Securing Apache with HTTPS

Andrew Mallett
LINUX AUTHOR AND TRAINER
@theurbanpenguin www.theurbanpenguin.com

Objectives

- SSL/TLS Overview
- Generate Server Private Key
- Generate a Certificate Signing Request (CSR)
- Signing Requests
- Deploying SSL on Apache
- Redirect HTTP to HTTPS and HSTS
- Free Signed Certificates from Linux Foundations' "Let's Encrypt" CA

SSL/TLS Timeline

SSLv2 1994	SSLv3 1995	TLSv1.0 1999
TLSv1.1 2006	TLSv1.2 2008	TLSv1.3 2017?

SSL or TLS

The SSL protocol is very much defunct nowadays. We can see this from the timeline. The name TLS as a replacement to SSL was more driven by politics rather than any key differences. Many people use the term SSL when in fact they are referring to the TLS protocol. Both work at the Presentation Layer (6) of the OSI Model.

TLS Provides the CIA Triad

Confidentiality: Encryption - keeping data secure across public networks including the infamous coffee-shop WiFi

Integrity: Has the data been interfered with since leaving the server. Has my mobile network provider inserted advertisements in a third-party site that I am viewing across their network

Authenticity: Are we talking to the correct server and when I enter my password is it being sent to the server I think I am sending it to

```
# openssl version # Make sure version is later that 1.0.1f
# cd /etc/httpd/conf
# openssl genrsa -out server.key 2048 # Don't use < 2048
# cat server.key # Looks quite simple
# openssl rsa -noout -text -in server.key # Until…
```

Generate Server Private Key

End-Entities such as HTTPS Servers will require their own public key and private key pair. We start by generating the private key. This should be kept securely.

```
# cd /etc/httpd/conf
# openssl req -new -key server.key -out server.csr
# cat server.csr
# openssl req -noout -text -in server.csr
```

Generate the CSR

The CSR or Certificate Signing Request creates the End-Entities Public Key. As the name suggests it will need to be signed before it is a fully functional x509 certificate.

```
# cd /etc/httpd/conf

# openssl x509 -req -sha256 -in server.csr \
  -signkey server.key -out server.crt

# cat server.crt

# openssl x509 -noout -text -in server.crt

# man x509 ; man req; man genrsa
```

Self-Sign the CSR

The Private Key is also an Untrusted-CA that can be used to sign CSRs. A Self-Signed Certificate is ok but will produce Browser UI warnings.

Configuration

```
Listen 443

LoadModule ssl_module modules/mod_ssl.so

SSLCipherSuite HIGH:MEDIUM:!3DES:!RC4:!MD5:!SSLv3:!SSLv2

SSLHonorCipherOrder on

SSLProtocol all -SSLv2 -SSLv3
```

```
openssl ciphers -V 'HIGH:MEDIUM:!aRSA'
```

Working with Ciphers

If we want to know what ciphers we have allowed in our filter we can use the openssl ciphers subcommand

Virtual Host Blocks

```
<VirtualHost *:443>
  ServerName pilabs.theurbanpenguin.com
  DocumentRoot “/srv/http”
  DirectoryIndex “index.html”
  SSLEngine on
  SSLCertificateFile “conf/server.crt”
  SSLCertificateKeyFile “conf/server.key”
  <Directory “/srv/http”>
    Require all granted
  </Directory>
</VirtualHost>
```

Let’s Encrypt

New CA founded in 2015 to ensure all traffic on the Internet can be encrypted and secured. This is partly funded by the Linux Foundation.

```
# cd ; pacman -S git python wget
# git clone git://github.com/diafygi/acme-tiny
# cp acme-tiny/acme_tiny.py /usr/local/bin/
# chmod +x /usr/local/bin/acme_tiny.py
```

Obtain Acme-Tiny

Certificate requests are automated via the ACME protocol. A simple ACME client is the Python Script acme_tiny.py.

```
# mkdir -p /srv/http/.well-known/acme-challenge

# cd /etc/httpd/conf ; openssl genrsa -out le.key 2048

# acme-tiny.py --account-key le.key \
  --csr server.csr \
  --acme-dir /srv/http/.well-known/acme-challenge/ \
  > server.crt
```

Configure the Challenge Directory

The ACME Protocol is there to ensure you can control the site that you requested a certificate for. Let's Encrypt will put a challenge file in the web site. We need to create that structure.

Gaining a Signed x509 Certifcate

```
# wget http://cert.init-x3.letsencrypt.org/ -O issue.der

# openssl x509 -in issue.der -inform DER \
  -out issue.crt -outform PEM

# cat issue.crt >> server.crt

# apachectl configtest && systemctl restart httpd
```

Certificate Chain

Our Web Server needs to present the complete certificate chain to the browser. The browser has the certificate for the Root CA but not from the issuing intermediate CA. We need to download this and merge it into the server's x509 Certificate

```
Redirect permanent / https://pilabs.theurbanpenguin.com
```

Redirect HTTP to HTTPS

If user just enters a URL without a method then the browser defaults to HTTP. We can redirect those request to HTTPS. Within the Port 80 Virtual Host we can add the above Redirect.

```
LoadModule headers_module modules/mod_headers.so

Header always set Strict-Transport-Security “max-
age=31536000; includeSubDomains"
```

HTTPS Strict Transport Security

Using HSTS we can tell the browser to only use HTTPS to our site; this also ensures that Certificate Warnings cannot be overridden by the user. We set for 1 year here but choose a smaller number at first to test.

Summary

openssl

- Generate key, csr, and sign

Configurations

- Virtual Host on Port 80 and 443

Let's Encrypt

- Free CA
- acme_tiny.py

Redirection

- Redirect in http site to https
- HSTS adds rule to browser for added security

Next up: Load Balancing HTTP Requests