Blake Cash
Gabriel Fukumoto
5/10/2019
CS 445 Project - Self-Signed Certificate

Introduction

The goal of this project is to create a custom certificate using openssl, and then have our browser falsely identify the certificate as a legitimate certificate when visiting a website that the certificate is tied to. First, we will go over the exact steps surrounding how to do that using the Firefox browser (though you could do it using any browser) with a certificate bound to a localhost (though you could do this using any host.). After that, in the discussion section, the team will discuss the implications of something like this, and how exactly individuals could use things like this in order to trick people into believing a website is safe.

Steps

The first step is to download and install OpenSSL, which was the primary tool used for this project. A link to OpenSSL's website can be found here:

https://www.openssl.org/

After downloading the openssl.tar.gz, running the tests, and installing, the first order of business is to create an RSA key that we will use in future steps. An RSA key is a tool used as a key in order to create our certificate, which is the next step.

```
(base) blakecash@ubuntu:~$ openssl genrsa -aes256 -out ca.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.................................+++++
..+++++
e is 65537 (0x010001)
Enter pass phrase for ca.key:
Verifying - Enter pass phrase for ca.key:
```

This generates the file 'rootCA.key' found in the github in the 'certfolder' directory.

From there, the next step is to use the key that was just generated, in this case ca.key, in order to create an SSL certificate using the following command:

```
(base) blakecash@ubuntu:~$ openssl req -x509 -new -nodes -key rootCA.key -sha256
 -days 30 -out rootCA.pem
```

This generates the file rootCA.pem found in the github in the ‘certfolder’ directory.

When generating the key, the user will be prompted (not pictured) for an array of fields, like the organization, location, and division of whoever is signing the certificate. For the sake of this project, we signed our certificate legitimately, with our own name, Reno as our location and UNR as our organization, but if we were actually trying to deceive people with this certificate, we would instead fill these spaces with fraudulent information, such as listing our location as California and our organization as Google, or something to that effect.

Since the key we just generated is a .pem file, and the Mozilla browser we’ll be using from this point forward likes .crt files more than .pem files, we simply use openssl to convert our .pem to a .crt using the command:

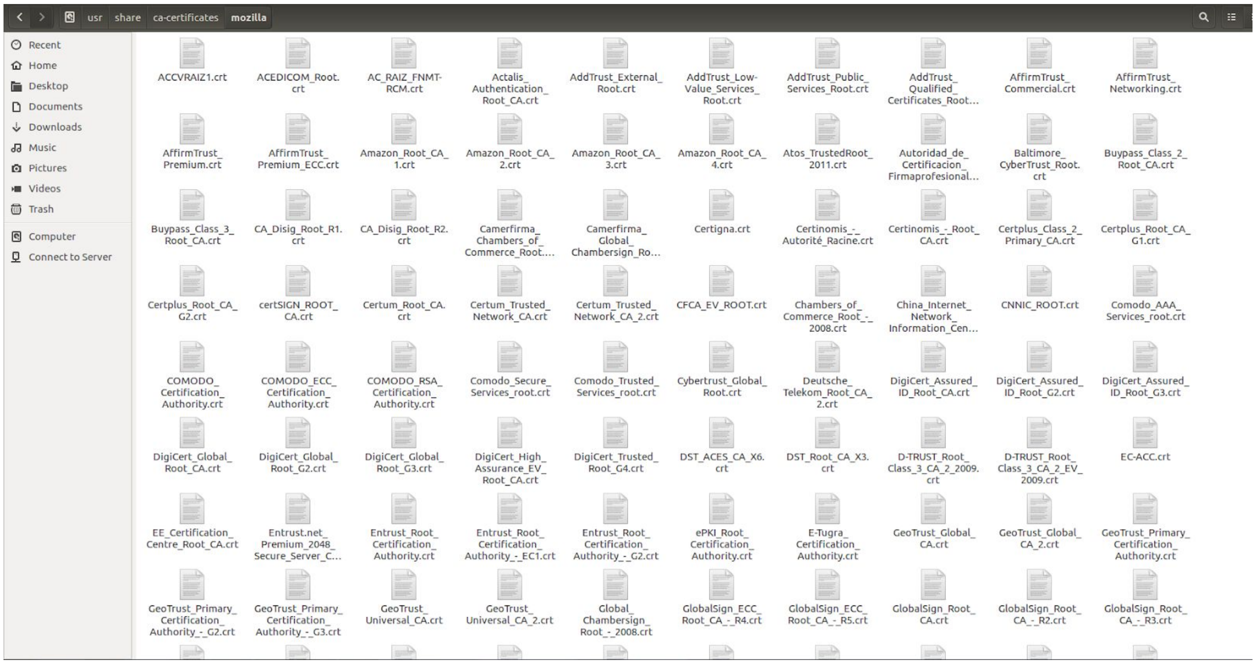
```
openssl x509 -outform der -in your-cert.pem -out your-cert.crt
```

This generates the file rootCA.key found in the github in the ‘certfolder’ directory.

