HSTS Supports Targeted Surveillance

Paul Syverson
U.S. Naval Research Laboratory
paul.syverson@nrl.navy.mil

Matthew Traudt
U.S. Naval Research Laboratory
matthew.traudt@nrl.navy.mil

Abstract

HTTP Strict Transport Security (HSTS) was introduced to force clients to use HTTPS connections on sites that support it, thus preventing Man in the Middle and other attacks. HSTS has always been understood to potentially allow sites to track visiting clients, but this security threat has been considered outweighed by the security benefits it provides. With specific examples, verified on a website constructed to test them, we show that tracking is far more significant than previously recognized. We also demonstrate how to use our approach to censor individuals or classes of visiting clients. Further, we describe and demonstrate how third parties, such as site analytics services, can track clients across multiple domains. We discuss possible changes to allow users to control HSTS settings and better manage their security, and we compare and complement HSTS with HTTPS Everywhere, a popular browser extension with similar goals.

1 Introduction

HTTP Strict Transport Security was introduced in 2009 to counter several threats including DNS hijack of site access, redirection to an adversary controlled TLS protected site, and security protocol downgrading [7]. Users might intentionally or accidentally initiate connections to, for example, a website via HTTP when HTTPS was available. ('HTTPS' indicates that HTTP communication is sent over an encrypted TLS connection.) The goal was thus to force the client to request only TLS connections to that domain and to fail any other connections. Connections where there were problems in the TLS handshake (for example, a domain name mismatch or an unrecognized certificate authority) would fail without giving the user an option to click through warnings. The idea was to have the server securely send the client an HSTS header to parse and store locally, instructing it to behave as above in future connections to that domain. HTTP Strict Transport Security (HSTS) was then relatively quickly adopted as an IETF standard [8].

A trade-off of HSTS is that it allows a site to track users: under HSTS a client attempting an HTTP connection to the site indicates that it has not visited the site before. This tracking is not affected by the blocking or removal of cookies, leading some to describe it as using "supercookies". This was recognized as a potential concern virtually from the start, but it was felt that the advantages outweighed the disadvantages [4].

The trade-off has generally been recognized and occasionally discussed over the years since adoption. But the advantages have continued to win out, and today HSTS is embedded in all major browsers and widely employed by sites run by corporations, governments, organizations, and individuals. In March 2018, Apple announced both that they had seen such attacks deployed in the wild against Safari users and that they had implemented mitigations against them. Discussions about incorporating their mitigations into the HSTS standard are apparently underway [2].

1.1 Contributions

We will show that the scope and significance of the threats posed by HSTS have been hugely underappreciated to date. Attacks that we have implemented work on Safari and appear largely unaffected by deployed mitigations. Readers can verify this for themselves by visiting our demonstration site. We describe and demonstrate the use of HSTS settings for censorship of site visitors as well. Further we show that it is not merely the visited destination site that can do such tracking and censoring. We describe and demonstrate how third parties, such as advertisers, can use HSTS settings to track client behavior across diverse, unrelated sites that the client visits.

Our demonstrated attacks also illustrate the use of the `max-age` directive, which is required in HSTS headers, to identify when the client last visited a site. This can be

significant information in itself but can also be used to facilitate additional classification and/or censoring of visitors. Our demonstration site illustrates censoring users by changing the contents of the page based purely on HSTS headers placed on previous visits. We describe the combination of HSTS information from various user activities while visiting a site. Our analysis shows that it is unlikely any mitigation to our attacks exist that does not obviate the usefulness of sending HSTS headers.

We compare HSTS to a similar effort, HTTPS Everywhere [10]. HTTPS Everywhere (H-E) is immune to the above attacks. Though our analysis indicates that the primary, dynamic use of HSTS is dangerous for users at risk from general tracking or targeted surveillance and/or censorship, there are also preferable features of current HSTS that current HTTPS Everywhere lacks, as well as ways they can be used to complement each other.

