

Name: Nicholas Mason
Course: CS445 – Internet Security
Date: May 2nd, 2018

CS 445 – Internet Security Technical Report

Introduction

Certificate identification and Certificate Authorities are crucial elements in Internet security. Without these key contributions to security, personal and confidential information would be widely available to anyone capable of performing basic computer functions. In this project, I will attempt to create an SSL self-signed certificate using openssl, encrypt the data stored on an apache webserver, and submit my self-signed certificate to my local machine to verify the authenticity of my certificate by a web browser. This technical report will explain in detail the process of creating the self-signed certificate and adding it to a web browser and my local machine, in addition to some figures to show my success. My GitHub will show the code segments that I use to accomplish these goals.

SSL Certificates

SSL certificates were originally created in order to provide an additional layer of security when a private party is attempting to communicate on the Internet, specifically connecting to websites where confidential information may be passed between each party. A website where confidential information is needed to complete a transaction or process, involving such information as a user's name, password, banking information, or social security number, must now have a SSL certificate. Having such a certificate allows the website to obtain a distinct public and private encryption key, further providing a higher level of security online.

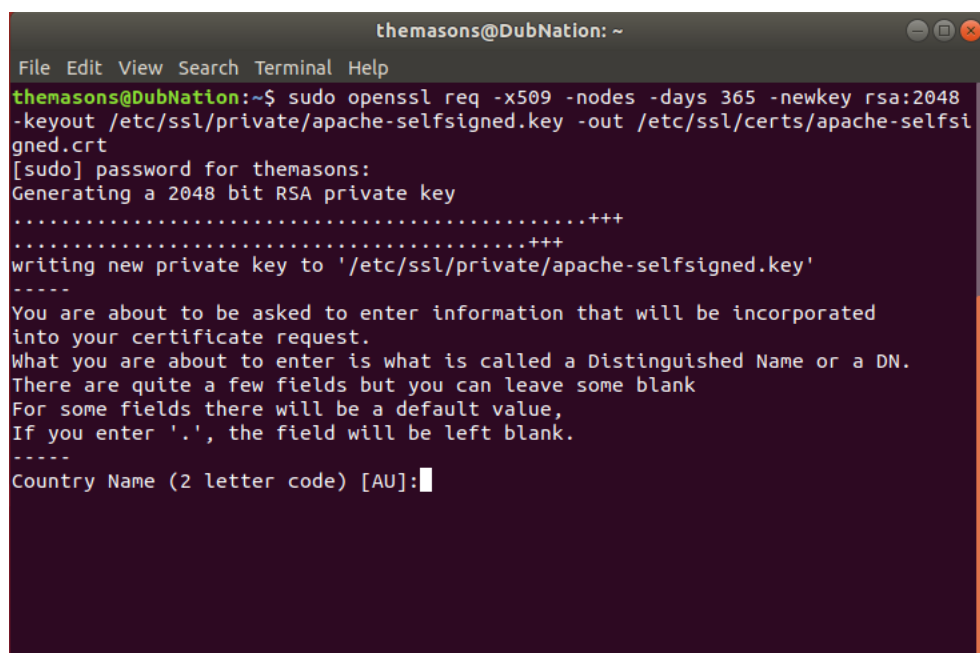
A SSL certificate can be obtained by a given website by going through the process of finding a third party Certificate Authority, also called a CA. The CA will possess the ability to authenticate and legitimize the intent of the website. If the website passes the qualifications of the CA, the website will gain access to their own unique public and private encryption keys for their specific IP address or domain name of their website. To a user on the Internet, the lock icon next to the URL and the URL starting with the symbols HTTPS ensures the user that the website has obtained some sort of SSL certificate from a Certificate Authority, resulting in their personal information being encrypted.

Creating a Self-Signed SSL certificate

Creating an SSL certificate creates many benefits for the person creating the SSL certificate, or for the website or server in which the certificate is made for. This process can help a website or server in fighting against various cyber-attacks. One of the most recognized attacks that can be avoided by creating a self-signed SSL certificate is the "man in the middle" attack. A man in the middle attack can be depicted as a user sitting between two party, listening to the content or traffic between each party. While listening to the traffic between each party, the malicious user can potentially relay or alter the communication between the two parties, resulting in the

relaying of false information. Through the process of creating a self-signed SSL certificate, a server or website of any kind is encrypted, hindering the malicious users ability to listen to the traffic between various groups or parties.

