The State of HSTS Deployment: A Survey and Common Pitfalls

Lucas Garron
Stanford University
lgarron@cs.stanford.edu

Andrew Bortz
Dropbox
abortz@dropbox.com

Dan Boneh
Stanford University
dabo@cs.stanford.edu

ABSTRACT

HSTS (HTTP Strict Transport Security) has gained significant browser and server adoption since reaching IETF proposed status. However, there are several important deployment challenges. A scan of top websites reveals that many HSTS sites have not properly configured the HSTS header, which still leaves them open to some attacks HSTS is meant to solve. We survey the current state of deployment and describe common mistakes and difficulties with HSTS configuration. We conclude with approaches for properly deploying HSTS as effectively as possible.

1. NOTE FROM LUCAS (2014-07-22)

Although this paper gives a good picture of all the issues with HSTS deployment I was aware of at the time, this paper has some flaws, partially because:

- it's the first "academic"-style CS paper I wrote myself from scratch, and
- I had about one week to gather all the data and write in order to meet the WWW2014 deadline (2013-10-09).

Unfortunately, the paper didn't get accepted to WWW2014, so I haven't made the time to clean it up and/or measure progress.

Also, a disclaimer: abortz and Dan helped me pursue this project, but didn't review this draft extensively (we were going to do that if it got accepted). Therefore, blame any mistakes on me.

For the latest info, visit:

- `https://github.com/lgarron/HSTS`
- `https://garron.net/crypto/hsts`

2. INTRODUCTION

Although HTTPS can be fairly effective defense against against passive and active attacks, many sites still accept requests over plain HTTP. The HSTS (HTTP Strict Transport Security) browser mechanism helps sites ensure security by avoiding plain HTTP. HSTS is now implemented in three of five major browsers and has seen significant uptake by security-conscious sites like Google, Paypal, Twitter, Stripe, and iCloud.

The core benefit of HSTS is that it allows any host response to specify a `Strict-Transport-Security` header. This creates an entry in the browser that causes all future resource loads from the domain to be sent over HTTPS. Such an entry mitigates a wide variety of attacks, such as eavesdropping, injection/redirection, HTTPS stripping, and DNS spoofing.

However, consider the situation where a user types `example.com` into their browser and is redirected to the canonical domain `https://www.example.com`. If the site only sends an HSTS header over `https://www.example.com`, future requests without `www` will not be secured (e.g. when the user types in the same URL or clicks a link to the unsecured domain). Thus, part of the domain is still suffers from the vulnerabilities HSTS was meant to address.

There are several countermeasures to improve the strength and practicality of HSTS: HSTS provides the `includeSubDomains` directive to protect subdomains of responses. In addition, browsers supporting HSTS ship with a list of preloaded HSTS entries that sites may request to be included in. There are also active efforts for related mechanisms, such as TACK[8] and other pinning mechanisms. However, these do not intrinsically guard against issues like our redirection example. Related domains and URL canonicalization can easily bypass the intention of HSTS, and deployment for a real-world site can benefit from consideration of browser details like HSTS implementation details and the cache mechanism.

Although some of these issues have been noted as early as 2010[15], they are not discussed in popular descriptions of HSTS, and many sites still appear to be unaware of them.

2.1 An Overview of HSTS

HSTS (HTTP Strict Transport Security) is a browser security mechanism introduced as ForceHTTPS in 2008 by Jackson and Barth[16] and accepted for the IETF Standards Track in 2012[17]. The introduction to RFC 6787 provides

a good overview and history of HSTS.

HSTS is an HTTP header (`Strict-Transport-Security`) that a host may send to browsers to request heightened security for the current domain. In particular, any conforming browser header that has received an HSTS header from `http://example.com` will automatically load future requests to the site from `https://example.com` instead. Thus, HSTS generally touted as a mechanism to "transform insecure URI references... into secure URI references". However, it also enforces strict security in related mechanisms, such as preventing mixed content and "click-through" certificate overrides[17].

