Robert Hurlbut

Software Security Architect
(ISC)2 CSSLP 2014-2017
Co-host with Chris Romeo – Application Security Podcast

Contacts
Web Site: https://roberthurlbut.com
Twitter: @RobertHurlbut, @AppSecPodcast

© 2017 Robert Hurlbut
SSL / TLS

If not already, consider HTTPS (TLS 1.1 or 1.2)

Make sure all site is using HTTPS

Use strong certificate – at least SHA-256, 2048 bit key (no SHA-1, SSL 1, etc.)

Test here: https://www.ssllabs.com/ssltest/
Self-signed SHA-256 Certificate

makecert.exe -r -pe -n "CN=%1" -b 01/01/2015 -e 01/01/2020 -eku 1.3.6.1.5.5.7.3.1 -sky exchange -a sha256 -len 2048 -ss my -sr localMachine -sp "Microsoft Enhanced RSA and AES Cryptographic Provider" -sy 24

(See CreateSelfSignedSHA256SslCert.bat under https://github.com/rhurlbut/HttpsTools)

Installs self-signed SHA256 certificate into the My/LocalMachine store

Useful for local dev / testing websites
Security Headers

Added layer of security sent with HTTP/S Response Headers

Others: Content-Security-Policy, X-Content-Type-Options, X-Frame-Options and X-XSS-Protection.

HTTPS only: Strict-Transport-Security and Public-Key-Pin

Test: https://securityheaders.io/
This talk is based on a talk given by Sun Hwan Kim and Julien Sobrier (both work at Salesforce) at AppSecCali 2017 – slides used by permission.

Acronyms, etc.

We’re talking about HTTP response headers
HPKP = HTTP Public Key Pinning (report or block)
HSTS = HTTP Strict Transport Security
CSP = Content Security Policy (report or block)
Secure cookie = cookie with secure flag
HTTPS/SSL = TLS (for this talk)
Agenda

Divide and conquer
  Domain separation

TLS version(s)

Enforce HTTPS on your domains
  HSTS and secure Cookie
  Public Key Pinning

CSP
  CSP reports

Enforce HTTPS on 3rd party domains
  CSP enforced
Multiple domains

Divide and conquer

Use multiple domains or subdomains to divide a big problem into smaller problems

- API vs Browser
- Static vs live content
- CDN as a separate domain/subdomain
- Subdomain per customer

But letting users bring their own CNAME makes things more complicated.
Why sub-domains?

Divide and conquer

HSTS/HPKP headers apply to a domain (and its subdomains optionally)
TLS can be managed per subdomain (SNI)
Isolate customer/systems for later update to HTTPS
  … while taking care of mixed-content issue
    Upgrade static content servers first (server)
Cached content/CDN: headers may be cached or the same for all users (no authentication)
TLS version

What version of TLS will you support?
… the latest of course! (1.2)
Except if you need to support:
  - Java 1.6 and earlier (API)
  - Other libraries that only support SSL 3.0 and TLS 1.0
  - Internet Explorer 10 and below out of the box
TLS 1.0 has to be disabled on server side to prevent downgrade attack
  - PCI compliance helps (deadline June 2016 then June 2018)
Certificate

TLS version(s)

What Certificate authority will you choose?

Which hasn’t suffered a big breach previously (DigiNotar, Comodo)

Which has not been backdating certificates (WoSign)

Which has not been silently sold to the Chinese in secret (StartCom)
HTTP Headers

Enforce HTTPS on your domains

HSTS/HPKP/CSP are great if you browser support them

<table>
<thead>
<tr>
<th></th>
<th>IE6-10</th>
<th>IE11-Edge</th>
<th>Chrome</th>
<th>Firefox</th>
<th>Safari</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSTS</td>
<td>2015</td>
<td></td>
<td>4</td>
<td>4</td>
<td>Maverick 2013</td>
</tr>
<tr>
<td>HPKP</td>
<td></td>
<td>46</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSP 1.1/2.0</td>
<td></td>
<td>47/47</td>
<td>49/49</td>
<td>9.1/10</td>
<td></td>
</tr>
</tbody>
</table>

- **Supported by the browser**
- **Not supported by the browser**
DEMO – SSL Strip