In order to successfully create an SSL certificate on a Linux machine, which is convenient for this project, various steps must be completed in order to ensure that the encryption will be executed correctly. To start, creating an RSA keypair is required. Using openssl in the Linux environment will allow for built in functionalities to assist in this process. An RSA key pair consists of two types of keys, private key and a public key. The RSA can be defined as “private key based on RSA algorithm. Private Key is used for authentication and a symmetric key exchange during establishment of an SSL/TLS session” (Namecheap, Inc.). This is one of the parameters needed when creating a SSL certificate. The public key will look that there exists asymmetric encryption, meaning that there is a private key and a public key included in the SSL certificate. The private key will not be present to any person in this process because it is unique to the server or website that the certificate is being made for. In the case of a self-signed certificate, the private key can be extracted to a folder because the user creating the SSL certificate is also the user who owns the domain or server that is being encrypted. I will provide figures throughout this technical report that I have indeed done this process.



```
themasons@DubNation: ~  
File Edit View Search Terminal Help  
themasons@DubNation:~$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048  
-keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-selfsi  
gned.crt  
[sudo] password for themasons:  
Generating a 2048 bit RSA private key  
.....+++  
.....+++  
writing new private key to '/etc/ssl/private/apache-selfsigned.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]:
```

Fig. 1: Figure 1 shows the process of typing the sudo openssl req command, creating the -x509 and the RSA private key for this certificate.

When creating the self-signed certificate, the openssl functionality requires the user to enter some information to display when others can see the certificate. You will need to input, the Country Name (2 letter code), the State or Province Name (full name), the Locality Name (e.g. City), the Organization Name, the Organizational Unit Name, the Common Name (e.g. server

FQDN or YOUR name), and an email address. Once you fill out the information requested by openssl, your certificate will be written to the directory in which you indicate in your terminal.

Configure Apache Webserver

Once the SSL self-signed certificate has been made and the individual specifications have been specified by the user, the next step in this project is to configure an apache webserver to test the encryption of our certificate. First, the user must update their system in the command line to account for any new updates. If apache2 is not yet installed on Linux, which in most cases it is, the command, `sudo apt install apache2`, will install apache webserver for future use. By simply typing, `sudo systemctl status apache2`, the user will be able to start their apache webserver. To test the connection, open a web browser, and type `http:// <IP address of server>`. In the case of this project, I have gone and changed the look of my personal apache web server to display By: Nick Mason, with the title of the website being CS445 – Internet Security.

