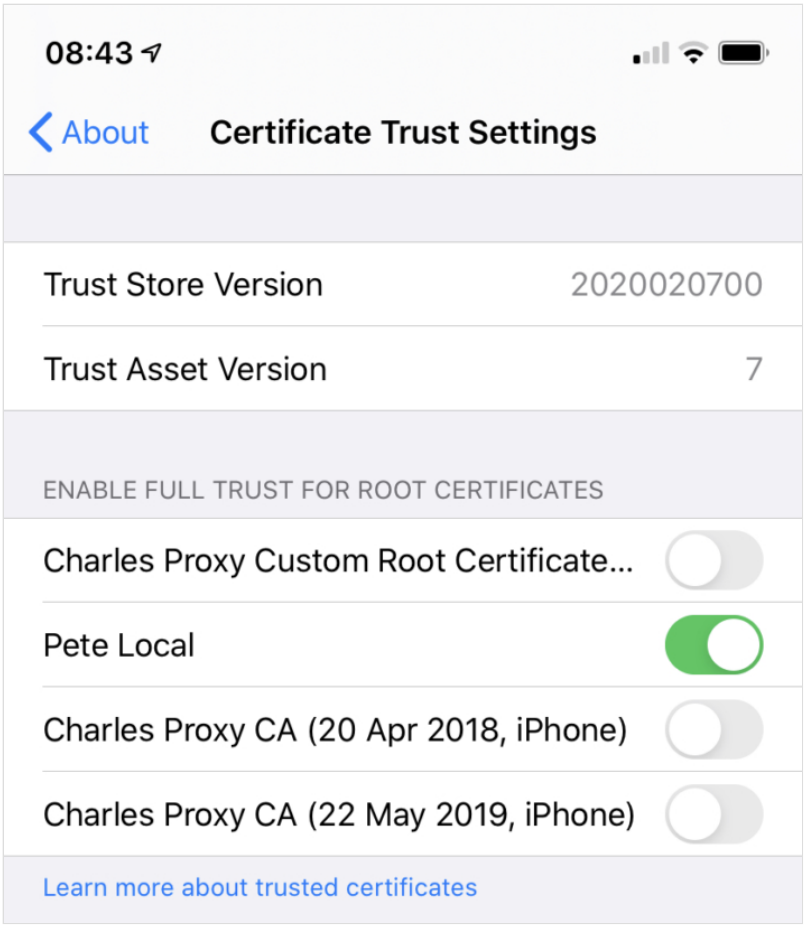


8. Close the certificate window
9. It will ask you to enter your password (or scan your finger), do that
10. 🎉 Celebrate!

Adding the Root Certificate to iOS

If you’d like to add the root certificate to your iOS devices, you can do so fairly easily by following these steps:

1. Email the root certificate to yourself so you can access it on your iOS device
2. Click on the attachment in the email on your iOS device
3. Go to the settings app and click ‘Profile Downloaded’ near the top
4. Click install in the top right
5. Once installed, hit close and go back to the main Settings page
6. Go to “General” > “About”
7. Scroll to the bottom and click on “Certificate Trust Settings”
8. Enable your root certificate under “ENABLE FULL TRUST FOR ROOT CERTIFICATES”



Creating CA-Signed Certificates for Your Dev Sites

Now that we're a CA on all our devices, we can sign certificates for any new dev sites that need HTTPS. First, we create a private key:

```
openssl genrsa -out dev.deliciousbrains.com.key 2048
```

Then we create a CSR:

```
openssl req -new -key dev.deliciousbrains.com.key -out dev.deliciousbrains.com.csr
```

