Enabling HTTPS on Your Servers

By Chris Palmer

Chris is a security engineer on the Chrome Security Team, focusing on secure usability.

By Matt Gaunt

Matt is a contributor to Web**Fundamentals**

TL;DR

- Create a 2048-bit RSA public/private key pair.
- Generate a certificate signing request (CSR) that embeds your public key.
- Share your CSR with your Certificate Authority (CA) to receive a final certificate or a certificate chain.
- Install your final certificate in a non-web-accessible place such as `/etc/ssl` (Linux and Unix) or wherever IIS requires it (Windows).

Generating keys and certificate signing requests

This section uses the openssl command-line program, which comes with most Linux, BSD, and Mac OS X systems, to generate private/public keys and a CSR.

Generate a public/private key pair

Let's start by generating a 2,048-bit RSA key pair. A smaller key, such as 1,024 bits, is insufficiently resistant to brute-force guessing attacks. A larger key, such as 4,096 bits, is overkill. Over time, key sizes increase as computer processing gets cheaper. 2,048 is currently the sweet spot.

The command to generate the RSA key pair is:

```
openssl genrsa -out www.example.com.key 2048
```

This gives the following output:

```
Generating RSA private key, 2048 bit long modulus
.+++
.................................................................................
e is 65537 (0x10001)
```

Generate a certificate signing request

In this step, you embed your public key and information about your organization and your website into a certificate signing request or CSR. The *openssl* command interactively asks you for the required metadata.

Running the following command:

```
openssl req -new -sha256 -key www.example.com.key -out www.example.com.csr
```

Outputs the following:

```
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]:CA
State or Province Name (full name) [Some-State]:California
Locality Name (for example, city) []:Mountain View
Organization Name (for example, company) [Internet Widgits Pty Ltd]:Example, Inc.
Organizational Unit Name (for example, section) []:Webmaster Help Center Example
Team
Common Name (e.g. server FQDN or YOUR name) []:www.example.com
Email Address []:webmaster@example.com

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

To ensure the validity of the CSR, run this command:

```
openssl req -text -in www.example.com.csr -noout
```

And the response should look like this:

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=CA, ST=California, L=Mountain View, O=Google, Inc.,
OU=Webmaster Help Center Example Team,
CN=www.example.com/emailAddress=webmaster@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ad:fc:58:e0:da:f2:0b:73:51:93:29:a5:d3:9e:
                    f8:f1:14:13:64:cc:e0:bc:be:26:5d:04:e1:58:dc:
                    ...
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         5f:05:f3:71:d5:f7:b7:b6:dc:17:cc:88:03:b8:87:29:f6:87:
         2f:7f:00:49:08:0a:20:41:0b:70:03:04:7d:94:af:69:3d:f4:
         ...
```

Submit your CSR to a certificate authority

Different certificate authorities (CAs) require different methods for sending them your CSRs. Methods may include using a form on their website, sending the CSR by email, or something else. Some CAs (or their resellers) may even automate some or all of the process (including, in some cases, key pair and CSR generation).

Send the CA to your CSR, and follow their instructions to receive your final certificate or certificate chain.

Different CAs charge different amounts of money for the service of vouching for your public key.

There are also options for mapping your key to more than one DNS name, including several distinct names (e.g. all of example.com, www.example.com, example.net, and www.example.net) or "wildcard" names such as *.example.com.

For example, one CA currently offers these prices:

- Standard: $16/year, valid for example.com and www.example.com.
- Wildcard: $150/year, valid for example.com and *.example.com.

At these prices, wildcard certificates are economical when you have more than 9 subdomains; otherwise, you can just buy one or more single-name certificates. (If you have more than, say, five subdomains, you might find a wildcard certificate more convenient when you come to enable HTTPS on your servers.)

Note: Keep in mind that in wildcard certificates the wildcard applies to only one DNS label. A certificate good for *.example.com will work for foo.example.com and bar.example.com, but *not* for foo.bar.example.com.

Copy the certificates to all your front-end servers in a non-web-accessible place such as `/etc/ssl` (Linux and Unix) or wherever IIS (Windows) requires them.

Enable HTTPS on your servers

Enabling HTTPS on your servers is a critical step in providing security for your web pages.

- Use Mozilla's Server Configuration tool to set up your server for HTTPS support.
- Regularly test your site with the Qualys' handy SSL Server Test and ensure you get at least an A or A+.

At this point, you must make a crucial operations decision. Choose one of the following:

- Dedicate a distinct IP address to each hostname your web server serves content from.
- Use name-based virtual hosting.

If you have been using distinct IP addresses for each hostname, you can easily support both HTTP and HTTPS for all clients.

However, most site operators use name-based virtual hosting to conserve IP addresses and because it's more convenient in general. The problem with IE on Windows XP and Android earlier than 2.3 is that they do not understand Server Name Indication (SNI), which is crucial for HTTPS name-based virtual hosting.