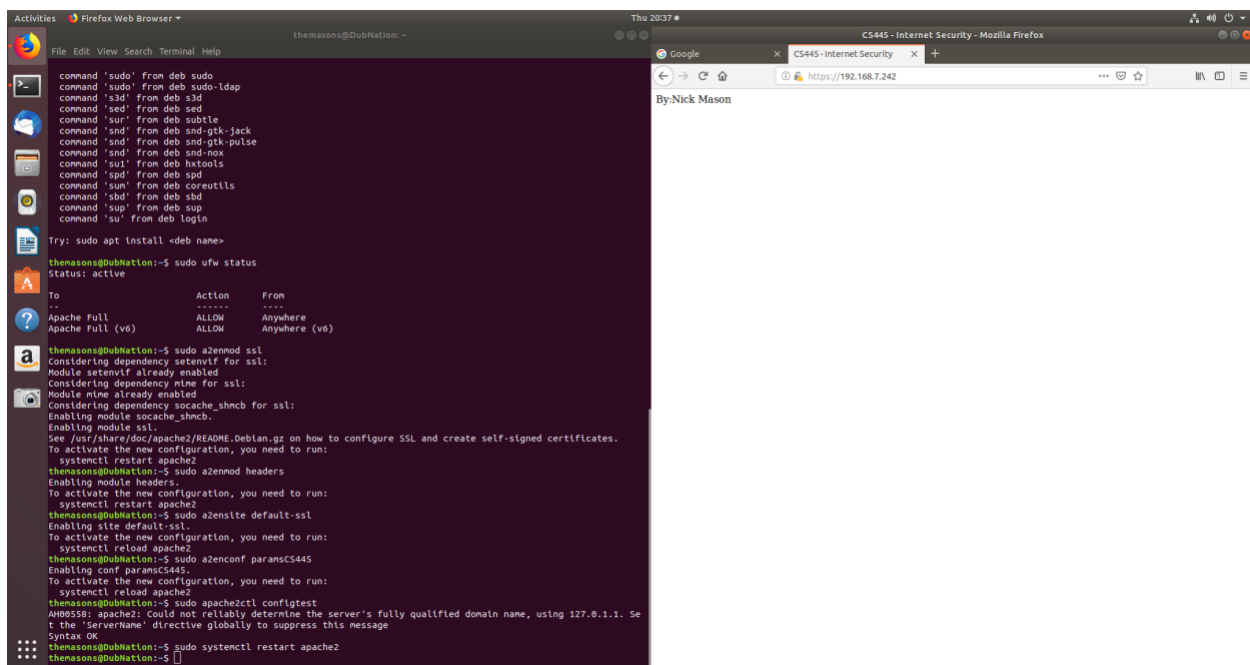


Fig 2: Figure 2 shows the apache web server being successful, resulting in the new appearance that I have encoded in the apache configuration file to show my name and the course in the title.

Configure Apache to Use SSL Certificate

Configuring apache to use the self-signed certificate is one of the most important processes in this project. To start, the user must build a configuration snippet in order to create a secure environment for my apache webserver. In this project, I have named my configuration file `paramsCS445.conf`, which can be located in the directory `/etc/apache2/conf-available/`. An important note when changing files in `/etc/` on your computer is to use the phrase `sudo`, which gives the user root privileges which are needed when changing certain files. Next, in order to get my apache webserver to recognize my certificate, I need to update the `default-ssl.conf`

configuration file to point to my newly created, self-signed certificate. I duplicated the default configuration file so I have a backup of my already existing file. When modifying the virtual host file, the user needs to update the ServerAdmin, ServerName, which should now be set to the IP of the apache webserver, the SSLCertificateFile, and the SSLCertificateKeyFile. The SSL certificate file and key file will be changed to the newly created apache-selfsigned.crt and apache-selfsigned.key. Figure 3 references the changes I have made for this project. Once this has been changed, save the file and continue.

```
<IfModule mod_ssl.c>
<VirtualHost _default_:443>
    ServerAdmin nicholasmason@nevada.unr.edu
    ServerName 192.168.7.242

    DocumentRoot /var/www/html

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf

    # SSL Engine Switch:
    # Enable/Disable SSL for this virtual host.
    SSLEngine on

    # A self-signed (snakeoil) certificate can be created by installing
    # the ssl-cert package. See
    # /usr/share/doc/apache2/README.Debian.gz for more info.
    # If both key and certificate are stored in the same file, only the
    # SSLCertificateFile directive is needed.
    SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
    SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key

    # Server Certificate Chain:
```

Fig. 3: Figure 3 shows the newly changed Virtual Host File.

