HyperionDev

macOS Guide to Web Server Hardening

Additional Reading

Visit our website

Introduction

In this guide, we will walk you through the installation and configuration of the Apache web server on a macOS, followed by the steps to secure the Apache web server.

Development environment

These instructions were developed using the following operating system and software.

Operating system: macOS Sonoma, Version 14.4

Web browser: Safari, Version 17.4

Configure Apache web server on your Mac

The Apache web server comes pre-installed on Mac, providing support for the HTTP protocol by default. In this section we will verify this functionality. It's important to note the web browser's caution regarding inherent security risks associated with continuing to use the insecure HTTP protocol.

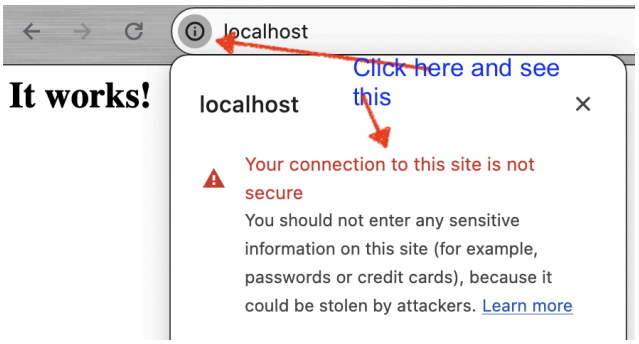
Please note: The Apache web server should be running by default on all later versions of macOS. You can confirm this by issuing the command: `httpd -v`

To ensure the Apache web server is operational and listening on port 80, enter **http://localhost/** in your browser. You should observe a screen resembling the following in the Safari browser:

It works!

If the Apache server is not running, add the `Listen localhost:80` command to **/etc/apache2/httpd.conf** in the section of the configuration file where other `listen` commands are located. Then, reboot the Apache server by executing the `sudo apachectl restart` command.

Now click on the "**i**" icon to the left of "localhost". You should receive the following or a similar display:



The error message indicates your connection is insecure. This is because you are using the HTTP protocol, which lacks encryption, authentication, data integrity, and secure communication channels, meaning all of your conversations can be eavesdropped and tampered with.

Configure a certificate authority on Mac

We will first create a local certificate authority (CA) to prepare for the hardening of the Apache web server. A CA is a commercial, third-party, trusted authority, such as **GoDaddy** and **Verisign**. CAs issue digital certificates, which are used to secure communication between two parties. You typically purchase the services of a CA to obtain such a certificate, ensuring secure communication, for example, between the Apache web server and its clients (i.e., web browsers).

We will establish a CA by creating a "self-signed" certificate. This certificate acts as a template for services like the Apache web server to generate their own certificate. Thus, Apache's web server certificate, signed by the CA, verifies the server's identity to web browsers requesting its service(s).

It is important to note that a "self-signed" CA certificate should only be used in lab environments. For production environments, it is necessary to use the services of a commercial CA to obtain a digital certificate.

OpenSSL toolkit

To "harden" the Apache web server, you will need to download the OpenSSL toolkit using the following instructions:

1. Open the terminal.
2. Determine if OpenSSL is installed on your Mac by issuing: `openssl version`. If not, install it by issuing: `brew install openssl`
3. Navigate to **/private/etc/apache2**.

4. `sudo openssl req -newkey rsa:2048 -nodes -keyout server.key -out server.csr -subj "/CN=localhost"` will generate a new private key (`server.key`) and certificate signing request (`server.csr`), both explained below. “localhost” is used as the common name (CN), which is a certificate field that identifies the entity for which the certificate is created. In most cases, an SSL/TLS certificate is used for secure websites (HTTPS), and the CN will typically be your organisation's website domain name. As we are only creating a self-signed certificate for test purposes, we will define the CN as “localhost”.

