

HTTP Strict Transport Security

In this lesson, we'll study the HTTP strict transport security.

We'll cover the following ^

- Introduction
- HSTS
 - HTTPS

Introduction

As we've seen, servers can send HTTP headers to provide the client with additional metadata around the response. Besides sending the content that the client requested, servers are then allowed to specify how a particular resource should be read, cached or secured.

There's a large spectrum of security-related headers that we should understand, as they have been implemented by browsers in order to make it harder for attackers to take advantage of vulnerabilities. The next paragraphs try to summarize each of them by explaining how they're used, what kind of attacks they prevent, and a bit of history behind each header.

HSTS

Since late 2012, HTTPS-everywhere believers have found it easier to force a client to always use the secure version of the HTTP protocol, thanks to the *HTTP Strict Transport Security*. A simple `Strict-Transport-Security: max-age=3600` will tell the browser that for the next hour (3600 seconds) it should not interact with the applications with insecure protocols.

When a user tries to access an application secured by HSTS through HTTP, the browser will simply refuse to go ahead, automatically converting `http://` URLs to `https://`.

You can test this locally with the code at github.com/odino/wasec/tree/master/hsts.

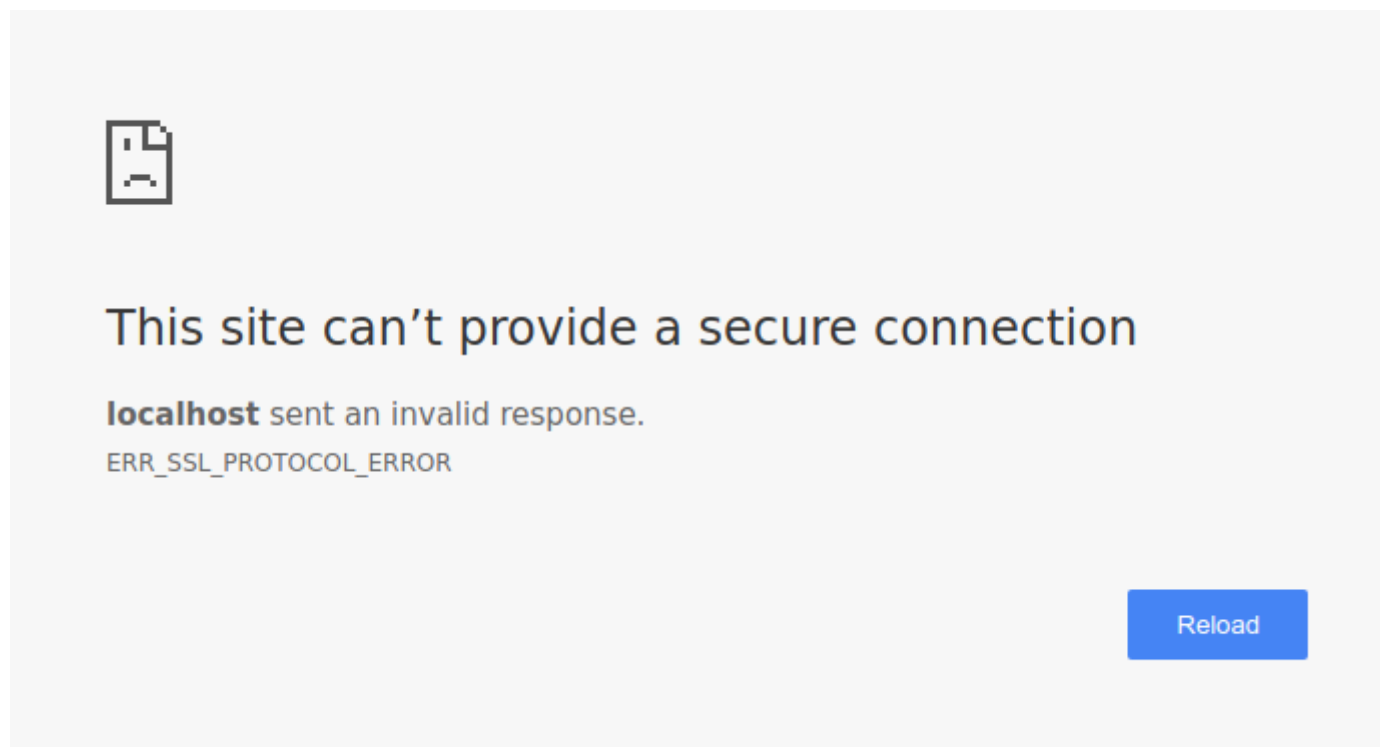
You will need to follow the instructions in the README (that involves installing a

You will need to follow the instructions in the README (that involves installing a trusted SSL certificate for `localhost` on your machine, through the `mkcert` tool, and then try opening `https://localhost:7889`.