The last step in this section that needs to be done is to change the unencrypted virtual host file to automatically redirect requests to the encrypted virtual host file. In addition, the user should change the http traffic to https to ensure higher level of security. The user can change the redirect to be permanent if they desire, but it is not required for this process to be successful. In addition, change the ServerAdmin on the 000-default.conf, or the unencrypted virtual host file in order to keep the systems connected and accurate. Figure 4 shows the newly configured 000-default.conf configuration file, where I have changed the traffic to https and permanent.

```
GNU nano 2.9.3 /etc/apache2/sites-available/000-default.conf
VirtualHost *:80>
# The ServerName directive sets the request scheme, hostname and port that
# the server uses to identify itself. This is used when creating
# redirection URLs. In the context of virtual hosts, the ServerName
# specifies what hostname must appear in the request's Host: header to
# match this virtual host. For the default virtual host (this file) this
# value is not decisive as it is used as a last resort host regardless.
# However, you must set it for any further virtual host explicitly.
#ServerName www.example.com

ServerAdmin nicholasmason@nevada.unr.edu
DocumentRoot /var/www/html

Redirect permanent "/" "https://192.168.7.242/"

# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# modules, e.g.
#LogLevel info ssl:warn

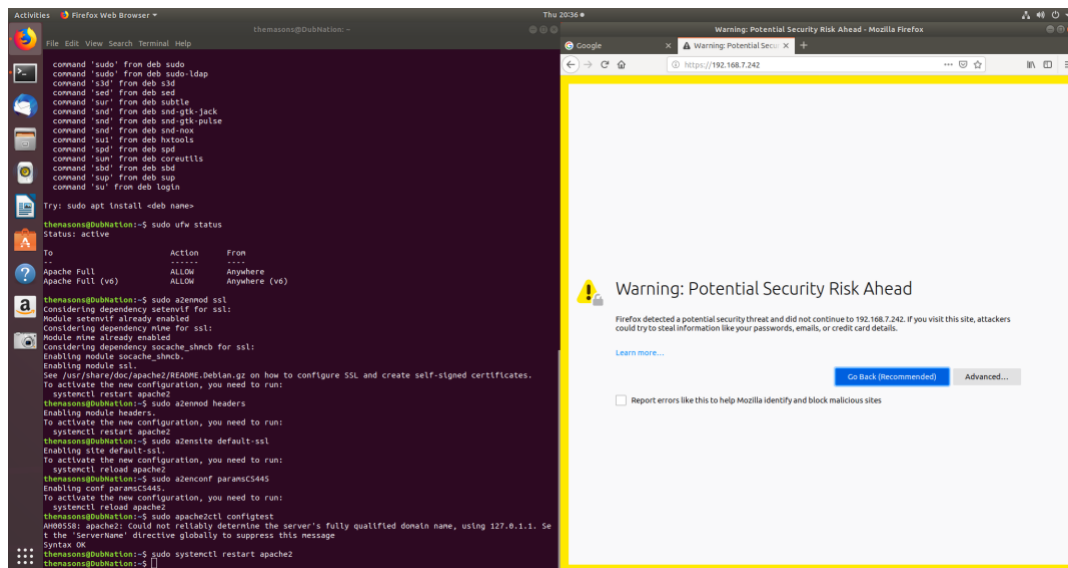
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

# For most configuration files from conf-available/, which are
# enabled or disabled at a global level, it is possible to
# include a line for only one particular virtual host. For example the
# following line enables the CGI configuration for this host only
# after it has been globally disabled with "a2disconf".
#Include conf-available/serve-cgi-bin.conf
VirtualHost>
```

Fig. 4: Figure 4 shows the changes to the 000-default configuration file to use my IP of my apache webserver.

Adjust the Firewall and Changing Features in Apache

In this section, the user will want to make sure that the firewall will allow for the Apache webserver's newly configured features. The ufw command in the terminal will assist in many of these firewall updates. To start, the user should check to see which applications are available to them. The main purpose of this is to allow full access to Apache. In the applications available, both 'Apache' and 'Apache Full' will be available. By typing, `sudo ufw allow 'Apache Full'`, the full apache access is available. The user will need to delete the regular 'Apache' application access to rely solely on the full apache application. This lets in the https traffic to my apache webserver that I am using to test my project. Figure 5 shows the steps just performed, in addition to the changes in apache.



```
command 'sudo' from deb sudo
command 'sudo' from deb sudo-ldap
command 'sbl' from deb sbl
command 'sed' from deb sed
command 'sur' from deb subtle
command 'smd' from deb smd-gtk-jack
command 'smd' from deb smd-gtk-pulse
command 'smd' from deb smd-mex
command 'sui' from deb hxttools
command 'spd' from deb spd
command 'sun' from deb coreutils
command 'sbd' from deb sbd
command 'sup' from deb sup
command 'su' from deb login

Try: sudo apt install <deb name>

nicholasmason@DuhNation:~$ sudo ufw status
Status: active

To Action From
--
Apache Full ALLOW Anywhere (v6)
Apache Full (v6) ALLOW Anywhere (v6)

nicholasmason@DuhNation:~$ sudo a2enmod ssl
Considering dependency setenvif for ssl:
module setenvif already enabled
Considering dependency mime for ssl:
module mime already enabled
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
Enabling module ssl.
See /usr/share/doc/apache2/README Debian.gz on how to configure SSL and create self-signed certificates.
To activate the new configuration, you need to run:
systemctl restart apache2
nicholasmason@DuhNation:~$ sudo a2ensite default-ssl
Enabling site default-ssl.
To activate the new configuration, you need to run:
systemctl reload apache2
nicholasmason@DuhNation:~$ sudo a2enconf paramsC444
Enabling conf paramsC444.
To activate the new configuration, you need to run:
systemctl reload apache2
nicholasmason@DuhNation:~$ sudo systemctl configtest
apache2: Syntax error on line 5 of /etc/apache2/httpd.conf:
Syntax error
nicholasmason@DuhNation:~$ sudo systemctl restart apache2
nicholasmason@DuhNation:~$
```