The next thing we need to do is ensure that our browser of choice, in this case Firefox, is going to accept the certificate we just created as something it considers safe. That way, when we use that .crt to create certificates specific to our needs, when Firefox sees and interprets our certificate, it will consider it safe rather than suspicious. Firefox keeps a list of .crts that it trusts in the following directory (for linux):

/usr/lib/mozilla/certificates

The directory looks like this, usually, as you can tell it's full of trusted certificates:



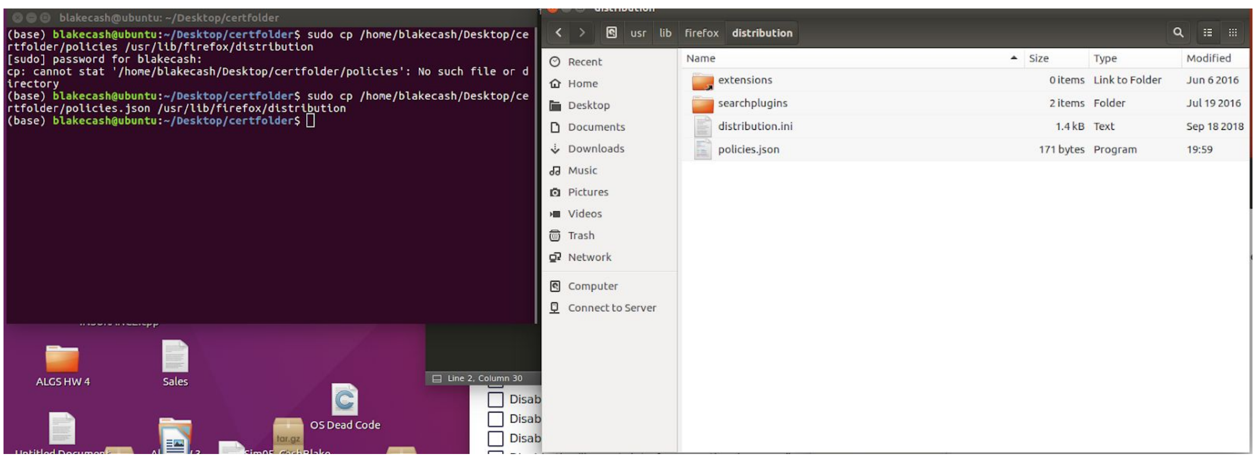
So our next step is clear, we simply need to get our .crt file into that list of .crt files. However, that's a little easier said than done. Mozilla, understandably, doesn't like people tinkering with what certificates are trusted normally, but there's a way to change that.

Within Firefox's 'distribution' directory, a user can create a file called 'policies.json', with a list of modifications they would like to make to the way that Firefox runs. Two of these modifications, luckily, are the ability to accept a custom certificate and the ability to specify what certificates are accepted. So, we create our own policies.json file and put it into our Firefox distribution library to activate these settings. The file looks like this:

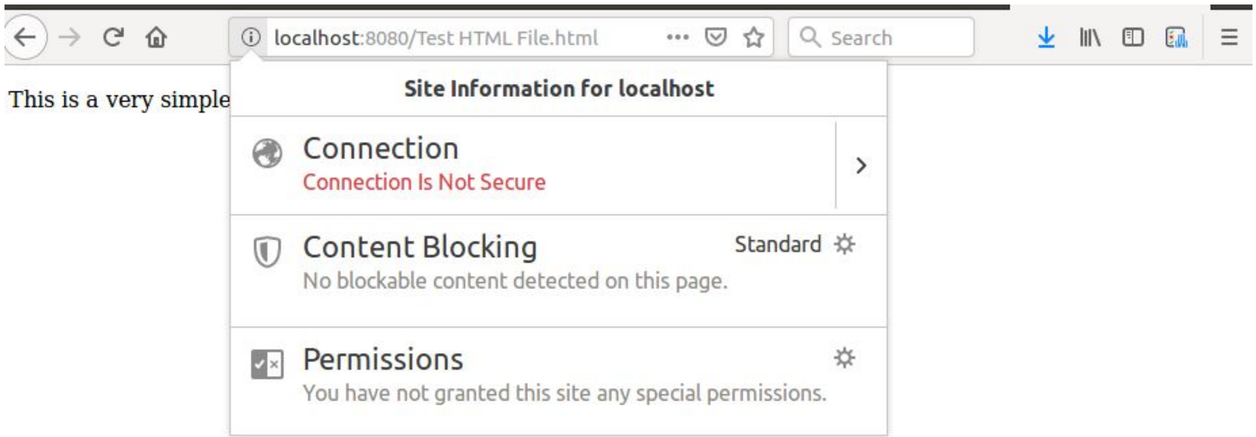
```json
{
  "policies": {
    "Certificates": {
      "ImportEnterpriseRoots": true,
      "Install": [
        "/home/blakecash/Desktop/certfolder/rootCA.crt"
      ]
    }
  }
}
```

This file is policies.json in the github in the ‘certfolder’ directory.

And the command to insert the .json into the correct directory is here:



So now, we have a rootCA.crt that Firefox falsely believes is a trustworthy.crt file. From here, what we need to do is use that in order to get firefox to falsely believe a website is secure. In order to do that, we first need a website, though. Originally, the plan was for the team to secure a free webhost to demonstrate this, but we later found out a localhost works just as well for our purposes, so we’ll be doing all of the following on a localhost. We’ll be doing that using node.js in order to modify our localhost as if it were a real web host, but that comes later. As you can see below, Firefox does **not** trust localhost by default, and upon expanding can’t even find a certificate for it at all.



From here, we need to generate two more files in order to accomplish the end result we want. The first thing we need to do is use the root certificate in order to issue a certificate that is for our localhost. The root certificate, as of now, isn't tied to our localhost, and won't work if we just plug it in.

