Don’t get pwned in 2018 webapp lockdown checklist

Luka Kladaric
Sekura consulting
luka@sekura.io
@kll

What this talk is about

Securing users, their browsers and computers.

What this talk *isn't* about

Securing your servers or the data on them.

Quick poll

Quick poll

- Who here runs a website?

Quick poll

- Who here runs a website?
- Does it support HTTPS?

Quick poll

- Who here runs a website?
- Does it support HTTPS?
- Does it enforce HTTPS?

Quick poll

- Who here runs a website?
- Does it support HTTPS?
- Does it enforce HTTPS?
- Does it have HSTS headers set?

HTTPS / SSL / TLS

A method of encrypting traffic to and from a web server using a server-side certificate.

If the certificate is signed by a trusted party it also gives us a degree of confidence that the server we're communicating with is actually authorized to speak on behalf of the website we're trying to visit.

HTTPS (2)

Once you've established the other side possesses a trusted certificate, the chance you're being **MitM**[1]-ed drops by orders of magnitude.

Certificates can also be self-signed, or otherwise signed by an untrusted party. These are mostly worthless.

---

[1] MitM - Man in the Middle attack

Forcing HTTPS and pitfalls

Forcing HTTPS means redirecting any plaintext traffic to HTTPS.

It’s good in that it redirects people to the secure site, and you should definitely do that.

But it’s bad in many ways.

Forcing HTTPS

1. Obscures/hides the fact that certain things still point to your plaintext address.
    - The worst scenario for this is if you have a login form or oAuth callback pointed at http. Users are successfully logged in, but the credentials are transmitted unencrypted first, before getting redirected to https.

Forcing HTTPS

1. Takes people to the right place which means they will never update their bookmarks.

Forcing HTTPS

1. Doesn’t usually work for non-human visitors - mobile apps and other clients, which generally construct each request from scratch rather than pick a link from the page they were served.
    - So, if it works at all it just duplicates traffic where every single request is initiated over HTTP first and then bounced to HTTPS.

Forcing HTTPS

1. Breaks POST requests which cannot be redirected.

Forcing HTTPS

1. The redirect itself is not secure.
    - An MitM attack could happen at this point, with the user continuing to browse the http site while the attacker proxies requests back and forth for the user over https with neither side being aware that anything is off.

Enter HSTS

The magic bullet?

HSTS

A response header that says “don’t even attempt to contact me over plaintext HTTP for X seconds”.

The header can optionally apply to all subdomains, too.

HSTS preload

And if it applies to subdomains and has a long enough expiry set, it can be flagged for inclusion in the preloaded list of domains that have HSTS applied on them.

All major browsers ship with a list of domains that should never be accessed over plaintext.

HSTS downsides

If you can’t force HTTPS, you can’t enable HSTS.

HSTS downsides

If you have subdomains which don’t support HTTPS, or still need to receive plaintext traffic, you can’t enable “includeSubdomains”.

HSTS downsides

If you can’t include subdomains, you can’t enable preload. Which means you still have that first request for each user that is vulnerable.

Secure cookies

Cookies are used for various tracking and storage purposes, but the most critical use is to keep logged in state.

To leak these is to let other people impersonate a legitimate user in their interaction with your website / application.

Secure cookies

Absolutely make sure that all the cookies you are sending to your users have the “Secure” flag set on them, so that even if a plaintext HTTP request ever happens, they are not transmitted.

XSS

XSS is bad.

Don’t let XSS happen to you.

The end. j/k

CSP

Content Security Policy is an HTTP response header that reduces the risk of XSS in modern browsers by declaring which inline or external resources are allowed.

It lets you define which hosts the browser should whitelist for Javascript, CSS and image resources, AJAX requests, fonts etc, while blocking all others.

CSP

It also lets you block all inline scripts except the ones explicitly whitelisted by several methods.

The combination of these two things makes XSS unusable on your website, despite any backend code flaws.

CSP - nonces

A nonce can be used to whitelist inline scripts which contain no dynamic content. A single nonce can be used for all static inline scripts.

```
Content-Security-Policy: default-src 'self'; script-src 'nonce-4AEemGb0xJptoIGFP3Nd'
<script type="text/javascript" nonce="4AEemGb0xJptoIGFP3Nd">
```

CSP - hashes

For scripts that are dynamic but trusted (no user input), we can use hashes.

```
content-security-policy: script-src 'sha256-
cLuU6nVzrYJlo7rUa6TMmz3nylPFrPQrEUpOHllb5ic='
```

Crudest and simplest way to generate the necessary hashes is to let the browser yell at you (through the developer console) that you’re missing that specific hash.

CSP - final

```
Content-Security-Policy:
    default-src 'none';
    script-src 'self' 'nonce-0gscAiOe9cvhDVnsp2jy'
            www.google-analytics.com
            cdnjs.cloudflare.com;
    object-src 'none';
    style-src 'self' 'unsafe-inline' cdnjs.cloudflare.com;
    img-src 'self' www.google-analytics.com
        stats.g.doubleclick.net;
    media-src 'none';
    font-src 'self' fonts.gstatic.com;
    connect-src 'self';
    base-uri 'self';
    child-src www.google.com;
    form-action 'self' accounts.google.com www.paypal.com;
    frame-ancestors 'none';
    upgrade-insecure-requests;
    report-uri https://yoursite.report-uri.com/r/d/csp/enforce
```

Questions?

Luka Kladaric
luka@sekura.io
@kll

Thank you!

Luka Kladaric
luka@sekura.io
@kll