Fig. 5: Figure 5 shows the commands necessary to get the apache webserver secure. Modification are needed in the firewall and apache itself.

In apache, the user needs to change some features in order to use the newly created and changed configuration files. The `a2enmod` command will modify the apache module and the headers of the configuration files. The `a2ensite` command will enable the SSL-virtual host, which was modified earlier in this project. The file was the default-ssl.conf file. Finally, the `a2enconf` command will enable apache to use the `paramsCS445.conf` configuration file that was created in the beginning of the project for additional security. This will finish the encryption of our apache webserver, however, it will not be recognized as a secure site by the browser. Figure 6 shows that our webserver has been tied to the certificate created in this project. The distinctive qualities of my self-signed certificate can be seen, including my name and class.

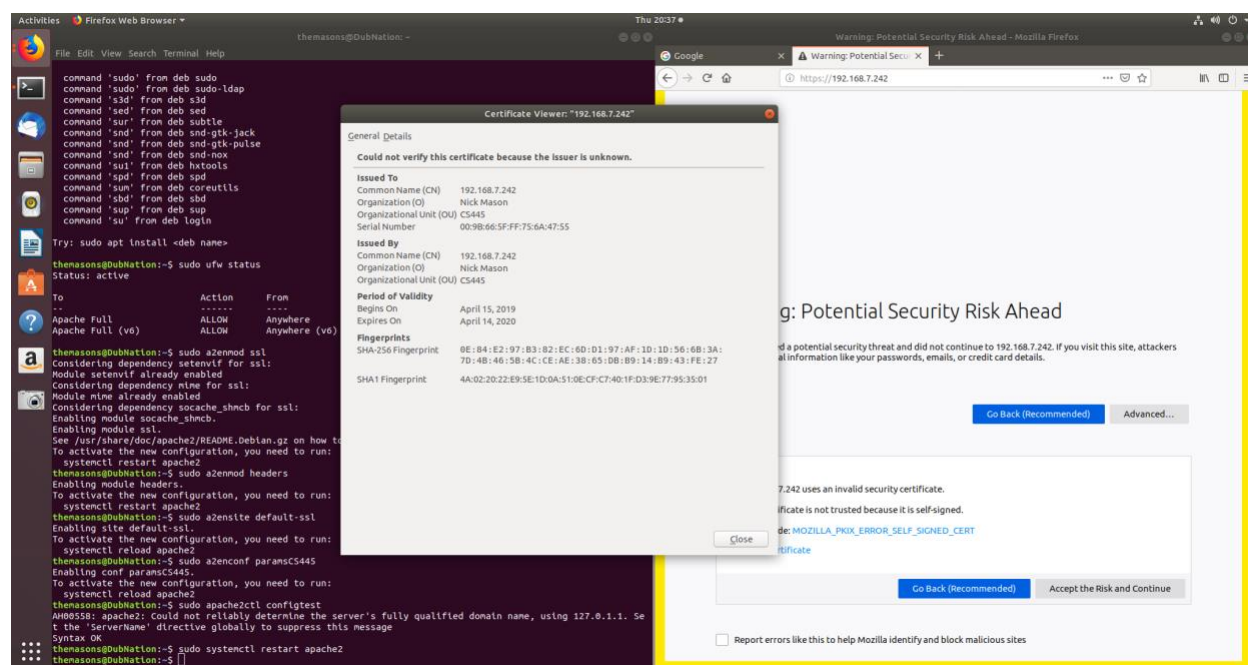


Fig. 6: Figure 6 shows the certificate that is tied to the apache webserver, indicating that it is indeed the self-signed certificate created in this project.