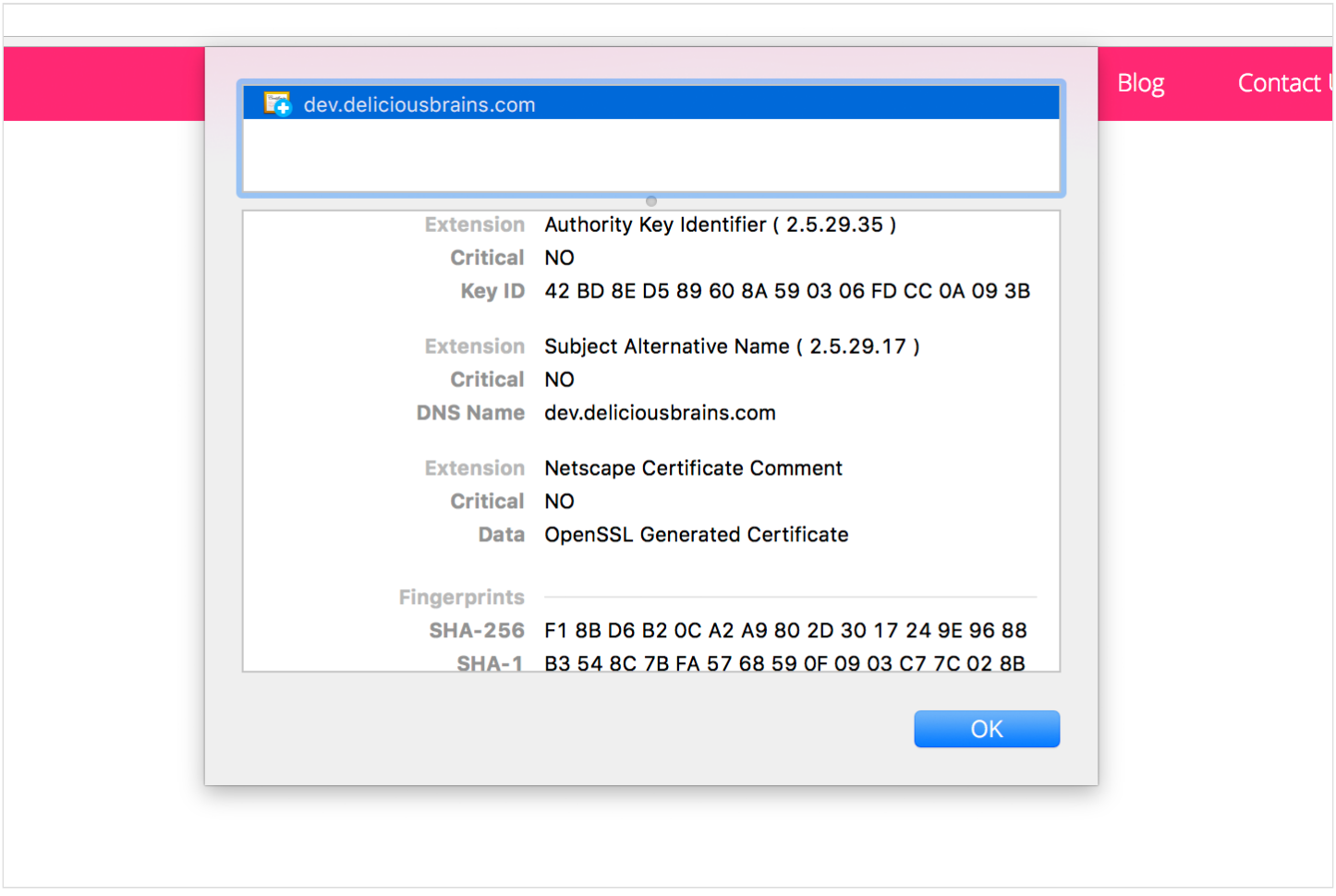
You'll get all the same questions as you did above and, again, your answers don't matter. In fact, they matter even less because you won't be looking at this certificate in a list next to others.

```
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]:CA
State or Province Name (full name) [Some-State]:Nova Scotia
Locality Name (eg, city) []:Truro
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Delicious Brains Inc
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:Mergebot
Email Address []:noreply@mergebot.com

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

Next we'll create the certificate using our CSR, the CA private key, the CA certificate, and a config file, but first we need to create that config file.

The config file is needed to define the Subject Alternative Name (SAN) extension which is defined in this section (i.e. extension) of the certificate:



The configuration file (dev.deliciousbrains.com.ext) contained the following:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = dev.deliciousbrains.com
```