   Here is an explanation of each parameter:

   - `-newkey rsa:2048` specifies that a new private key and certificate signing request (CSR) will be generated. The rsa:2048 option indicates that an RSA key with a length of 2048 bits will be used. This part of the command combines the key generation and CSR creation into a single step.
   - `-keyout server.key` specifies the filename (**server.key**) where the generated private key will be saved.
   - `-out server.csr` specifies the filename (**server.csr**) where the generated CSR will be saved. The CSR contains information about the entity requesting the certificate, such as the CN and organisation.
   - `-subj "/CN=localhost"` specifies the subject (i.e., information about the entity) for the CSR. In this case, the subject is set to `/CN=localhost`, where `/CN=` indicates the CN field of the certificate. The CN is typically the domain name or hostname of the entity for which the certificate is being issued. In this example, the CN is set to “localhost”, indicating that the certificate will be valid for the domain name “localhost”.

   When you create the CA root private key by issuing `sudo openssl req -newkey rsa:2048 -nodes -keyout server.key -out server.csr -subj "/CN=localhost"`

   The sample output should be:

```
[Password:
Generating a 2048 bit RSA private key
..+++++
.................. +++++
writing new private key to 'server.key'
```

5. `sudo openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt` is used to perform operations related to X.509 certificates, including the signing, verifying, and manipulating of certificates.

   Here is an explanation of each parameter:

   - `-req` specifies that the input file specified in this command (**server.csr**) is a CSR rather than a certificate.
   - `-days 365` specifies the validity period of the generated certificate in days. In this case, the certificate will be valid for 365 days from the time it is generated.
   - `-in server.csr` specifies the input file containing the CSR (**server.csr**). The CSR contains the information required to generate the certificate, including the public key and details about the entity requesting the certificate.
   - `-signkey server.key` specifies the private key (**server.key**) that will be used to sign the certificate. The private key is necessary for generating a self-signed certificate.
   - `-out server.crt` specifies the output file where the generated certificate will be saved. In this case, the certificate will be saved as **server.crt**.

   When you create a CA self-signed certificate by issuing: `sudo openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt`

   The sample output should be:

   ```
   Signature ok

   subject=/CN=localhost

   Getting Private key
   ```

Secure the Apache web server with the SSL/TLS using the CA certificate

Now, let's discuss the process of hardening the Apache web server, which involves implementing the SSL/TLS protocol, which is dependent on the digital certificate you created above. When a web browser connects to the website via **https://**, it utilises the HTTPS protocol, and the Apache web server presents its digital certificate, issued by the local CA, to the browser. After the browser validates the certificate's validity, the SSL protocol establishes an encrypted and secure communication channel between the client and the server, protecting all data from eavesdropping and tampering.

In summary, the SSL protocol will provide the following critical security functions:

- Integrity: Data in transit remains intact and unaltered.
- Confidentiality: Data in transit is encrypted and, as such, cannot be eavesdropped.
- Authentication: Digital certificates will verify the identity of the servers, reducing the risk of man-in-the-middle attacks.

For more information on SSL, visit this **web page**.

To configure the Apache web server, follow these steps:

1. Edit the Apache configuration file with: `sudo nano /private/etc/apache2/httpd.conf`
2. Uncomment or add the following lines to enable SSL encryption support in Apache: `LoadModule ssl_module libexec/apache2/mod_ssl.so`, `Include /private/etc/apache2/extra/httpd-ssl.conf`
3. Uncomment or place the following command in **/etc/apache2/httpd.conf**: `Listen localhost:443`
4. If the following command or similar exists in the file **/private/etc/apache2/extra/httpd-ssl.conf**, comment it out: `SSLSessionCache "shmcb:/private/var/run/ssl_scache(512000)"`

   The file referenced does not natively exist and is used only for performance reasons. If this command stays in the **.conf** file, Apache will not start.