Adding my Self-Signed Certificate to Firefox Web Browser

Finally, I am able to add my newly encrypted apache webserver self-signed certificate to Firefox. First, I went to Settings in Firefox and went to Advanced. From there, I searched for certificates and found the Authorities menu of the Certificate Manager. Next, I imported my self-signed certificate to the Authorities menu. Figure 7 shows where I was able to select the trust settings for my certificate by the web browser. I selected "This certificate can identify websites", and "This certificate can identify mail users" to allow full trust of the certificate.

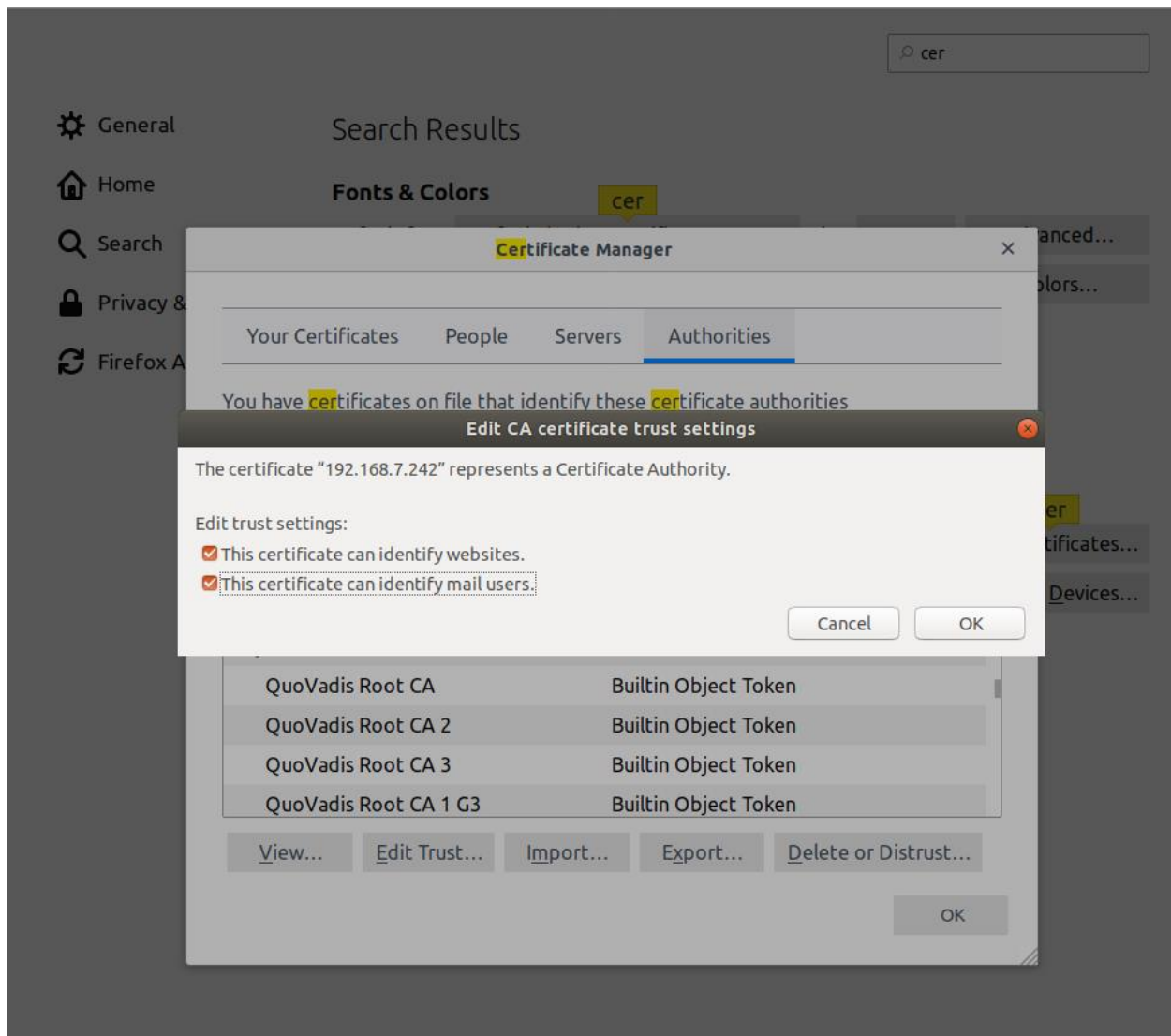


Fig. 7: Figure 7 shows where I was able to allow the Certificate Authority to trust my self-signed certificate.

Once I allowed the Certificate to be trusted by the web browser, I was able to see my certificate in the certificate manager in Firefox, indicating the IP address of my apache webserver. Figure 8 shows this process being completed, resulting in my apache webserver being trusted by Firefox, while also maintaining the encryption I created for it earlier in the project.

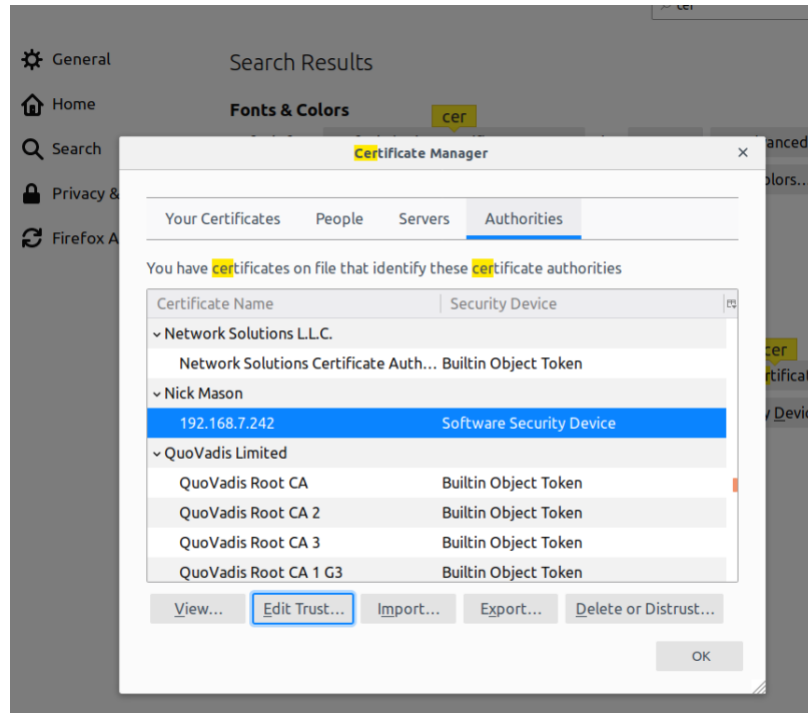


Fig. 8: Figure 8 shows my certificate with my name on it in the Firefox Certificate Manager, showing the IP address of my apache webserver.

Conclusion

Because I was able to encrypt the data on my apache webserver, and create a self-signed certificate, I was able to create security for my data on my webserver. By adding my certificate to my system, and importing it into Firefox, I was able to have my self-signed certificate recognized. This process proves that anyone can create a certificate to secure their personal data from malicious users. A full description of the process I followed for this project, including research, day-to-day progress, and all applicable code, has been included in full on my GitHub.

References

- <https://www.netburner.com/learn/creating-a-self-signed-certificate-for-secure-iot-applications/>
- https://www.openssl.org/docs/manmaster/man3/X509_new.html
- <https://stackoverflow.com/questions/256405/programmatically-create-x509-certificate-using-openssl>
- <https://www.rosehosting.com/blog/how-to-generate-a-self-signed-ssl-certificate-on-linux/>
- <https://www.namecheap.com/support/knowledgebase/article.aspx/798/67/what-is-an-rsa-key-used-for>
- <https://unix.stackexchange.com/questions/90450/adding-a-self-signed-certificate-to-the-trusted-list>
- <https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-apache-in-ubuntu-18-04>
- https://www.bounca.org/tutorials/install_root_certificate.html