There are three semantically distinct ways to send the HSTS header; these are described clearly in RFC 6797[17]:

> *The HSTS header field below stipulates that the HSTS Policy is to remain in effect for one year... and the policy applies only to the domain of the HSTS Host issuing it:*
>
> `Strict-Transport-Security: max-age=31536000`

> *The HSTS header field below stipulates that the HSTS Policy is to remain in effect for approximately six months and that the policy applies to the domain of the issuing HSTS Host and all of its subdomains:*
>
> `Strict-Transport-Security: max-age=15768000 ; includeSubDomains`

> *The HSTS header field below indicates that the UA must delete the entire HSTS Policy associated with the HSTS Host that sent the header field:*
>
> `Strict-Transport-Security: max-age=0`

The specification also states that the HSTS must only be sent and accepted over HTTPS, and that the `includeSubDomains` directive is ignored for domains when `max-age=0`.

In addition, all browsers currently supporting HSTS also ship with a list of known HSTS hosts, called the "pre-loaded" / "preload" list. HSTS is enforced for these sites by default.

3. BROWSER CONSIDERATIONS

3.1 HSTS Browser Support

As of September 29, 2013, HSTS is supported in the desktop versions of Chrome, Firefox, and Opera. It is not supported in Internet Explorer or Safari.[12]

In addition, the mobile versions of Chrome (iOS and Android) and Firefox (Android) support HSTS like their desktop counterparts[13].

3.2 Chromium Preload List

The Chromium project maintains a list of sites in one of its source files[9]. A site owner can request their site to be included in this list to enforce a combination of several browser-side requirements including:

- `force-https` - enables HTTPS for a site by default (without expiration).
- `includeSubDomains` - same as HSTS.
- `pins` - specific pinned certificates.

Google Chrome and Chromium employ this list directly, enforcing the specified security measures.

Chrome considers HSTS to be part of the cache. The "Empty the cache" checkbox under "Clear Browsing data" wipes stored HSTS values.

3.3 Opera

Opera shares the WebKit-derived Blink engine with Chromium, so it exhibits the same HSTS behavior as Chrome.[1]

3.4 Firefox Preload List

Mozilla "seeds" the Firefox preload list by starting with the domains in the Chromium preload list, but performs its own filtering to keep the list fresh and observe the latest host settings. An automated build bot[2] regularly runs[10] a script[3] that processes the Chromium list and commits a resulting preload list[6] (along with the list of errors[5]) to the `mozilla-central` repository.

A site is included in the Firefox preload list if the following hold:

- It is in the Chromium list (with `force-https`).
- It sends an HSTS header.
- The `max-age` sent is at least 10886400 (18 weeks).

Firefox enforces HSTS for all sites in its preload list, which is hard-coded and compiled into the main binary.

The Firefox preload list also includes a boolean flag for `includeSubDomains`. This flag is set to true if the script recorded `includeSubDomains` is in the site's HSTS header (regardless of the Chromium list setting).

The Firefox preload list also includes an expiration time: each script run sets a timeout of 18 weeks from the time it is run, after which Firefox will ignore it (i.e. sites can use HSTS normally, but the browser will no longer use the preload list to enforce HTTPS on first load for any sites)[1]. Since the release cycle shortens the time HSTS is useful in the browser to significantly less than 18 weeks, Mozilla may addressed this in the future, and any sites considering HSTS

[1]Opera originally implemented support in Presto engine[11], but Opera is closed-source, and does not provide explicit documentation about HSTS support since the Blink transition. It behaves like Chrome for preload sites according to the network console.

should check current Mozilla policies before selecting a `max-age`.

Firefox considers HSTS to be part of "Site Preferences" in its "Clear Recent History" feature.

4. CURRENT DEPLOYMENT

4.1 Alexa Top 100000 Sites

We automatically surveyed the top 100000 sites in the Alexa global rankings[14] by making `GET` requests to four root URLs each:

- `http://example.com`
- `http://www.example.com`
- `https://example.com`
- `https://www.example.com`

(Where "`example.com`" is replaced by the domain.)

