Hardening TLS with Apache as a reverse proxy

Aug 6, 2015 – By: Tim Wells

This article documents how I configured Apache as a reverse proxy to IIS to harden TLS encryption. This could also be applied to other applications on Windows or Linux servers with a few modifications.

I try to use the best encryption available for TLS connections on all of the web servers I work with. One of the servers has Windows Server 2008 and IIS 7.0, which does not support the newer and more secure protocols and cipher suites. I do not have the authority to upgrade the operating system on this server, so I had started a discussion with the host about upgrading the server to Windows Server 2008 R2. Then I got an idea: maybe Apache or PHP could somehow act as a reverse proxy to the IIS server. I already knew that I could configure Apache to use the more secure protocols and cipher suites. It turns out that Apache can function as a reverse proxy on its own. I also did this for a web application on an Ubuntu Linux server that did not support the newer and more secure protocols and cipher suites.

The first step is to install Apache on the system (if it isn't already installed). On the Windows server I used XAMPP and only installed the Apache module.

The following modules need to be enabled

- mod_headers (optional - needed to use the RequestHeader or Header directives)
- mod_proxy
- mod_proxy_http

To enable these modules in XAMPP, uncomment the following lines in C:\xampp\apache\conf\httpd.conf (use the a2enmod command in Ubuntu).

```
LoadModule headers_module modules/mod_headers.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
```

Next, change Apache's HTTP listen port so that it does not conflict with IIS by modifying the following line in C:\xampp\apache\conf\httpd.conf (/etc/apache2/apache2.conf in Ubuntu). This port does not need to be open to the internet.

```
Listen 8080
```

Next, add the following lines to the SSL config file in C:\xampp\apache\conf\extra\httpd-ssl.conf (/etc/apache2/mods-available/ssl.conf in Ubuntu). Also add a # before any existing SSLCipherSuite and SSLProtocol directives to comment them out.

```
# Custom TLS Cipher Suite configuration for optimized security
      SSLProtocol -SSLv2 -SSLv3 +TLSv1 +TLSv1.1 +TLSv1.2
      SSLHonorCipherOrder on
```

```
      SSLCipherSuite "ECDHE-RSA-AES256-GCM-SHA384 ECDHE-RSA-
AES256-SHA384 ECDH-RSA-AES256-GCM-SHA384 ECDH-RSA-AES256-SHA384
DHE-RSA-AES256-GCM-SHA384 DHE-RSA-AES256-SHA256 ECDHE-RSA-
AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDH-RSA-AES128-GCM-
SHA256 ECDH-RSA-AES128-SHA256 DHE-RSA-AES128-GCM-SHA256 DHE-RSA-
AES128-SHA256 ECDHE-RSA-AES256-SHA ECDH-RSA-AES256-SHA DHE-RSA-
AES256-SHA ECDHE-RSA-AES128-SHA ECDH-RSA-AES128-SHA DHE-RSA-
AES128-SHA DHE-RSA-CAMELLIA256-SHA DHE-RSA-CAMELLIA128-SHA
AES256-GCM-SHA384 AES256-SHA256 AES128-GCM-SHA256 AES128-SHA256
AES256-SHA AES128-SHA CAMELLIA256-SHA CAMELLIA128-SHA"

      SSLCompression Off
      SSLOptions +StdEnvVars
      SSLUseStapling on
      SSLStaplingCache "shmcb:logs/stapling-cache(150000)"
```

This configuration will enable TLS 1.2, TLS 1.1 and TLS 1.0. It prioritizes the TLS Cipher Suites in this order: Forward Secrecy suites first, SHA2 before SHA1, AES before Camellia, higher Encryption Bits first, higher SHA2 bits first, GCM before CBC. The only algorithms used are AES and Camellia. RC4 and 3DES are not included because they are considered to be weaker. It also enables OCSP stapling. This configuration scores an A on the SSL Labs website (A+ if you use HSTS).

Next setup the virtual host by adding the following lines to C:\xampp\apache\conf\extra\httpd-vhosts.conf (Ubuntu uses a separate file for each host in /etc/apache2/sites-enabled). You will need to modify the path to the SSL certificates and key.

```
<VirtualHost *:443>

      # Reverse Proxy to IIS
      ProxyPreserveHost On
      ProxyPass / http://127.0.0.1/
      ProxyPassReverse / http://127.0.0.1/
      ProxyVia Full
      ProxyTimeout 30
      ProxyRequests off

      # TLS
      SSLProxyEngine on
      SSLEngine on
      SSLProtocol +TLSv1 +TLSv1.1 +TLSv1.2
      SSLCertificateFile "conf/ssl.crt/SSL_CERTIFICATE.crt"
      SSLCertificateKeyFile "conf/ssl.key/PRIVATE_KEY.key"
      SSLCACertificateFile "conf/ssl.ca/CA_INTERMEDIATES.pem"
```

```
        # Pass SSL Info to Web Application in Request Headers
(optional - requires mod_headers)
        RequestHeader set X-SSL-Protocol "%{SSL_PROTOCOL}s"
        RequestHeader set X-SSL-Cipher "%{SSL_CIPHER}s"

</VirtualHost>
```

The next step is to remove port 443 from the IIS site bindings. I recommend moving HTTPS to another port (not open to the internet) so this change can be easily reverted. I also created another HTTP port (not open to the internet) for the proxy requests so that the web application can distinguish between HTTP and HTTPS via proxy connections based on the port, and automatically redirect any HTTP requests to the HTTPS proxy. You may need to restart IIS or reboot the server to free port 443.

Now you can start (or restart) Apache. I recommend installing it as a service in XAMPP.

If the web application has redirects, you may need to add the following line to the VirtualHost. I had to add this for the web application on Ubuntu because its redirects went back to HTTP. This basically replaces http:// with https:// in the location header.

```
Header edit location ^http://(.*)$ https://$1
```

If the web application links back to the HTTP site, you may need to use mod_proxy_html to rewrite HTML links to go through the proxy. Note: mod_xml2enc should be enabled when using mod_proxy_html.

Apache is capable of doing even more as a proxy, including load balancing and caching.

Additional Resources:

- Apache HTTP Server Documentation - mod_proxy Module
- Apache HTTP Server Documentation - mod_headers Module
- Apache HTTP Server Documentation - mod_ssl Module
- Apache HTTP Server Documentation - mod_proxy_html Module