Someday—hopefully soon—clients that don't support SNI will be replaced with modern software. Monitor the user agent string in your request logs to know when enough of your user population has migrated to modern software. (You can decide what your threshold is; perhaps < 5%, or < 1%.)

If you don't already have HTTPS service available on your servers, enable it now (without redirecting HTTP to HTTPS; see below). Configure your web server to use the certificates you bought and installed. You might find Mozilla's handy configuration generator useful.

If you have many hostnames/subdomains, they each need to use the right certificate.

Warning: If you've already completed these steps, but are using HTTPS for the sole purpose of redirecting clients back to HTTP, stop doing that now. See the next section to make sure HTTPS and HTTP work smoothly.

Note: Ultimately you should redirect HTTP requests to HTTPS and use HTTP Strict Transport Security (HSTS). However, this is not the right stage in the migration process to do that; see “Redirect HTTP To HTTPS” and “Turn On Strict Transport Security And Secure Cookies.”

Now, and throughout your site's lifetime, check your HTTPS configuration with Qualys' handy SSL Server Test. Your site should score an A or A+; treat anything that causes a lower grade as a bug. (Today's A is tomorrow's B, because attacks against algorithms and protocols are always improving!)

Make intrasite URLs relative

Now that you are serving your site on both HTTP and HTTPS, things need to work as smoothly as possible, regardless of protocol. An important factor is using relative URLs for intrasite links.

Make sure intrasite URLs and external URLs are agnostic to protocol; that is, make sure you use relative paths or leave out the protocol like `//example.com/something.js`.

A problem arises when you serve a page via HTTPS that includes HTTP resources, known as mixed content. Browsers warn users that the full strength of HTTPS has been lost. In fact, in the case of active mixed content (script, plug-ins, CSS, iframes), browsers often simply won't load or execute the content at all, resulting in a broken page. And remember, it's perfectly OK to include HTTPS resources in an HTTP page.

Key Point: See Fixing Mixed Content for more details about ways to fix and prevent mixed content.

Additionally, when you link to other pages in your site, users could get downgraded from HTTPS to HTTP.

These problems happen when your pages include fully-qualified, intrasite URLs that use the *http://* scheme.

Not recommended — We recommend you avoid using fully qualified intrasite URLs.

```
<h1>Welcome To Example.com</h1>
<script src="http://example.com/jquery.js"></script>
<link rel="stylesheet" href="http://assets.example.com/style.css"/>
<img src="http://img.example.com/logo.png"/>;
<p>A <a href="http://example.com/2014/12/24/">new post on cats!</a></p>
```

In other words, make intrasite URLs as relative as possible: either protocol-relative (lacking a protocol, starting with `//example.com`) or host-relative (starting with just the path, like `/jquery.js`).

Recommended — We recommend that you use relative intrasite URLs.

```
<h1>Welcome To Example.com</h1>
<script src="/jquery.js"></script>
<link href="/styles/style.css" rel="stylesheet"/>
<img src="/images/logo.png"/>;
<p>A <a href="/2014/12/24/">new post on cats!</a></p>
```

Recommended — Or, you can use protocol-relative intrasite URLs.

```
<h1>Welcome To Example.com</h1>
<script src="//example.com/jquery.js"></script>
<link href="//assets.example.com/style.css" rel="stylesheet"/>
<img src="//img.example.com/logo.png"/>;
<p>A <a href="//example.com/2014/12/24/">new post on cats!</a></p>
```

Recommended — We recommend that you use HTTP**S** URLs for intersite URLs (where possible).

```
<h1>Welcome To Example.com</h1>
<script src="/jquery.js"></script>
<link href="/styles/style.css" rel="stylesheet"/>
<img src="/images/logo.png"/>;
<p>A <a href="/2014/12/24/">new post on cats!</a></p>
<p>Check out this <a href="https://foo.com/">other cool site.</a></p>
```

Do this with a script, not by hand. If your site's content is in a database, test your script on a development copy of your database. If your site's content consists of simple files, test your script on a development copy of the files. Push the changes to production only after the changes pass QA, as normal. You can use Bram van Damme's script or something similar to detect mixed content in your site.

When linking to other sites (as opposed to including resources from them), don’t change the protocol since you don’t have control over how those sites operate.

Success: To make migration smoother for large sites, we recommend protocol-relative URLs. If you are not sure whether you can fully deploy HTTPS yet, forcing your site to use HTTPS for all sub-resources may backfire. There is likely to be a period of time in which HTTPS is new and weird for you, and the HTTP site must still work as well as ever. Over time, you’ll complete the migration and lock in HTTPS (see the next two sections).