2 Basics of HSTS and HSTS Tracking

A client attempting a connection via HTTP to a server supporting HSTS should be redirected to create an HTTPS connection to that server's domain. Within the HTTPS connection, the server will send an HSTS header instructing the client to subsequently connect only via HTTPS. (HSTS headers sent over HTTP are ignored to prevent Man in the Middle (MitM) insertion or removal of them.) An HSTS header has the format `Strict-Transport-Security: max-age=<expire-time>`, with two optional directives that we will discuss in Section 4.

The `max-age` directive gives the time in seconds that the HSTS setting should be respected. After it has expired, a client visiting that site will have the same HSTS behavior as one visiting the site for the first time. This limits the tracking we mentioned in Section 1, and it allows flexibility if, for example, the site is still transitioning to 100% HTTPS. Additionally, the site can change `max-age` to `0` to tell clients to immediately expire the relevant HSTS state.

As noted, an attempt to connect to an HSTS-configured server via HTTP indicates that a client has not visited that server previously (or recently). Building on that, a web page could, for example, contain several invisible images, each hosted on its own subdomain so that each has the potential for an entry in the client's local HSTS state. The site can therefore present during a client's first visit HSTS headers on some subset of invisible images, and then on subsequent visits build a vector comprised of one bit for each image and how it was fetched (0 for over HTTP, 1 for over HTTPS). With thirty images on a page, the site can uniquely identify over a billion distinct clients [4].

Attacks that motivated HSTS assume an adversary that can manipulate DNS and direct clients to the wrong server, where the adversary can perform MitM attacks or serve up censored/altered content. Users expecting DNS manipulation from their network environment may be especially concerned about being censored or tracked online. Thus the population most likely protected by HSTS may also be a population more vulnerable and less willing to accept the tracking that HSTS supports. Of course users who are not worried about or aware of DNS manipulation may also be concerned about tracking.

Users concerned about an adversary tracking which sites they visit might clear cookies; generally it is straightforward how to do so in a browser interface. HSTS, however, creates a so-called "supercookie" that will persist even if cookies are cleared. Depending on the browser, HSTS settings may not persist when switching to or from private browsing mode or when clearing temporary data by other means. What clears HSTS state varies with browsers in a not entirely predictable manner. For example, in Firefox one can instruct the browser to "Forget About This Site", which will remove local HSTS state for the domain; however, visiting `about:preferences#privacy` and clearing recent history won't remove local HSTS state unless the "Site Preferences" box is checked. In Chrome, removing an entry from history does not clear the associated HSTS state; however, visiting `chrome://settings/clearBrowserData` and checking the "Cached images and files" box will. Even a user monitoring HSTS settings might miss them: Firefox only writes new HSTS settings to disk when the browser closes. Numerous websites offer advice on how to clear HSTS settings, and readers can explore this for their own browsers following the guidance at our demonstration site described below. That is not our point here. First, any suggestion that users concerned about HSTS tracking simply remove HSTS settings is unrealistically facile for ordinary users and current popular browsers. Second, HSTS setting are applied automatically, and unlike cookies, browsers do not offer any user-friendly way to simply ignore HSTS headers. Third, clearing HSTS settings obviates the protections that creating them is supposed to provide.

Though long recognized as a theoretical possibility, in March 2018, Apple reported both observing such "supercookie" attacks against Safari users in the wild and modifications to Safari intended to counter them [2]. The solution they describe has two parts. First, if the currently loaded hostname is `bar.bar.bar.bar.foo.com`, then attempts to set HSTS state are ignored except for those applying to it exactly and to `foo.com`. Second, they use *Intelligent Tracking Prevention* (ITP) to ignore attempts to set HSTS state from any domain for which WebKit blocks cookies, "such as [a domain that serves] an in-

visible tracking pixel" [2]. Three months later they announced "the ability to detect when a domain is solely used as a 'first party bounce tracker,' meaning that it is never used as a third party content provider but tracks the user purely through navigational redirects. [...] ITP 2.0 detects such tracking behavior and treats those domains just like any other tracker, i.e. purges their website data" [15].