The first step in doing that is generating a server key (server.key) for localhost with a lot of very specific configurations, as seen below.

```
writing new private key to 'server.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]:US
State or Province Name (full name) [Some-State]:NV
Locality Name (eg, city) []:Reno
Organization Name (eg, company) [Internet Widgits Pty Ltd]:University of Nevada,
 Reno
Organizational Unit Name (eg, section) []:CS 445
Common Name (e.g. server FQDN or YOUR name) []:localhost:8080
Email Address []:blakecash@nevada.unr.edu

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

This generates the file server.key found in the github, as well as the certificate request server.csr, both in the 'certfolder' directory.

As you can see, just like we did earlier, we input a bunch of specific information for our key. Again, if we were attempting to run some sort of scam, this information could easily be falsified. It's worth noting that even though we're using localhost(port 8080 specifically), if we wanted to do this same operation for an actual web server, the only thing we'd need to do differently so far is changing the 'common name' field to whatever our host name was. The key comes out looking like this:

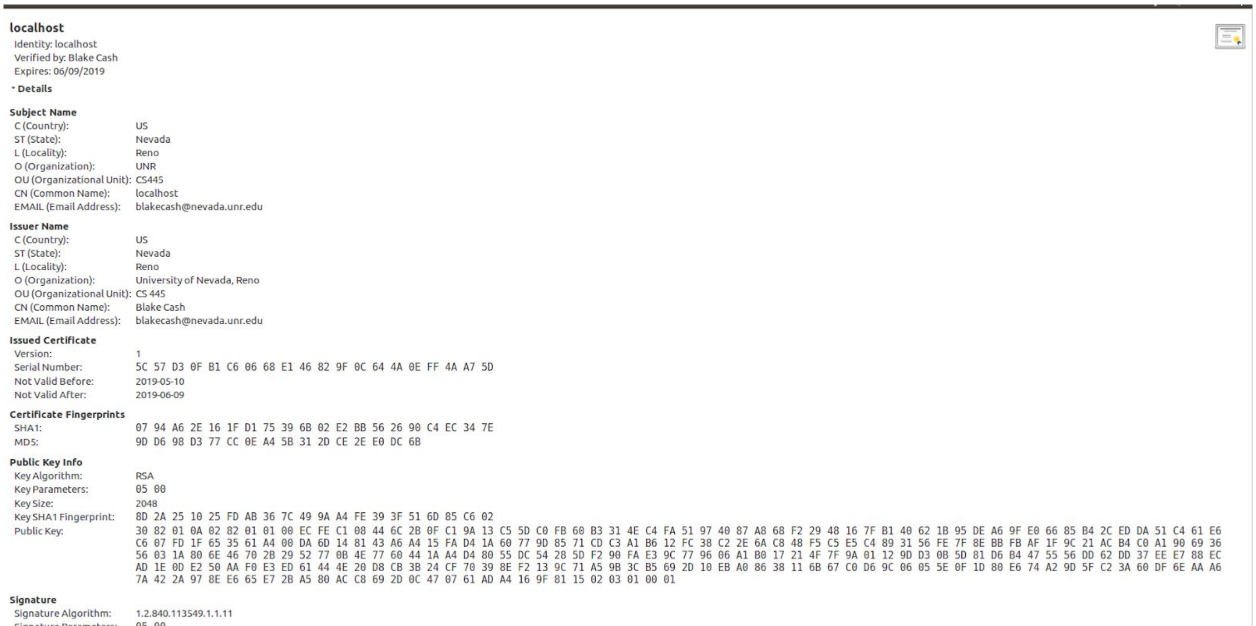