Although sites may use complicated HSTS mechanisms, we assume that front pages are a sufficient proxy of HSTS behavior. If sites are likely to go for maximum protection and/or simplest configuration, this is a fair assumption.

We recorded:

- which sites responded over HTTPS at all, either at `https://example.com` or `https://www.example.com`,
- how many sites used HSTS with `max-age`> 0 (and thus had any protection),
- how may HSTS sites used `includeSubDomains`, and
- how many sites used HSTS with a `max-age` of 0.

We only considered HSTS headers that were well-formed and sent over HTTPS on either `https://example.com` or `https://www.example.com`. Several technical and financial examples among the top few HSTS sites stand out, but we have not attempted to categorize which kinds of sites use HSTS. The tallies for top sites are summarized in table 1 and figure 1.

# top sites	HTTPS	HSTS	incl. sub.	`max-age=0`
10	8	1	0	1
100	76	3	0	2
1000	629	11	3	4
10000	5402	56	11	10
100000	46943	277	66	25

Table 1: HTTPS adoption vs. adoption of HSTS, HSTS with `includeSubDomains`, and HSTS explicitly disabled.

These numbers were found by using a `python` script using the `scrapy`[7] library with a current Google Chrome user agent string[2]. All results are from early October 2013. Although we accepted any status code, we only considered initial responses (i.e. no redirects), since a primary goal is to secure the initial landing.

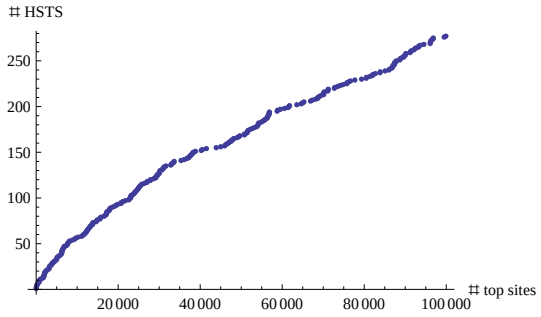

Figure 1: HSTS adoption among the Alexa top 100000 sites.

4.2 The Alexa + Chromium HSTS Sites

For the rest of this paper, we will discuss sites from two sources:

- Alexa top 100000 sites who send a valid HSTS header over HTTPS with `max-age`> 0.
- Sites from the Chromium HSTS Preload list with `force-https` set to `true`.

4.3 Common `max-age` Settings

By far the most common `max-age` setting is a static value of 31536000 seconds, which is roughly one year. This may be because the implementation examples on Wikipedia use this value[4], and it is the `max-age` of two (out of three) examples in RFC 6797[17]. The majority of remaining sites are testing HSTS with very low values, ranging from seconds to days, although `max-age` values are as high as 20 years (`twitter.com`).

As an interesting point, nearly every site serving HSTS on both `https://example.com` and `https://www.example.com` sends the same `max-age` on both domains.

The `max-age` values group naturally into several buckets, which are displayed in figure 2 (including sites with `max-age` 0 for comparison).

5. COMMON DEPLOYMENT PATTERNS

The minimum use of HSTS is to secure a single canonical domain; either `example.com` or `www.example.com`. However, it is usually desirable to do some of the following:

- Secure all visited (sub)domains (e.g. protect any resource on second load).
- Secure `example.com` and `www.example.com`.

[2]`Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.76 Safari/537.36`

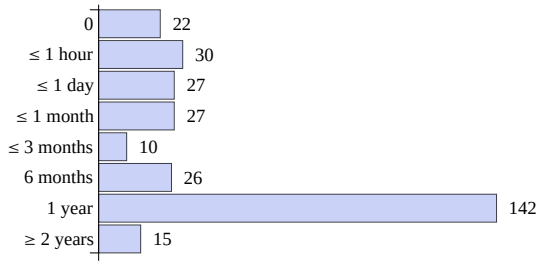

Figure 2: HSTS `max-age` settings of sites in the Alexa + Chromium HSTS Sites. (Each site is placed in the highest valid bucket.)