3 Surveiling and Censoring with HSTS

Apple's countermeasures may prevent or sufficiently mitigate the attacks they initially described. However, we have implemented a variant attack that works despite Apple's mitigations. To set state we make links first traverse a series of redirects each setting the desired HSTS state before dropping the user off at their intended destination. To check what state has been set previously, we made resources such as images and CSS first traverse a similar series of redirects before delivering the actual content. Safari 11.1.1 permitted 16 redirects. Chrome 67.0.3396.99 permitted 19, while Firefox 61.0 and Tor Browser 7.5.6 permitted 20 redirects. These and results described below were gathered using the the latest stable versions of each browser in late June 2018. We also tested using Safari Tech Preview Release 58, which became available June 6 2018 and is intended to incorporate the ITP 2.0 mitigations mentioned just above. Our test results were the same for Safari 11.1.1 and Safari Tech Preview Release 58.

Given sixteen or more redirects, a basic version of such an attack will permit partitioning users into at least 2^{16} (65,536) buckets. Note that previous discussions of HSTS tracking have described the number of unique individuals that can be identified, but it is not necessary to uniquely identify individuals for this to be an effective surveillance technique. If servers keep a record of client behavior and/or apparent client network location, then they can combine this with the partition information about returning visitors to further refine individuation within a partition.

Simply partitioning client visits is of little real use by itself, but it is not necessary for a malicious server to retain logs of previous behavior to learn useful information from client HSTS settings. By setting varying `max-age` values on the HSTS headers of tracking domains, a server can learn how long ago a returning client last visited a site. This temporal information can be combined with logs and the partitioning information from those HSTS settings that would not yet have expired. Files to demonstrate for oneself this and other results we describe are available on GitHub. Details are given below.

It might seem that, by limiting browsers to permit a much smaller number of redirects, a workable trade-off of user protections might still be achieved—if, for example, only 32 buckets could be generated from redirection by loading a page. This overlooks important points, however. First, loading a given page might involve the loading of multiple legitimate resources (such as visible images) that would be harder to separate off using WebKit modifications such as Apple introduced. Assuming a limit to five redirects, each of these is adding five bits to the HSTS state vector for that page.

More importantly, however, any webpage that offers users options to enter content or simply click on links provides an opportunity to add to the HSTS setting information created by that site visit. Further, if the site offers a navigation menu on all pages, it can offer different redirection domains for menu items based on what has been visited, and in what order, during that session.

For example, visiting `www.eff.org` and choosing "Tools" first from the navigation menu could yield different HSTS state than visiting "Take Action" followed by "Tools". Similarly, if the user visits both "Issues" and "Tools", different state can be set based on the order in which the user visited them. In general, each of the $k!$ ordered selection possibilities for k menu items adds n bits, given n permitted redirects per link and assuming no menu item is selected more than once. Of course what users are selecting is meaningful in itself besides simply increasing the number of identifiers or classifications they can be assigned.

The above attacks are primarily *hoovering* attacks that attempt to gather as much information as possible about any site visitors [12]. But an adversary may also be interested in specific visitors, e.g., those that follow particular links or enter certain keywords/information at parts of the site. Such a targeting adversary can hold off on storing very much HSTS state until a user has been deemed as worth targeting. Identifying the visitor as targeted need not happen in realtime; but if the visitor is recognized in realtime or deemed interesting either based on behavior exhibited thus far in the session or based on exogenous information such as connection IP address, then the redirects to set HSTS state can be optimized for refinement of classifying or identifying those users in future sessions.

In addition to the targeted placement of HSTS settings, the probing of site visitors for recognition or reidentification can also follow a targeted strategy. Adversaries can optimize the choice of initial HSTS state for which to test based on which targets are of greatest or most current interest. Similarly, they can optimize these to the expected next links most likely to be selected by targeted users, and they can optimize the domains to redirect through when those links are clicked on, just as when initially reconnecting to the homepage.