```
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDs/sEIRGwrD8Ga
E8VdwPtgszFOxPpRl0CHqGjyKUgWf7FAYhuV3qaf4GaFtCzt2lHEYebGB/0fZTVh
pADabRSBQ6akFfrUGmB3nYVxzcOhthL8OMIuashI9cXlxIkxVv5/jrv7rx+cIay0
wKGQaTZWAxqAbkZwKylSdwtOd2BEGqTUgFXcVChd8pD645x3lgahsBchT3+aARKd
0wtdgda0R1VW3WLdN+7niOytHg3iUKrw4+1hRE4g2Ms7JM9wOY7yE5xxpZs8tWkt
EOughjgRa2fA1pwGBV4PHYDmdKKdX8I6YN9uqqZ6QiqXjuZl5yulgKzIaS0MRwdh
raQWn4EVAgMBAAECggEBAJd6qaUAHudTNdqmonMvYz1Gq9B+JMU72PockZ+e9T20
NnZBfwJHAteTldQF+uW4sqTEMr4G4ypLBVi4e/cg24dX105v4hfqGBi8bUv1SgK1
nuLp4GvMwuGnfetDuLD5useLUuom4BxqhboumdX0+c72Qt0uHwWZANt9zZNEyBoe
dqoX0kHdc9yBJMOkyO8r5RsAEYAL64PcPUC5DdqNEWKi5PIw6lDbwLmPZevuqBzv
MVXr2NpD/VgwAwMuvz3NXg3FMN9y3/zoZRyCEyDZnwBbBRb3JwDgpWpC4OCFjmtq
Py6OgjPIE9C85WF0fD8wDFOYHcnNA8sPEp76Oy8Bg4ECgYEA+p9ZzSsruIIrqHLw
FBAtY12ktNIxTfkRw3ekeFh5pIbFkeqbWzFnBvJ2PzOl/TkOiTXMspRKBij6q24w
Y2YdNBv0w3z7RvgRNTKSlLuwkdMhxnZrni7b0A/uKrTis/32e51ee8+lZzPsvn4c
ZkBvhMuK77UkARsNW4UBHmEK1TUCgYEA8hSMpaMx+O5eKE+SBBYligOhHbnxRY8q
FOuJAEHN3Pl9u37yiTjGWrDNR6T9pDYAc3Uucr/xandIJ9X9JxGw92Qa2+Kh1nj9
PvFi63sGBA2kDStF1zKuuebPlrWJaDcPsQH4rGsT/YRu6xLXd/o4rUbTVEDTYUtV
GejUk0sg2GECgYBaQOKFCUPwaSgxvFqKzxyZSRLUb1GEI+rqun5HPI1p24Lwvkz+
NTdGADDnJ8FiBJcggHhb4x0ZRM6ox8CzOwXPSzYE5FBVSWZhAvpYh5LZoO/r2Z99
0qAkOGhrhsKRUSbfc1egPLHzDXb3TtEJXbELYIWDN3dk0oon6Cjz+Lx+9QKBgDO4
dDxHW7vlobRXG8lHugl8sQWa3pOP/NuvXvvxEzytCVHv033B5Y8myxUNiSt2Zi5E
0QGvRLMMfRwVuhqIyxhwCNUF3LHn86NpC0toY2amS3CM2EUcDPym9Z8rdgoQCYg1
9Z1Q21qE2vXadrKpgUZ0JV6Q4xiccbgxaIl3ubLBAoGBANPN7jpeb3IxRSTlm/2I
4h2MW97ikSM9USbk5qUzplNBQk3rpw9JoUn2FRD8krFagBw1ApOqTrkXw24busqQ
1E9f6hv8Lb4TDXU4ChypLfSZWBR/bQgrh52GS6iFxObacHD9Woz8CafVY19eGOoN
IfStHgffJuc0K/sI1HoxyXZh
-----END PRIVATE KEY-----
```


The final file we need to generate is our certificate specific for the localhost, using our rootCA key that we already generated much earlier, this also satisfies the certificate request that was also generated by us creating the .key file.