- Secure all (sub)domains of `example.com`.

Most HSTS sites redirect straight to a canonical URL, adding `HSTS` and modifying the presence of `www` consistently. However, there is wide variation in redirect schemes and HSTS header. We considered all of the Alexa + Chromium HSTS sites which only redirected to common subdomains of a main domain. There were 409 of these, including some Chromium sites that do not send HSTS headers.

5.1 Redirect to `https://example.com`

A plurality of 78 sites redirect straight to a final destination of `https://example.com`, where they serve an HSTS header, including 31 who are subdomains themselves.

Of these, 19 serve an HSTS header with `includeSubDomains` over `https://example.com`, automatically protecting all subdomains.

27 sites did not send an HSTS header over `https://www.example.com`, and left it unprotected in all cases.

All but 1 of these were straight redirects.

5.2 Redirect to `https://www.example.com`

37 sites redirected straight to `https://www.example.com`. Although 10 of them set `includeSubDomains` over `https://www.example.com`, only 9 secured both `https://example.com` and `https://www.example.com` at their root URLs – and only 2 did both.

34 additional sites directed straight to `https://www.example.com` but did not allow a valid connection over `https://example.com`. 18 of these sent `includeSubDomains`.

All but 4 of these were straight redirects.

5.3 Redirect to HTTPS

24 sites did not add or remove `www`. All sites sent an HSTS header over both `https://example.com` and `https://www.example.com`. 6 sent `includeSubDomains` over both, and the rest sent it over neither.

Most of these sent HSTS over both `https://example.com` and `https://www.example.com`, with roughly 1/4 of each using `includeSubDomains`.

All but 4 of these were straight redirects.

5.4 Canonical Redirect for HTTP Only

23 sites served either `https://example.com` or `https://www.example.com` but consistently redirected plain HTTP requests to `https://example.com`. Another 23 similarly sent plain HTTP to `https://www.example.com` instead, although 13 of them redirected to `https://example.com` first.

5.5 Miscellanous

The remaining sites usually matched the patterns above, with particular anomalies.

6. COMMON MISTAKES

6.1 Permanently Insecure Redirects

A common pattern is to redirect to `https://www.example.com` when the user types `example.com` into their browser. Assuming the parent domain is not on the preload list, the initial redirect is necessarily insecure on a fresh browser. If the site does not set an HSTS header for `https://example.com`, then this insecure redirect occurs *every time* the user types `example.com` into their browser.

This is mitigated by the browser cache retains an entry for `example.com`. However, it is undesirable to rely on the cache for such a critical security guarantee, and the protection does not apply for any other pages/resources on `http://example.com` except the specific cached URL.

6.2 Unprotected Subdomains

Unless a site sets `includeSubDomains` on `https://example.com`, neither of `https://example.com` or `https://www.example.com` automatically set HSTS for each other. Assuming only one of these is visited by a normal user, the other will be unprotected.

6.3 Accidental HSTS Expiration

It is reasonable to send an HSTS header at `http://example.com` along with a $30x$ redirect to `https://www.example.com`. However, browsers may cache 301 redirects aggressively by default. The browser will send `http://example.com` visits directly to `https://www.example.com`, without making a request to `https://example.com`; thus, the browser will not see another HSTS header for `https://example.com`, and HSTS will eventually expire once the `max-age` is reached. If the `max-age` is updated on the server, the browser will also not see the updated value.

Again, this reduces the security guarantee for HSTS to the cache for known URLs, and will eventually fail to protect other resources.