Moxie Marlinspike: More Tricks for Defeating SSL

https://youtu.be/5dhSN9aEljg?t=10m
HSTS

Enforce HTTPS on your domains

Force HTTPS Only with HSTS

HSTS tells the browsers to connect over HTTPS only
   Optionally includes all sub-domains
   “cache” duration of header

**Strict-Transport-Security:** max-age=31536000; includeSubDomains

First time connection

Does not solve the first-connection issue

Many “first time”: HSTS cache expired (occasional login), new subdomain
   Use includeSubDomains and make a call to the top domain
Limitations of HSTS

Enforce HTTPS on your domains

HSTS preload list
   Only possible on top domain with includeSubDomains

Don’t forget about IE
   Redirect HTTP to HTTPS
   Use Secure cookies

… but credentials might still be sent directly over HTTP
   Use HTTPS-only domain for authenticated traffic (no port 80)
Separate HTTPS login

Enforce HTTPS on your domains

Use HTTPS-only domain for authenticated traffic

Should include login page

If you want to do fancy (AJAX) login forms from HTTP to HTTPS sub-domain, you need to enable CORS

*Might* be a good idea in theory; not seen widely adopted.
HPKP: Public Key Pinning

Public Key Pinning

Public Key Pinning: indicate to the browser what certificate to expect

*Public-Key-Pins* and *Public-Key-Pins-Report-Only* headers

What certificate or key to pin:

- Leaf certificate: best for security but changes often, may change per host
- Public key: great if you rotate leaf certificate often but use the same key to generate them
- Intermediate CA: typically changes every few years, trust put in your CA to never issue certificates to the wrong people

*Public-Key-Pins-Report-Only:*

```
pin-sha256="9n0izTnSRF+W4W4JTq51avSXkWhQB8duS2bxVLfzXsY="
pin-sha256="6m4uJ26w5zoo/DLDmYNWGIdWpZ8/GSCP6SBri8Euw0=";
max-age=604800; report-uri="https://otherdomain.com/path";
```
HPKP Workflow

Public Key Pinning

Browser

Certificate

* ≠

Violation report

HTTP Request

HTTP Response with HPKP pin(s)

TLS negotiation

Certificate

HTTP server/application

Public end-point

TLS termination
HPKP in practice

Public Key Pinning

Consider using HPKP in reporting mode only

Several issues found:

- CDN use a subdomain of xyz.com but generate the certificate themselves.
- Hard to use includeSubDomains
- Public proxies/anonymizers: do not rewrite response headers but use different CA
  example: https://nodeunblock.herokuapp.com/
- Chrome 38/39 uses expired pins until browser restart
  It looks like some other browser has the same behavior, expired pin is used one time for new access
- Internal proxy: end up setting the HPKP header for custom CNAMEs with customer certificates
Scott Helme on Pinning

Guidance on setting up HPKP (9/15/2015)
https://scotthelme.co.uk/guidance-on-setting-up-hpkp/

I’m giving up on HPKP (8/24/2017)
https://scotthelme.co.uk/im-giving-up-on-hpkp/

Your mileage may vary …
Dealing with 3rd party assets
Mixed content

You want to make sure that you don’t have mixed-content (HTTPS loading HTTP assets)

- From your own site (hardcoded references) or 3rd party
- Browsers don’t (yet) block mixed-content for images (img or CSS), audio, video and object (“passive” content)

In a heavily customized sites, how do you check that all assets are always loaded over HTTPS?
Customized content

Content Security Policy to the rescue!

A way of whitelisting allowed sources

HTML Meta elements can be used in place/addition of HTTP header

CSP applies to a URL.
Customized content

CSP has many directives addressing different issues

Type of content: font-src, img-src, child-src, script-src, connect-src, etc.

Origin allowed: ‘self’, https: (Any HTTPS URL), data: (inline data), none, etc.