```
(base) blakecash@ubuntu:~/Desktop/certfolder$ openssl x509 -req -in server.csr -
CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out server.crt -days 30 -sha256
Signature ok
subject=C = US, ST = NV, L = Reno, O = "University of Nevada, Reno", OU = CS 445
, CN = localhost:8080, emailAddress = blakecash@nevada.unr.edu
Getting CA Private Key
Enter pass phrase for rootCA.key:
```

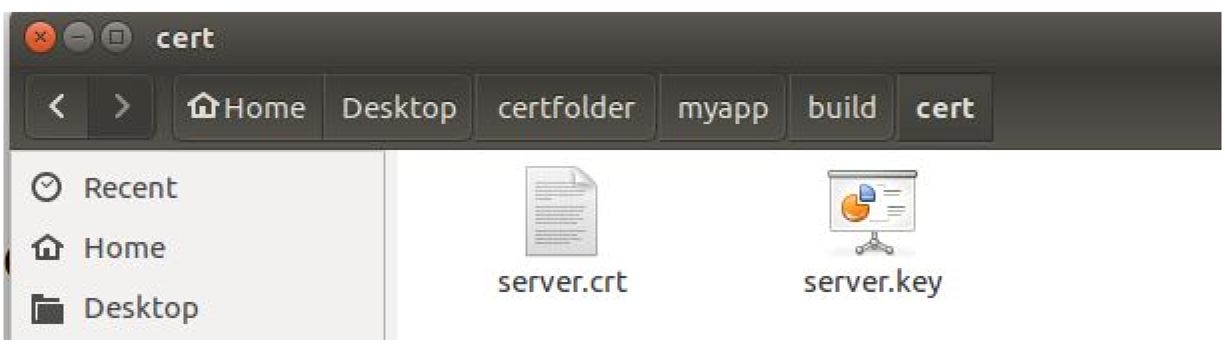
This generates the file server.crt found in the github.

The .crt lookslike this:



So now we have everything we need in order to attach our certificate to the page we want to attach it to. The following steps are specific to doing so using node.js, so if we wanted to do this in a real situation these steps would be replaced with whatever steps are necessary for binding a .crt that the specific host uses.

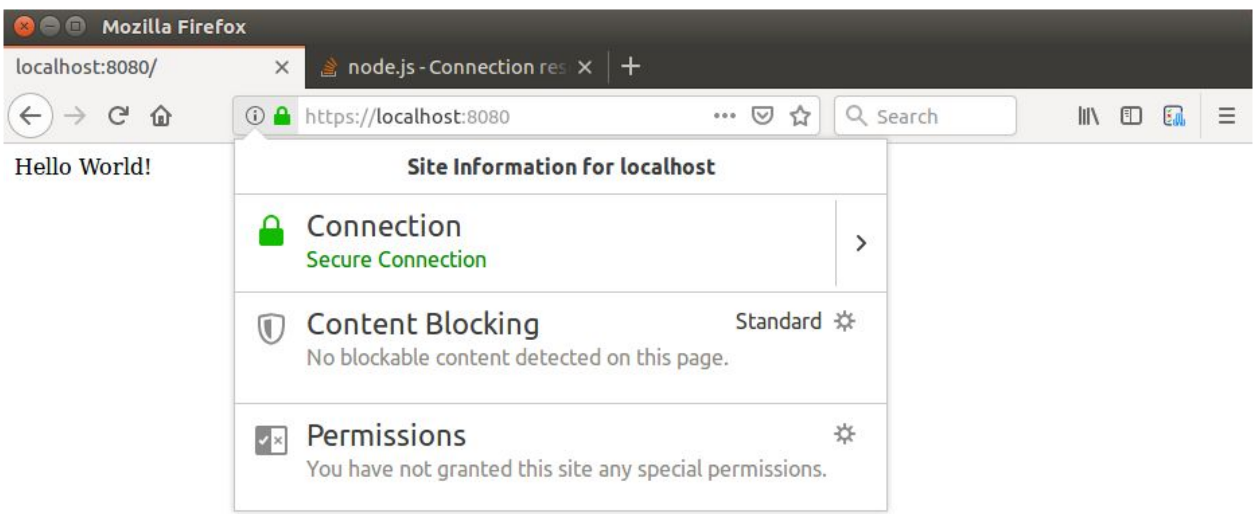
We place our .key and .crt into specific locations that our node.js can access.



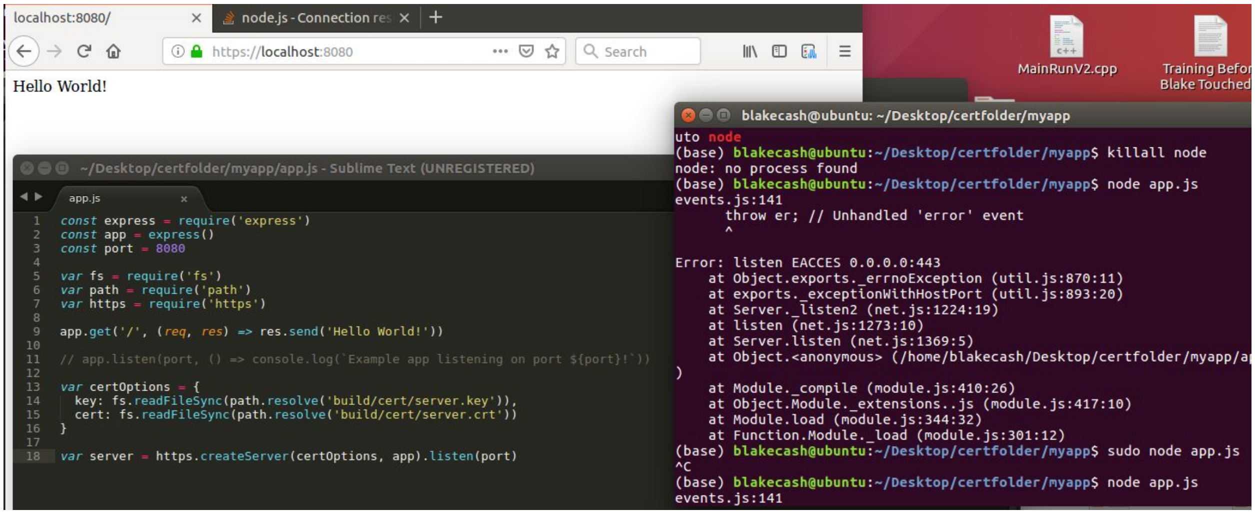
And our node.js code is below as follows. Again, this is specific for implementing with node.js and we would plug these two files in another way for another method.

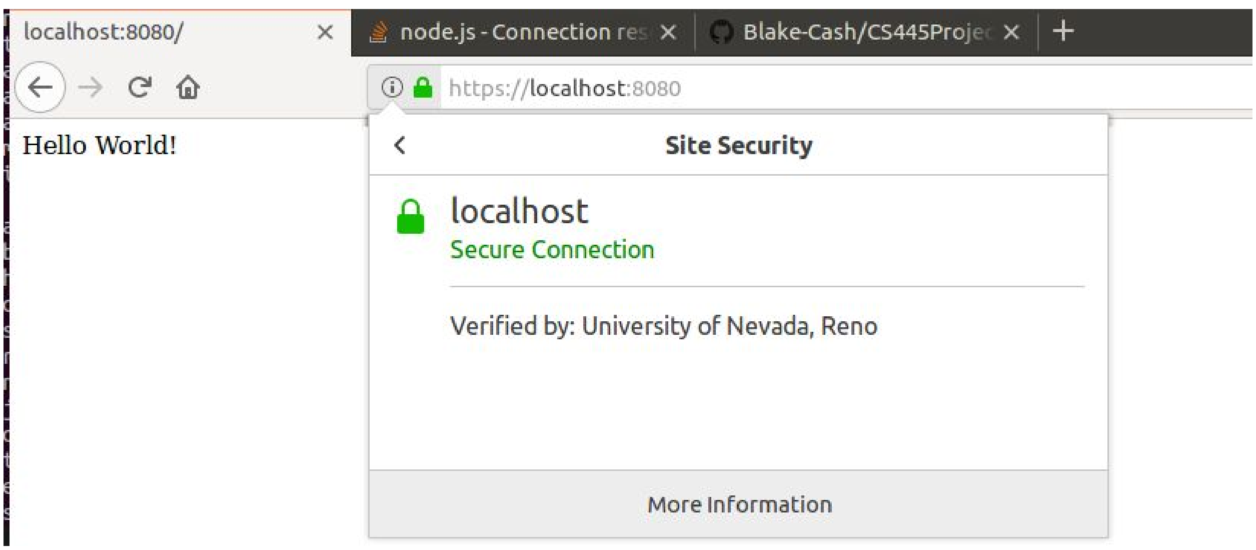
This file is app.js in the github in the 'certfolder' directory.

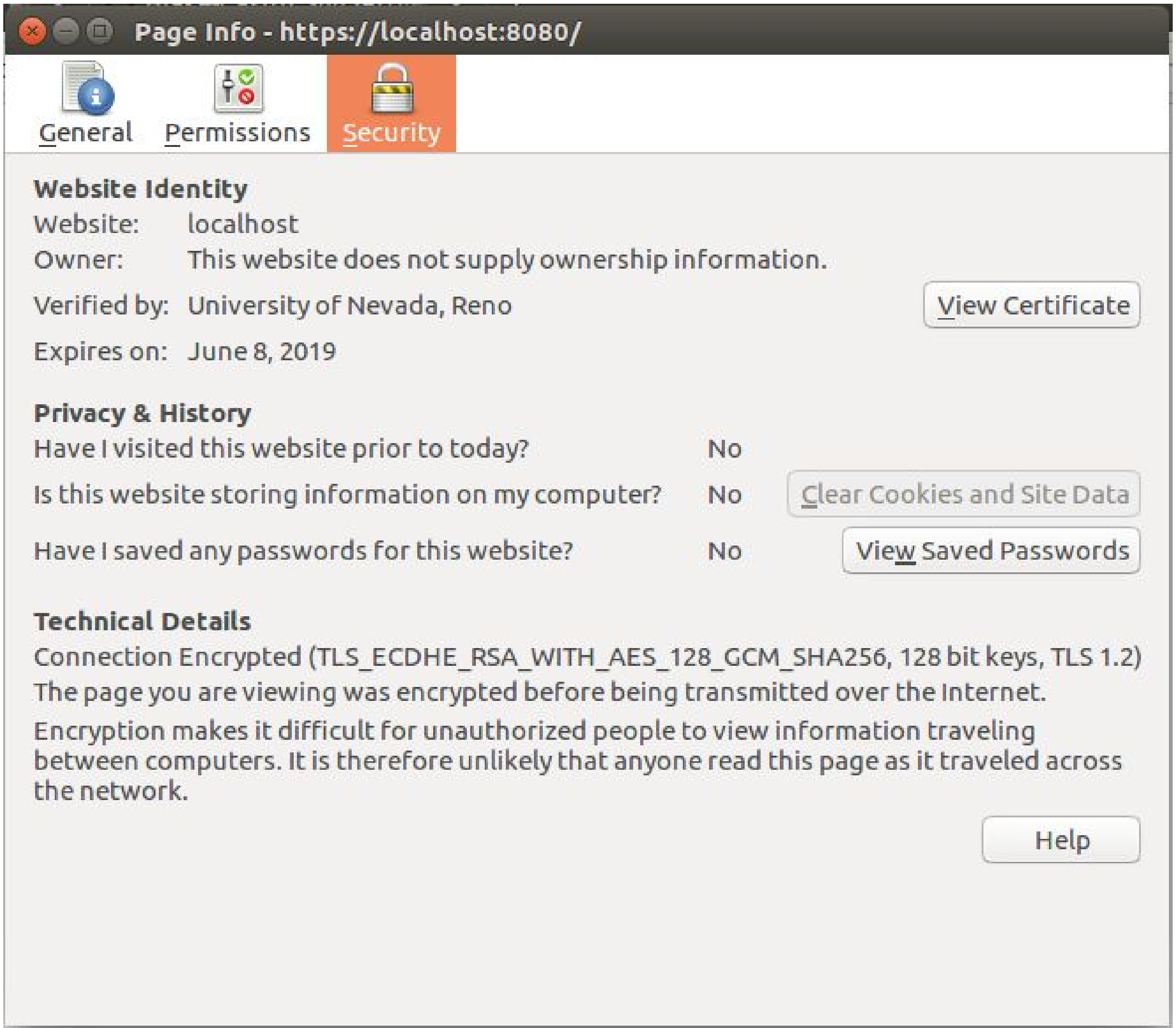
And now, all that's left to do is run the node.js and see if it works:

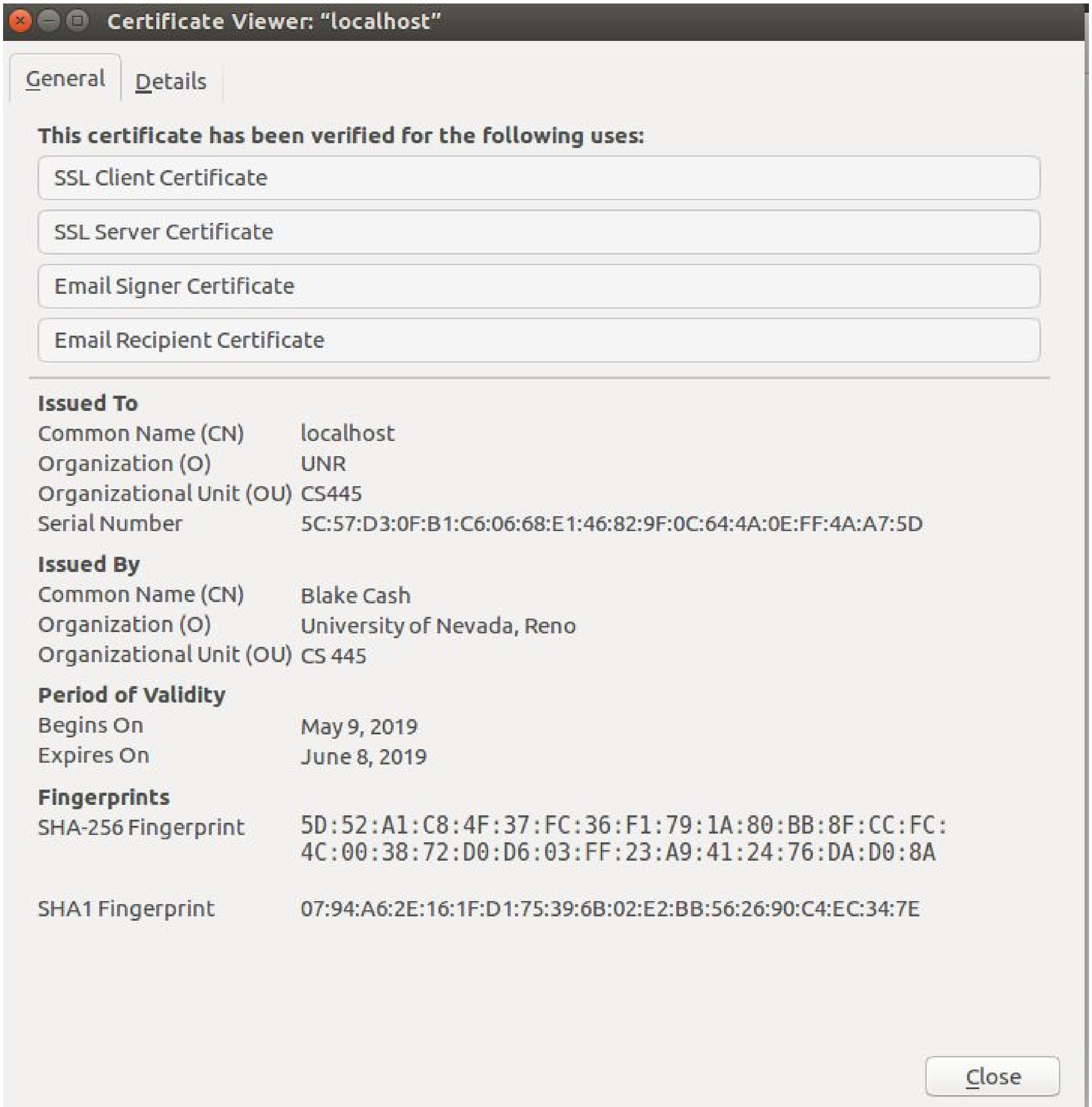


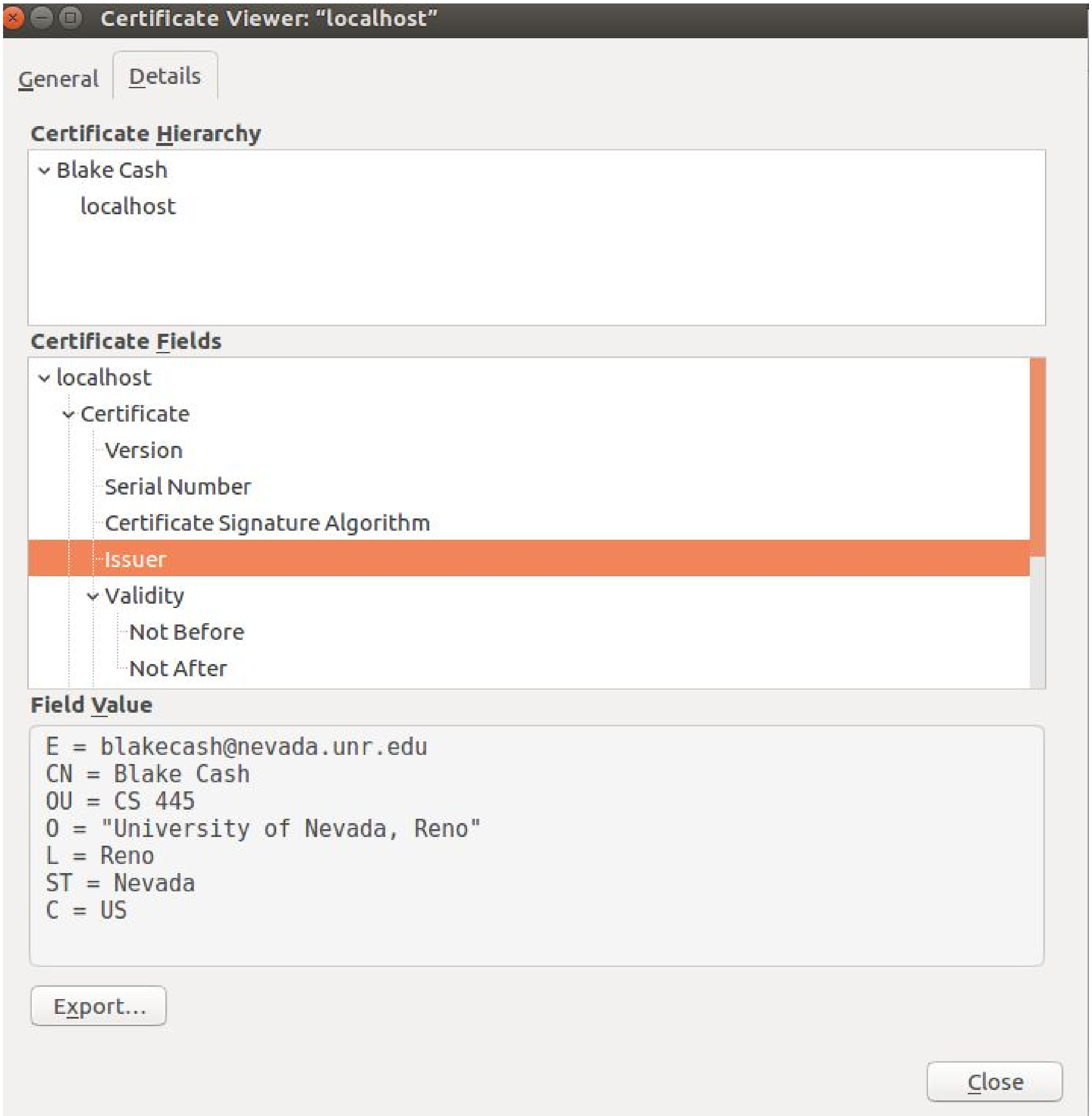