6.4 Clerical Mistakes

30% of HSTS sites in our survey also send an HSTS header over `http://example.com` or `http://www.example.com`. While an RFC-conforming browser is required to ignore this, RFC 6797[17] also states that:

> *An HSTS Host MUST NOT include the STS header field in HTTP responses conveyed over non-secure transport.*

Sending the header over plain HTTP suggests an unfamiliarity with HSTS, apathy towards the specification, or inadequate control over redirect configuration.

In our scans, we also encountered various occasional mistakes, such as using a comma instead of a semicolon, sending an integer value without `max-age=`, including the `must-revalidate` directive (thus resembling a cache-control header), or removing the hyphen from `max-age`. A few sites, including `paypal.com`, also send two HSTS headers in the same request.

7. ISSUES WITH IMPROVING HSTS CONFIGURATION

7.1 Canonical Subdomain

If the canonical domain for a site is `https://www.example.com`, the user may rarely load a page/resource from `https://example.com`. This makes it more difficult to secure a fresh visit to `https://example.com`, should the user visit it directly later. It also fails to protect any other subdomains.

7.2 Implications of `includeSubDomains`

In practice, using `includeSubDomains` may mot be straightforward at the top-level (`https://example.com`) of an existing site because there is at least one subdomain that may break over HSTS. Possibilities for this include:

- Insecure resources will fail to load and break pages that worked over HTTPS without HSTS.
- Such a large change is usually accompanied by increasing the scope of secure redirects ("*if* we're going to do it, we may as well do it all at once"), which may break certain clients that rely on plain HTTP.
- The server for a subdomain is not configured to use HTTPS.
- It may be impractical/expensive/risky to furnish every subdomain server with a valid CA-signed certificate.
- The browser will prevent the user from using self-signed certificates for any subdomains.
- The browser will enforce HTTPS for all subdomains even if there are other security measures in place (such as local testing or gating for a corporate network) that would otherwise allow the continued use of plain HTTP.

While it is worthwhile to address these issues and move towards more robust solutions that can operate under HSTS with `includeSubDomains`, any site should be aware of these issues before considering HSTS a free panacea.

8. BEST PRACTICE

With regard to HSTS, a site should operate under the threat model that an attacker can run a man-in-the-middle attack over plain HTTP for any (sub)domain of the site, e.g. by exposing a victim to spoofed DNS entries. However, the attacker cannot intercept or modify valid HTTPS traffic over any of these subdomains.

Thus, it is desirable to protect as many subdomains as possible, as early as possible, by registering HSTS for them in the browser.

8.1 Ideal Practice

The strongest protection is to redirect every resource to HTTPS immediately and serve well-formed HSTS header:

- with every HTTPS request,
- with a long `max-age` (18 weeks minimum, 1 year recommended), and
- with `includeSubDomains`.

To be fully correct, the site must also:

- never send an HSTS header over plain HTTP.

This protects all future requests to the site or any of its subdomains. If the canonical site is `https://example.com`, any requests to the site will automatically extend the expiration date (including for subdomains).

In addition, once a site is committed to HSTS, it should:

- be included in the browser preload lists.

Any security-conscious site should strive to all of take these steps, if possible. In case this is initially impractical, we provide separate recommendations below.

8.2 Send an HSTS header whenever possible

It is advisable to send an HSTS header on any request sent from a (sub)domain only intended to be accessed over HTTPS. This ensures that HSTS is enabled on every subdomain as early as possible, without depending on other sub/superdomains.

In addition, every visit extends the expiration of HSTS.

8.3 Send the most secure settings possible

Every site should aim to ramp up its `max-age` as high as possible (at least to the popular default of 6 or 12 months).

As advocated in section 14.4 of RFC 6797[17], a site should aim to send `includeSubdomains` whenever possible. Although section 7.2 discusses issues with `includeSubDomains` for `https://example.com`, it is usually safe to send `includeSubDomains` on any particular subdomain, including `https://www.example.com`.

8.4 Redirect Immediately and Securely

Unless legacy use cases require plain HTTP support, every request should be redirected to HTTPS (using a $30x$ status) before any content is served.