5. Restart the Apache web server with: `sudo apachectl restart`

Take note

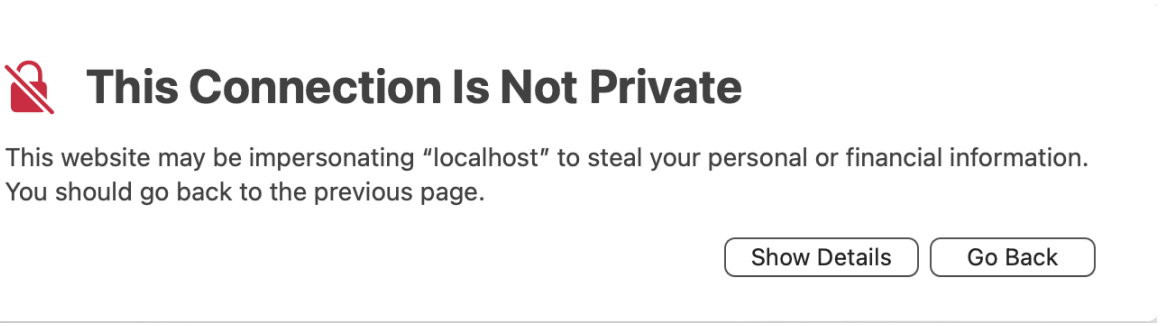
If you get stuck bringing up the Apache web server, here are some suggestions:

- Issue `sudo apachectl stop` to stop Apache, then issue `sudo apachectl -e` to restart in debug mode.
- Analyse the debug file in **/var/log/apahe2/error_log** or wherever the log file is on your Mac.
- Confirm that Apache is listening on ports 80 (HTTP) and 443 (HTTPS) via the following command: `sudo Isof-i -P| grep LISTEN`

You will now access the Apache web server via HTTPS, which uses the SSL protocol mentioned above.

Access the Apache web server securely

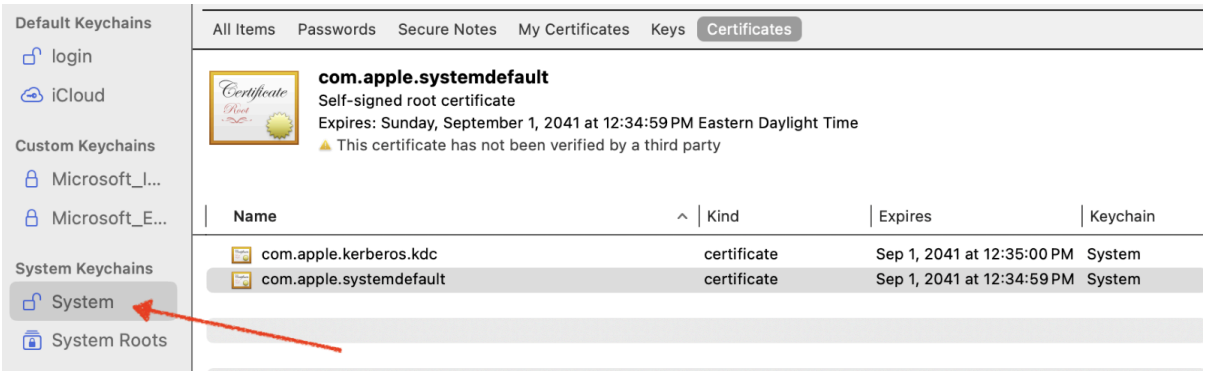
1. In your Safari browser, navigate to https://localhost. You should see output similar to this:



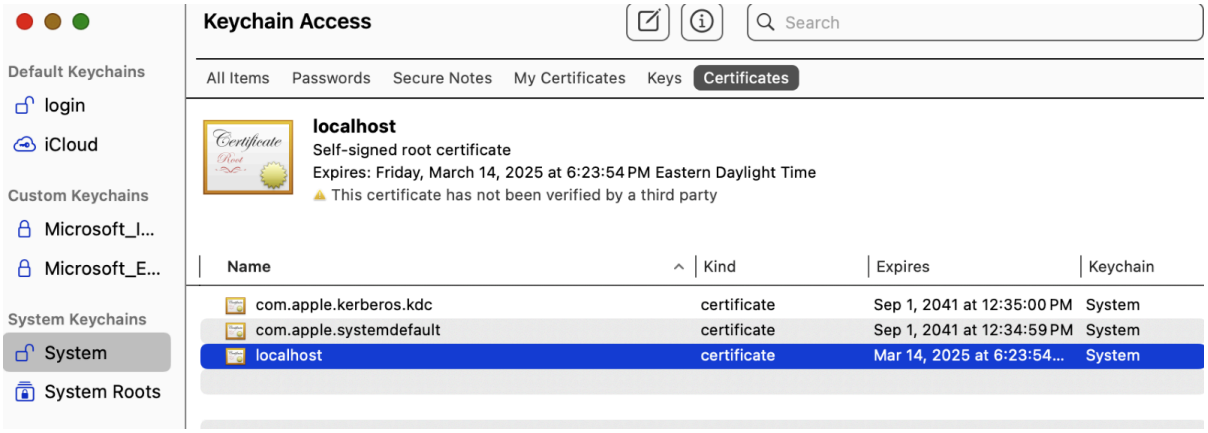
This warning appears because your browser doesn't recognise the SSL certificate since it hasn't been issued from what it considers a trusted CA (e.g., Verisign). This is expected behaviour since we have created a self-signed certificate that your laptop does not recognise.