2 CSP headers

Content-Security-Policy-Report-Only : Report only, does not work with all directives
Content-Security-Policy : Enforce and report (when possible)
Example of CSP headers, describe where assets can be accessed from

Content-Security-Policy-Report-Only: # Check policy but do not enforce
  default-src 'self' https://*.site.com; # Default for all assets: same origin or site.com
  img-src https://*.cdn.com; # But images should be download for cdn.com
  report-uri /my-url # Report any violation to this URL
CSP for HTTPS

Example of CSP policy to check that all requests are done over https

Content-Security-Policy-Report-Only:
  default-src https;                     Any HTTPS
  script-src https: 'unsafe-inline' 'unsafe-eval'; Any HTTPS,
  inline, eval()                      Any
  style-src https: 'unsafe-inline';          Any
  HTTPS, inline                     Any
  img-src https: data::*              Any
  HTTPS, inline                     Any
  font-src https: data::*            Any
  HTTPS, inline                     Any
  connect-src: https::*             Any
  HTTPS (AJAX)                     Any
CSP format – Violation report

Example of violation report (Firefox)

{"csp-report":{


"document-uri": "https://intertnal.my.salesforce.com/00XXXXXXXXXX"


"referrer": "https://intertnal.my.salesforce.com/00XXXXXXXXXX",

"violated-directive": "img-src https: data:"

}}
How to use CSP?

Learned from report-only mode

Inline scripts used widely

Customers using external scripts from Facebook, Skype, AWS, Twitter, etc.

Lots of assets loaded over http

Even Report-Only Mode can cause an issue

100+ reports were generated in one page, and slowed down performance
How to use CSP?

Different Features required different Policy
If there are multiple CSP headers, each one is applied separately
Default CSP for every request, additional CSP headers where needed
Enforce HTTPS on 3\textsuperscript{rd} Party Domains

CSP Report Only Mode First – Analyzing reports periodically

Make necessary adjustments – whitelist several sources, fine-tune the policy

Make sure automatic redirection from http to https do not cause any service interruption

Staged rollout

Based on report analysis, enforce https selectively

Give Customers option to disable CSP temporarily
Upgrade-insecure-requests & blocked-all-mixed-content

**upgrade-insecure-requests**: upgrade all HTTP requests to HTTPS
Can be dangerous with 3\textsuperscript{rd}-party domains: no HTTPS or invalid certificate (EC2…)
Good to use if all assets are on your own servers

**block-all-mixed-content**: block mixed content for passive tags too.

Obviously these directives must be enforced (no report-only). First directive has no report, fail silently. Second directives has reports.
General Advice for CSP

**CSP is constantly changing**
- Stay up-to-date with new CSP directive
- Introduce new CSP directive in report only mode first

**Behaviors might vary depending on browsers**
- Some browsers did not send cookies required for authentication
- Not all directives are supported by browsers
HTTP/2

Only available for use by HTTPS

Performance improvements in all requests (binary vs textual, fully multiplexed, header compression, etc.)

Available IIS 10 (seamless)
HTTP/2

Networking protocol for low-latency transport of content over the web. Originally started out from the SPDY protocol, now standardized as HTTP version 2.

See also support for the SPDY protocol, precursor of HTTP2.

1. Partial support in IE11 refers to being limited to Windows 10.
2. Only supports HTTP2 over TLS (https)
3. Partial support in Safari refers to being limited to OSX 10.11+
4. Only supports HTTP2 if servers support protocol negotiation via ALPN
Key takeaways

Staged rollout
    Static content first

TLS set up
    Decide what clients you want to support

HSTS + HPKP
    Can you includeSubDomains?
    Start with small max-age durations
    (consider not using HPKP because of problems)

CSP
    Check for http requests
    Ensure not too many violations are generated
    See if upgrade-insecure-requests is an option for you

HTTP/2
    Check into setting up on server for performance improvements of HTTPS
Resources - Books

Bulletproof SSL and TLS by Ivan Ristic
The Tangled Web: A Guide to Securing Modern Web Applications by Michal Zalewski
Iron-Clad Java: Building Secure Web Applications by Jim Manico and August Detlefsen
Secure Your Node.JS Web Applications: Keep Attackers Out and Users Happy by Karl Duuna
Resources - Tools

HSTS Preload
https://hstspreload.org/

Qualys SSL Labs – SSL Server Test
https://www.ssllabs.com/ssltest/

Security Headers
https://securityheaders.io/

Automated Security Analyzer for ASP.NET Websites
https://asafaweb.com
Questions?

Contacts
Web Site: https://roberthurlbut.com
Twitter: @RobertHurlbut, @AppSecPodcast

© 2017 Robert Hurlbut