Ta-da! Now the localhost is considered safe by our browser. We did this by attaching these two files, the .crt and .key to this host, and since Firefox has been told that the certificate is safe (by modifying the policies.json), Firefox in turn tells us that the site is secure. It's worth noting that on a different web browser, the process for embedding the certificate would be the same, but the process for telling the browser that certificate is safe might be different. Here's some more proof that the certificate we created is the one it's reading, and the one it thinks is safe.






Page Info - https://localhost:8080/
General
Permissions
Security
Website Identity
Website: localhost
Owner: This website does not supply ownership information.
Verified by: University of Nevada, Reno
View Certificate
Expires on: June 8, 2019
Privacy & History
Have I visited this website prior to today? No
Is this website storing information on my computer? No
Clear Cookies and Site Data
Have I saved any passwords for this website? No
View Saved Passwords
Technical Details
Connection Encrypted (TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, 128 bit keys, TLS 1.2)
The page you are viewing was encrypted before being transmitted over the Internet.
Encryption makes it difficult for unauthorized people to view information traveling between computers. It is therefore unlikely that anyone read this page as it traveled across the network.
Help


Certificate Viewer: “localhost”
General
Details
This certificate has been verified for the following uses:
SSL Client Certificate
SSL Server Certificate
Email Signer Certificate
Email Recipient Certificate
Issued To
Common Name (CN) localhost
Organization (O) UNR
Organizational Unit (OU) CS445
Serial Number 5C:57:D3:0F:B1:C6:06:68:E1:46:82:9F:0C:64:4A:0E:FF:4A:A7:5D