We have so far been describing customization of HSTS settings to facilitate tracking and classification of either targeted users or all users. Such customization can also be used to censor users by modifying the content/services offered on a site. Specifically, if a user's HSTS state reveals that they're an interesting user (perhaps because of a series of links visited previously), then the site owner can choose to simply not present (effectively block) or alter content/services offered. This is also an especially strong support for our earlier point that it is not necessary to precisely identify the specific previous individual visitor for HSTS tracking to be effective. If the current visitor can be recognized as having previously been interested in a type of content or service or having followed a type of site-visitor behavior, that is enough to censor their current visit. And as before, though HSTS state is enough, this could also be used in combination with other available information about the visitor, such as IP address, browser fingerprint, server logs of previous sessions, etc.

Finally, we have also so far only been describing tracking or censoring users *within* a single site. An ad network, Content Delivery Network (CDN), site analytics service, or in general any party that has its resources loaded into a wide variety of websites can track or censor users *across many* sites.

3.1 Demonstrations and browser tests

We set up a site and performed tests to determine results and demonstrate attacks described above.

For readers who wish to set up their own test pages to determine the number of redirects a browser permits, HTML files are available at `https://github.com/pastly/satis-hsts-tracking` along with a readme file describing how to use them.

We created a simple demonstration of both cross-domain tracking and cross-domain censorship using only HSTS settings—no cookies or other tracking mechanisms. In our demonstration, first we visit `https://pastly.xyz/test01.html` and take note of the Chrome browser logo there. Then we open `http://hsts.satis.system33.pw` and click on the link to the `superuser.com` page about how to check HSTS state in Chrome. When we go back to `pastly.xyz` and refresh the page, the Chrome logo has been replaced by an image of a cat. In basic form, this works in Chrome, Firefox, Safari, and Tor Browser.

All browsers seemed to respect keeping separate HSTS state updated while in private browsing mode versus state updated while not in private browsing mode. (Chrome calls private windows "incognito".) And none of these browsers stored HSTS updates from a private browsing window once all private windows were closed. Chrome and Firefox, however, did permit such cross-domain tracking and censorship within browsing sessions even between different private browsing windows or tabs. Indeed, even if the `http://hsts.satis.system33.pw` private window was closed and only some unrelated private browsing window remained open, HSTS state was preserved and manifested with the appearance of a cat if `https://pastly.xyz/test01.html` was subsequently opened. Thus, at the moment users might not be getting the protection and separation they expect from opening a new private browsing window in one of these browsers.

Safari did not permit cross-domain HSTS state to bleed from one private tab to another, even during the same browsing session. Tor Browser 7.5.6 did, but the alpha version of Tor Browser available at time of writing (8.0a9) no longer permitted such bleeding. Safari seems to be separating state based on separate private tabs whereas Tor Browser 8.0a9 seemed to separate based on URL: HSTS state was accessible to new private tabs within a session if they displayed the same URL as when the HSTS state was set. This bleeding of cross-domain state information also occurred when opening the first connection in one Firefox tab group and the second connection in a different Firefox tab group.

A minute-long video is available at `https://github.com/pastly/satis-hsts-tracking` showing a version of this censorship attack (but not cross-domain), time-since-last-visit tracking via `max-age`, and two of the ways users might try to clear website records from Chrome—one removes HSTS state and the other does not. Chrome 67.0.3396.99 and Firefox 61.0 permitted such tracking from HSTS state that is loaded cross-domain via CSS. Safari does not (whether in private browsing mode or not). Tor Browser 8.0a9 loads HSTS state via CSS, but as above appears not to permit other domains access to it, only the domain that appeared in the URL bar when it was set. Safari still accepts HSTS headers received in response to clicks, so ad networks can still set HSTS state when their ads are clicked on, and in general JavaScript can be used to hijack clicks on links and direct them into an HSTS-state-setting redirect chain before dropping the user off at their intended destination. Another short video available there shows a CSS cross-domain tracking attack. There is also a readme file walking through the videos, along with configuration and other files sufficient for readers to set up their own test websites.