If your site depends on scripts, images, or other resources served from a third party, such as a CDN or jquery.com, you have two options:

- Use protocol-relative URLs for these resources. If the third party does not serve HTTPS, ask them to. Most already do, including jquery.com.
- Serve the resources from a server that you control, and which offers both HTTP and HTTPS. This is often a good idea anyway, because then you have better control over your site's appearance, performance, and security. In addition, you don't have to trust a third party, which is always nice.

Note: Keep in mind that you also need to change intrasite URLs in your stylesheets, JavaScript, redirect rules, `<link>` tags, and CSP declarations, not just in the HTML pages.

Redirect HTTP to HTTPS

You need to put a canonical link at the head of your page to tell search engines that HTTPS is the best way to get to your site.

Set `<link rel="canonical" href="https://…"/>` tags in your pages. This helps search engines determine the best way to get to your site.

Turn on Strict Transport Security and secure cookies

At this point, you are ready to "lock in" the use of HTTPS.

- Use HTTP Strict Transport Security (HSTS) to avoid the cost of the 301 redirect.
- Always set the Secure flag on cookies.

First, use Strict Transport Security to tell clients that they should always connect to your server via HTTPS, even when following an `http://` reference. This defeats attacks such as SSL Stripping, and also avoids the round-trip cost of the `301 redirect` that we enabled in Redirect HTTP to HTTPS.

Note: Clients that have noted your site as a known HSTS Host are likely to *hard-fail* if your site ever has an error in its TLS configuration (such as an expired certificate). HSTS is explicitly designed this way to ensure that network attackers cannot trick clients into accessing the site without HTTPS. Do not enable HSTS until you are certain that your site operation is robust enough to avoid ever deploying HTTPS with certificate validation errors.

Turn on HTTP Strict Transport Security (HSTS) by setting the `Strict-Transport-Security` header. OWASP's HSTS page has links to instructions for various server software.

Most web servers offer a similar ability to add custom headers.

Note: `max-age` is measured in seconds. You can start with low values and gradually increase the `max-age` as you become more comfortable operating an HTTPS-only site.

It is also important to make sure that clients never send cookies (such as for authentication or site preferences) over HTTP. For example, if a user's authentication cookie were to be exposed in plain text, the security guarantee of their entire session would be destroyed—even if you have done everything else right!

Therefore, change your web application to always set the Secure flag on cookies that it sets. This OWASP page explains how to set the Secure flag in several application frameworks. Every application framework has a way to set the flag.

Most web servers offer a simple redirect feature. Use `301 (Moved Permanently)` to indicate to search engines and browsers that the HTTPS version is canonical, and redirect your users to the HTTPS version of your site from HTTP.

Migration concerns

Many developers have legitimate concerns about migrating from HTTP to HTTPS. The Google Webmasters Team has some excellent guidance available.

Search ranking

Google uses HTTPS as a positive search quality indicator. Google also publishes a guide for how to transfer, move, or migrate your site while maintaining its search rank. Bing also publishes guidelines for webmasters.

Performance

When the content and application layers are well-tuned (see Steve Souders' books for great advice), the remaining TLS performance concerns are generally small, relative to the overall cost of the application. Additionally, you can reduce and amortize those costs. (For great advice on TLS optimization and generally, see High Performance Browser Networking by Ilya Grigorik.) See also Ivan Ristic's OpenSSL Cookbook and Bulletproof SSL And TLS.

In some cases, TLS can *improve* performance, mostly as a result of making HTTP/2 possible. Chris Palmer gave a talk on HTTPS and HTTP/2 performance at Chrome Dev Summit 2014.

Referer headers

When users follow links from your HTTPS site to other HTTP sites, user agents don't send the Referer header. If this is a problem, there are several ways to solve it:

- The other sites should migrate to HTTPS. If referee sites can complete the Enable HTTPS on your servers section of this guide, you can change links in your site to theirs from `http://` to `https://`, or you can use protocol-relative links.
- To work around a variety of problems with Referer headers, use the new Referrer Policy standard.

Because search engines are migrating to HTTPS, in the future you are likely see *more* Referer headers when you migrate to HTTPS.

Caution: According to the HTTP RFC, clients **SHOULD NOT** include a Referer header field in a (non-secure) HTTP request if the referring page is transferred with a secure protocol.

Ad revenue

Site operators that monetize their site by showing ads want to make sure that migrating to HTTPS does not reduce ad impressions. But due to mixed content security concerns, an HTTP `<iframe>` doesn't work in an HTTPS page. There is a tricky collective action problem here: until advertisers publish over HTTPS, site operators cannot migrate to HTTPS without losing ad revenue; but until site operators migrate to HTTPS, advertisers have little motivation to publish HTTPS.

Advertisers should at least offer ad service via HTTPS (such as by completing the "Enable HTTPS on your servers" section on this page. Many already do. You should ask advertisers that do not serve HTTPS at all to at least start. You may wish to defer completing Make IntraSite URLs relative until enough advertisers interoperate properly.

Previous
← Important Security Terminology