Therefore, you must now add the certificate to your Mac's list of trusted certificates. You can do this by accessing the Keychain Access app manually and adding the self-signed certificate to your browser's list of trusted certificates.

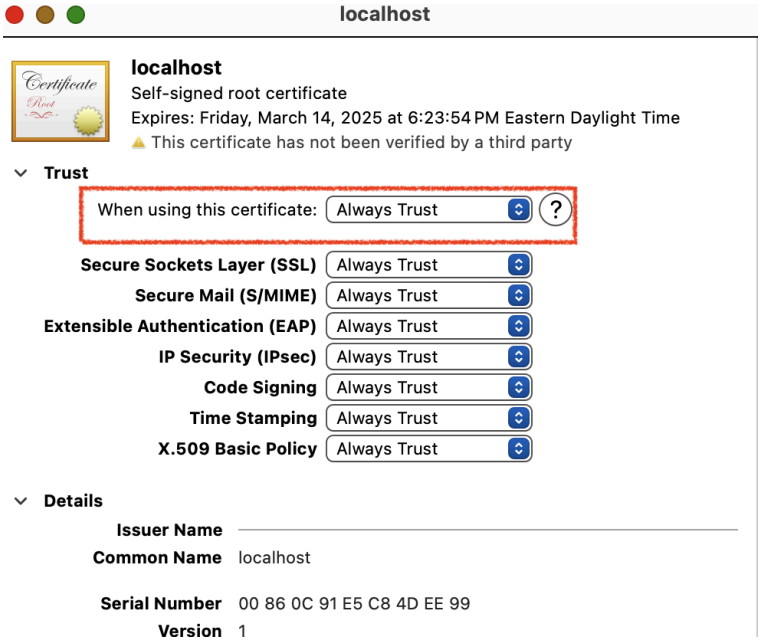
2. Copy the certificate /private/etc/apache2/server.crt file to your desktop directory.
3. Open Keychain Access.
4. Unlock the “System” under “System Keychains”:



5. From Mac Finder, drag the server.crt file on your desktop into the System Keychains. Specify your system password if asked.

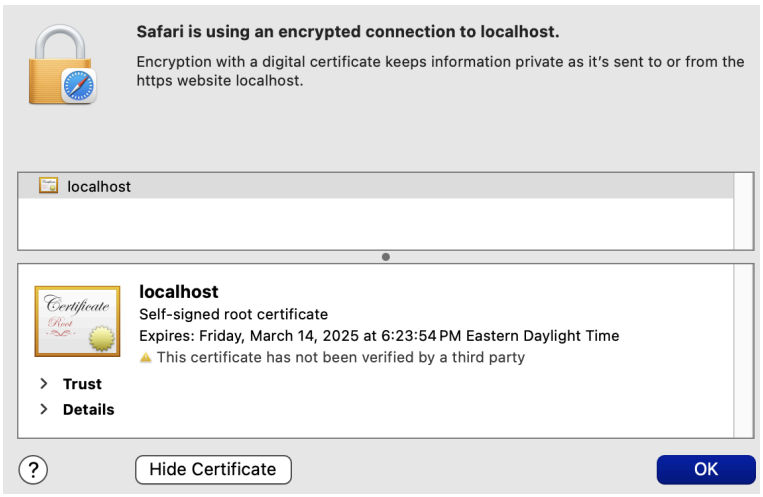


6. Double-click on the “localhost” certificate and specify “Always Trust” next to “When using this certificate” under the “Trust” settings:

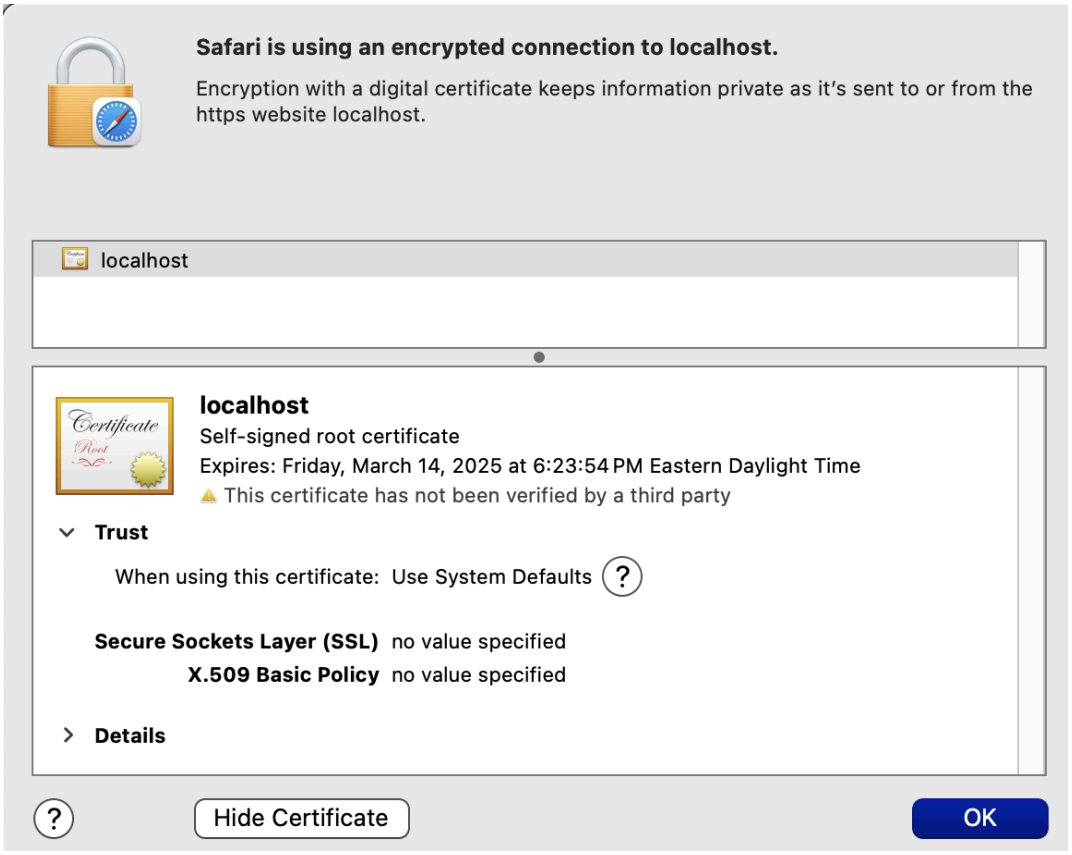


7. In your Safari browser, refresh or open a new session to https://localhost. You should see the following:

Safari now recognises the SSL certificate as trusted. Let’s look at the certificate itself. Click on the padlock next to “localhost” in the URL, and click on “Show Certificate”. You should see the following:



8. Now, click on the arrow next to “Trust” and you should see the following details:



Take note

Things to note when analysing the digital certificate:

- Your browser now recognises this certificate as being valid and, as such, the user is assured that this connection is authenticated, encrypted, and has not been tampered with, and is therefore a secure connection.
- The “Connection Name” value has been set to what was specified when the certificate was created.
- The validity duration of the certificate is shown.
- In a "real-world" production environment, as a representative of an organisation you would employ the services of a commercial CA such as GoDaddy or Verisign to obtain a valid certificate that would be recognised by default by your web browser.

Summary

In this document, you have seen how using the SSL/TLS protocols, in conjunction with a digital certificate, ensures encrypted communications between a client (e.g., web browser) and server (e.g., web server), safeguarding against interception. The PKI certificate you created establishes trust by verifying the server's identity, thus mitigating risks associated with unauthorised access or man-in-the-middle attacks prevalent in HTTP-based connections.