HTTPS

There are two servers in this example, an HTTPS one listening on `7889`, and an HTTP one on port `7888`. When you access the HTTPS server, it will always redirect you to the HTTP version, which will work since there is no HSTS policy on the HTTPS server.

If you instead add the `hsts=on` parameter in your URL, the browser will forcefully convert the link in the redirect to its `https://` version. Since the server at `7888` is http-only, you will see a page that looks more or less like this:



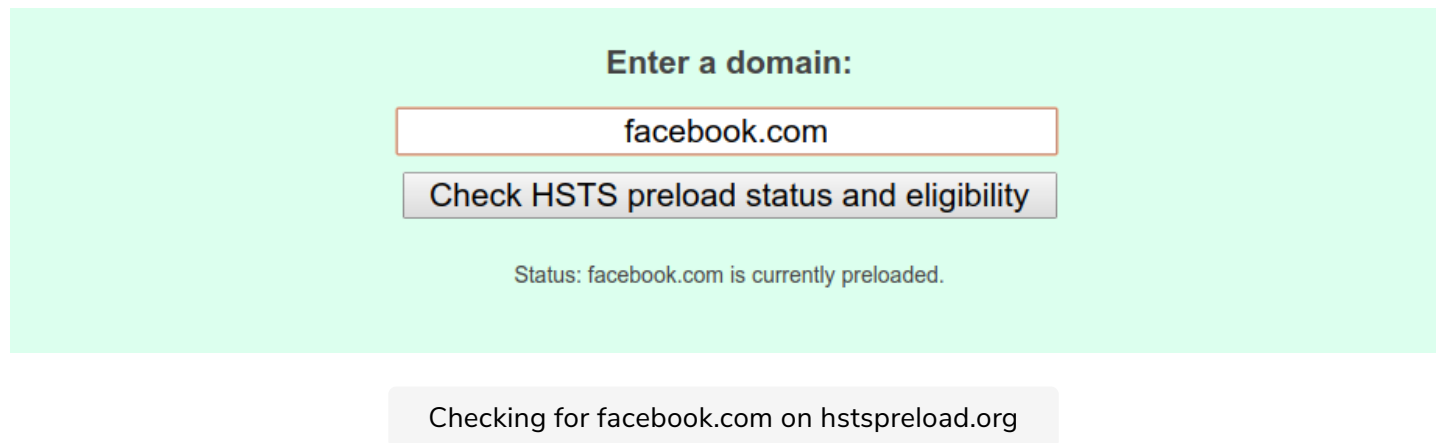
Chrome's response when redirected to http server with hsts parameter on

You might be wondering what happens the first time a user visits your website, as there is no HSTS policy defined beforehand. Attackers could potentially trick the user to the `http://` version of your website and perpetrate their attack there, so there's still room for problems. That's a valid concern, as HSTS is a *trust on first use* mechanism. It tries to make sure that once you've visited a website, the browser knows that subsequent interactions must use HTTPS.

A way around this shortcoming would be to maintain a huge database of websites

that enforce HSTS, something that Chrome does through hstspreload.org. You must

set your policy then visit the website to check whether it's eligible to be added to the database. For example, we can see Facebook made the list.



The screenshot shows the hstspreload.org interface. At the top, it says "Enter a domain:". Below this is a text input field containing "facebook.com". Under the input field is a button labeled "Check HSTS preload status and eligibility". Below the button, the status is displayed: "Status: facebook.com is currently preloaded." Below the main interface, there is a separate box that says "Checking for facebook.com on hstspreload.org".

By submitting your website to this list, you can tell browsers in advance that your site uses HSTS so that even the first interaction between clients and your server will be over a secure channel. This comes at a cost though, you really need to commit to HSTS. It's not an easy task for browser vendors to remove your website from the list.

Be aware that inclusion in the preload list cannot easily be undone.

Domains can be removed, but it takes months for a change to reach users with a Chrome update and we cannot make guarantees about other browsers. Don't request inclusion unless you're sure that you can support HTTPS for your entire site and all its subdomains for the long term.

hstspreload.org

This happens because the vendor cannot guarantee that all users will be on the latest version of their browser, with your site removed from the list. Think carefully, and decide based on your degree of confidence in HSTS and your ability to support it in the long run.

In the next lesson, we'll study HTTP public key pinning.