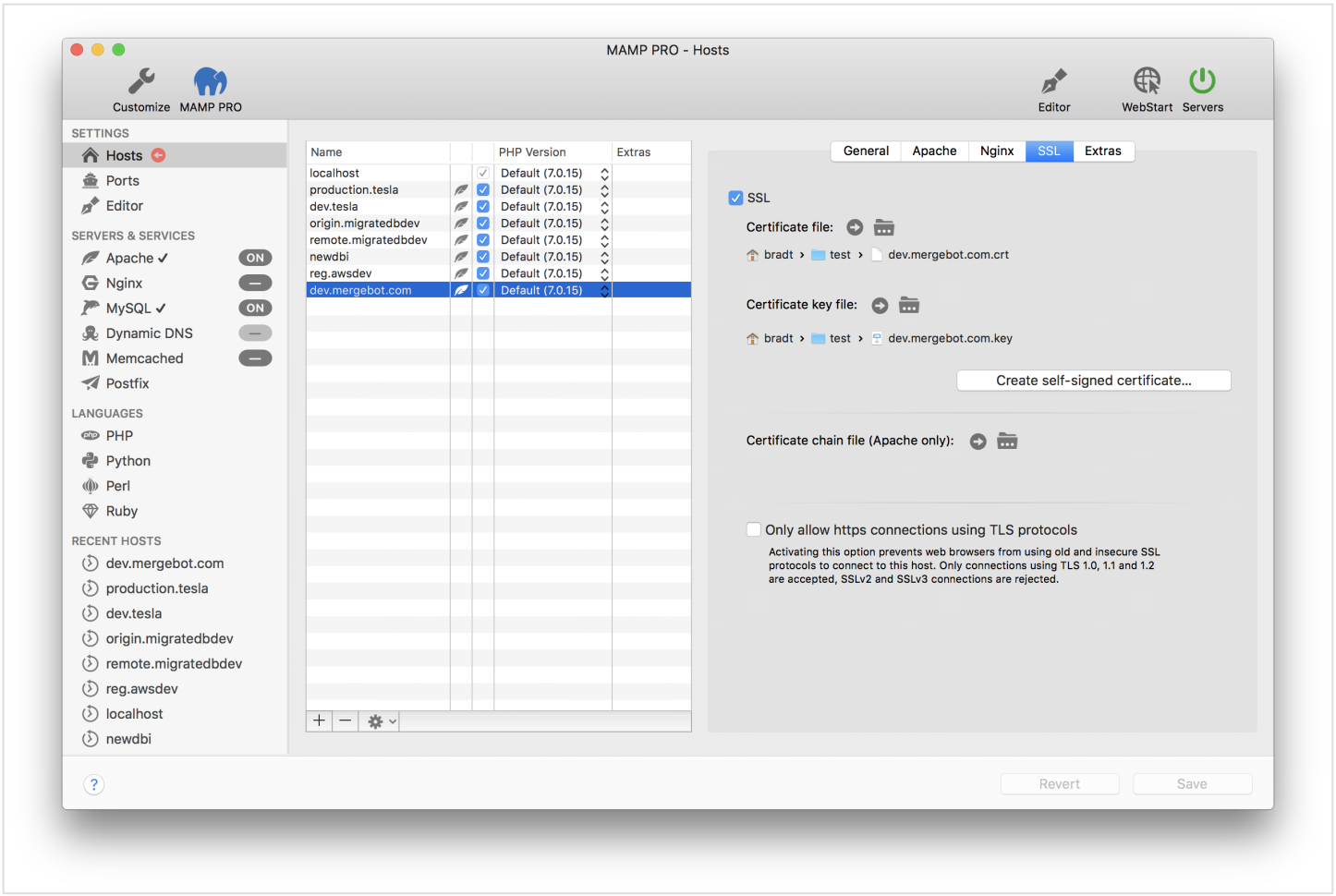
We'll be running the `openssl x509` command because from what I understand, the x509 command is needed to do the signing with the root certificate and private key. I found this example config file on Stack Overflow and it seems to work.

Now we run the command to create the certificate:

```
openssl x509 -req -in dev.deliciousbrains.com.csr -CA myCA.pem -CAkey myCA.key -CAcreateserial \
-out dev.deliciousbrains.com.crt -days 825 -sha256 -extfile dev.deliciousbrains.com.ext
```

I now have three files: dev.deliciousbrains.com.key (the private key), dev.deliciousbrains.com.csr (the certificate signing request), and dev.deliciousbrains.com.crt (the signed certificate).

I can now configure my web server with the private key and the certificate. If you're running a Linux server, you can use the instructions in our Install WordPress on Ubuntu 20.04 series If you're using MAMP, you can select the certificate and key files using the UI:



Unfortunately MAMP (tested with version 5.7) doesn't create SSL certs with a CA, so you'll have to use the manual method for now.

For any other dev sites, we can just repeat this last part of creating a certificate, we don't have to create a new CA for each site.

Shell Script

To make things even speedier, here's a handy shell script you can modify for your own purposes:

```
#!/bin/sh

if [ "$#" -ne 1 ]
then
  echo "Usage: Must supply a domain"
  exit 1
fi

DOMAIN=$1

cd ~/certs

openssl genrsa -out $DOMAIN.key 2048
openssl req -new -key $DOMAIN.key -out $DOMAIN.csr

cat > $DOMAIN.ext << EOF
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = $DOMAIN
EOF

openssl x509 -req -in $DOMAIN.csr -CA ../myCA.pem -CAkey ../myCA.key -CAcreateserial \
-out $DOMAIN.crt -days 825 -sha256 -extfile $DOMAIN.ext
```

Conclusion

So there you have it, how to become your own local certificate authority to sign your local SSL certificates and use HTTPS on your local sites. Hopefully this will eliminate the dreaded ‘Your connection is not private’ message for you in Chrome.

Have you tried setting up a CA of your own? Do you work locally with HTTPS? Let me know in the comments below.