Any redirect served over HTTPS should include an HSTS header, to ensure that the redirect is fully secured in the future.

A site should redirect HTTP requests to the HTTPS version the request before performing other canonicalization steps like adding/removing `www`. This ensures that HSTS will be applied to future redirects on the same domain.

In the case where all the redirect URLS and final destinations of a site are protected using `includeSubDomains` at the destination, a two-hop redirect and a separate HSTS setting for the redirect (sub)domain may seem redundant. However, the first redirect will be eliminated once HSTS is in place, and continues to protect the redirect in case `includeSubDomains` needs to be removed.

8.5 Securing `https://example.com` from a subdomain

If most pages are served from a subdomain like `https://www.example.com`, securing the entire domain requires an additional step. A simple solution is to make a request to an uncached HSTS-serving resource on `https://example.com`, such as a 1-pixel `<img>`. This is best done using a cache-control header, but simple cache-busting tricks work equally well.

This uncached-resource approach works in all current HSTS browsers. In case a site would like to minimize resource fetches, the particular HSTS resource can:

- send a short-lived `cache-control` header for the resource instead, and/or
- be configured not to redirect from `https://example.com` to `https://www.example.com`.

Such a solution is being used successfully in practice, e.g. by `https://www.icloud.com`.

8.6 Ensure that incoming visits are secure

Although a site should serve a plain HTTP request once if it is received, it is useful to a site from receive plain HTTP requests in the first place – by making sure they are likely to visit over HTTPS first.

Once a site is committed to HTTPS, it should ensure that all links on the website are updated to HTTPS (or protocol-relative URLs) and that automatically generated links (e.g. shortened URLs, "copy link" buttons, API-generated links) are generated with `https://`. This will help ensure that future incoming visits are secured immediately.

9. CONCLUSION

We described the current state of HSTS in the IETF specification and browser implementations. Based on this, we surveyed top sites and found a significant number of them send an HSTS header, but many of them do not secure as many of their (sub)domains as they should – leaving them open to many attacks that HSTS was meant to mitigate. In particular, we found that canonicalization redirects often stay insecure even if HSTS is used at the final page. We briefly discussed issues that sites should be aware of when using `includeSubDomains`, and concluded with concrete suggestions to improve HSTS deployment. These suggestions mostly center around selecting the most secure options available, and ensuring that the browser receives them for the desired domain – either during a redirect or using a resource load on the final destination.

These results remind us that HSTS, while effective, should not be treated as a black box security mechanism. Sites must be aware of HSTS details, and we hope this paper provides many of the details helpful for making the correct choices about initial and continued deployment.

We note that most of the issues stem from the fact that HSTS can only protect the current domain and not subdomains. Our survey shows that sites are likely to end up with unintentional security weaknesses given the currently available information. Thus, we hope that future work will either address related domains at the specification level, or that the issues and solutions identified in this paper are clearly available to sites considering HSTS – leading to a more secure web.

10. REFERENCES

[1] Bug 800444 - disable the hsts preload list if the list has gone 18 weeks without an update.
[2] Bug 836097 - automate updating the in-tree hsts preload list.
[3] gethstspreloadlist.js.
[4] Http strict transport security.
[5] nsstspreloadlist.errors.
[6] nsstspreloadlist.inc.
[7] Scrapy | an open source web scraping framework for python.
[8] Tack :: Trust assertions for certificate keys.
[9] transport_security_state_static.json.
[10] update_hsts_preload_list.sh.
[11] Web specifications support in opera presto 2.10.
[12] Can i use... hsts?, 2013.
[13] Http strict transport security, 2013.
[14] I. Alexa Internet. Alexa top sites.
[15] coderrr. Canonical redirect pitfalls with http strict transport security and some solutions.
[16] A. B. Collin Jackson. Forcehttps: Protecting high-security web sites from network attacks.
[17] A. B. J. Hodges, C. Jackson. Http strict transport security (hsts), November 2012.