Note that when loading `hsts.satis.system33.pw` or clicking on any of the links on the page that return there, the entire loading process was at most a few seconds, essentially indestinguishable from a normal user experience when loading a webpage. And in no browser did the redirects change what was displayed in the URL

bar at all. The one exception was that Safari showed redirected links only when clicking on the link to start the demo of time-of-last-visit tracking, and fleeting URL bar displays of redirects are quite common when loading webpages in general. Similarly, all browsers took at most a few seconds in tests of the redirect chain length mentioned above, and all but Safari displayed only the ultimate destination URL in the URL bar. Again, this is consistent with common experiences of surfing the web. Thus, none of the attacks or examples from our demos exhibit anything that would look or seem odd to ordinary users, and mostly they were literally indistinguishable from the normal experience of loading a static webpage.

The files we have made available on GitHub are designed to support safe demonstrations of our results as well as related testing. They cause storage of information only on the client; no records of client visits are retained on the server. The information stored on the client is entirely HSTS state derived from HSTS headers, though of course if we tracked user behavior this could be combined. Finally, `max-age` for all headers is at most 30 minutes since we are not actually interested in tracking visitors. There should be no HSTS record on the client of having visited a demo site at all beyond half an hour following the last visit. A site using our files will both demonstrate attacks and provide or give links to instructions for how to find and/or remove HSTS settings from Chrome, Firefox, Tor Browser, and Safari. Since Safari stores HSTS settings in a binary file, we have also provided a Python script to render it into human-understandable form.

We do not intend in our current research to extensively set out all the possible surveillance and censorship variants that can occur from HSTS. And though we have noted Apple's claim of having detected HSTS state attacks against Safari users in the wild [2], we have not attempted to measure indications of how or to what extent these attacks have moved beyond the anecdotal stage. Our goal is simply to show how significant the potential threat is in order to motivate change, so as to prevent a situation where someone *is* publishing a paper at FOCI 2025 documenting how these attacks have been extensively used in the wild. And though we feel that our analysis is straightforward and simple enough to be compelling, we have also implemented simple examples of tracking and censorship and made available the files needed to set up demonstrations of these for oneself, along with videos walking through some of them.

4 Preemptively Bootstrapping HSTS

A client visiting an HSTS-enabled domain for the first time will not automatically initiate connection to it within the protection of TLS. Rather, a client initiating an HTTP connection would be redirected to an HTTPS connection, over which the server would send an HSTS header. This first-time client could thus be just as vulnerable to MitM attacks as when visiting a non-HSTS-enabled domain. The client is similarly vulnerable if it visited the site further in the past than the `max-age` value it had set for that domain. Further, published attacks on HSTS have also described significantly altering NTP to effectively expire the `max-age` setting [14].

To counter any attacks based on visiting sites for the first time (or first time in a while), domain owners can add the `preload` directive to the HSTS headers they send, which will then satisfy one of the criteria to submit for entry on the HSTS preload list [9]. This HSTS preload list is managed by Chrome, and most major browsers use it. Browsers will use HSTS settings from this list for connecting to any domain, even one that has not been visited previously.

Having a site on the HSTS preload list does not preclude the dynamic tracking and censorship attacks described above. As we have observed, tracking can be handled at a different domain. We were considering typical third parties such as advertisers or site analytics services. But site owners can redirect to other domains they themselves own even when loading the site homepage. And as noted, in our tracking tests of browsers none displayed the URLs of intermediate redirected domains, except that Safari flashed these very briefly in the how-recently-visited test. Thus there may be nothing at all that is visible to the user, and, more certainly, nothing that appears odd. (It is still feasible to display different *content* to the visitor based on the identified HSTS bit vector.)