Issued By
Common Name (CN) Blake Cash
Organization (O) University of Nevada, Reno
Organizational Unit (OU) CS 445
Period of Validity
Begins On May 9, 2019
Expires On June 8, 2019
Fingerprints
SHA-256 Fingerprint 5D:52:A1:C8:4F:37:FC:36:F1:79:1A:80:BB:8F:CC:FC:
4C:00:38:72:D0:D6:03:FF:23:A9:41:24:76:DA:D0:8A
SHA1 Fingerprint 07:94:A6:2E:16:1F:D1:75:39:6B:02:E2:BB:56:26:90:C4:EC:34:7E
Close



And that's it. As discussed already, these steps could be replicated on any web host, not just the localhost, as long as all fields (like common name) that included localhost were replaced with the name of the localhost. The hard part, of course, would be getting another user to modify their policies.json (or equivalent action for another browser) in order for them to see the certificate as trustworthy, but we'll discuss the logistics of that in the next section. The above is all necessary steps in order to create a fraudulently trusted certificate.

Discussion

In conclusion, it’s easy to see how these tools could be used for bad. It’s extremely easy to create your own SSL certificate. To get this working on another machine, you would obviously not use the localhost, and in addition you would simply modify any part of the code that asks for credentials in order to impersonate another more trustworthy source in order to add legitimacy to your certificate. The good news is that even if this wasn’t done on localhost, our certificate would appear whenever someone visited our website if we attach the certificate correctly depending on our webhost. The bad news is it wouldn't be considered valid by any browser. As we saw before, browsers keep a comprehensive list of all the .crt's they think are safe, and only give the fancy green 'safe' lock to websites on that list. That means, in order to be seen as a valid certificate even after we embed it like this, we have to find a way to manipulate the preferences .json of a target user in the same way we update the preferences.json of our own firefox browser (on different browsers/operating systems this would work differently.)

This could, theoretically, be done by a simple scam that asks user to or tricks users into running a script that would modify their policies.json (or the equivalent for another browser), or simply injecting code that runs the script yourself. Then, when the user visits a website of the attacker’s designation, the user would see the approved certificate and falsely believe that the website is safe. Less technically-savvy users could be very easily convinced that the ‘secure’ connection that is represented by the lock at the top of the screen means sensitive information like their credit card information can be safely inputted into a website

In this way that attacker, after having tricked the user into believing their website is secure using the certificate falsely approved by some sort of script, would be able to trick the user into trusting the website an possibly entering valuable information that the attacker would then have access to.

In conclusion, creation of your own self-signed certificate is a powerful and easy to do tool that, in addition to other malicious strategies, would be an excellent way of deceiving users into trusting a website.