Returning from tracking and censorship concerns to MitM protection, adding a site to the HSTS preload list is not a trivial undertaking. First, a site is only eligible if it can set a `max-age` of at least one year and must also use the `includeSubDomains` directive, which will thus require HTTPS for all subdomains (and nested subdomains). Guidance at the HSTS Preload site suggests incremental testing taking at least a few months before deploying [9]. Also, "inclusion in the preload list cannot easily be undone. [...] Don't request inclusion unless you're sure that you can support HTTPS **for your entire site and all its subdomains** the long term." Thus, site owners must be prepared for potentially extensive retooling and/or testing of their sites prior to requesting to join the list, but they must also be confident that they will have no reason to change *any part of* their site in an HSTS incompatible way for a very long time. They must similarly monitor all parts of their operation for accidental changes that would clash with being on the HSTS preload list. Further, site owners who choose to go ahead must monitor to make sure they remain on the list: " [...]

sites may be removed automatically in the future for failing to keep up the requirements." This all makes adding one's site to the preload list a serious undertaking with mixed incentives.

As a specific example, one of the authors recently received a prompt to update an expiring system password on a large enterprise site with hundreds of thousands of registered users. Attempting to connect to the relevant URL yielded a page that claimed the owner of the site, "has configured their website improperly. [. . .] This site uses HTTP Strict Transport Security (HSTS) to specify that Firefox may only connect to it securely." The domain of the visited site is not itself on the HSTS preload list, but some of its pages redirect to a domain that is. It remains unclear if there was any actual HSTS website misconfiguration, per se. But some pages within the site could be accessed while others effectively could not (in one case depending on whether a terminal "/" was included or not in the URL entered in the address bar).

Analysis of the state of HSTS deployment in 2013 showed improper configuration on many sites, leaving them open to attacks HSTS is meant to counter [3]. This is perhaps normal for any complex protocol less than a decade from initial design and only a few years after standaradization. Also, there has been much progress since 2013, and HSTS is now set by shared service providers such as CloudFlare. The study underscores, however, that unless site administrators skillfully set up and persistently maintain configuration of their sites, they might not be getting the protection from HSTS that they are expecting. And as already noted, if browsers are configured to fail safely, they will not get the desired functionality either.

HSTS preload guidelines recommend a months-long process of configuration and testing before a site owner requests addition to the list [9]. But, according to someone responsible for HSTS and preload rollout on multiple large sites, "For most sites, once they've migrated to HTTPS, adding HSTS support is straightforward. However, large and content-heavy sites with rich mixed content and disparate CMS backends may need to migrate their website in chunks, rather than a full migration at once, which would make HSTS difficult until site migration is complete. Many major news sites fit this category." And, "For most domains, once HSTS is in place, preloading the domain is straightforward, even trivial. However, domain names that are used for a sprawling array of services (such as an organization's primary 'corporate' domain) may need to migrate their subdomains in chunks, rather than migrating the entire zone at once, which would make preloading difficult until the migration of all subdomains is complete. This is particularly true of zones where split-horizon DNS is in use, and intranet sites are used within the same zone as an organization's internet-facing services."

Still, it is generally over a month from request till a site is added to the HSTS preload list, and then it can be another month until that update makes its way to updated client browsers [5]. For those putting a new site online or newly making a site compliant with HSTS preload, it would be better if they did not have to choose between making the site available now versus protecting first time visitors during the interval until their HSTS-preload status is incorporated into most clients.

5 HSTS and HTTPS Everywhere

HTTPS Everywhere [10] is a popular browser extension that functions roughly like HSTS preload: any HTTP request for a URL covered by the HTTPS Everywhere (H-E) ruleset is rewritten in the address bar to an HTTPS request for the appropriate URL. (For some sites, this is not always simply the original URL, and H-E rules are more flexible than HSTS redirects.) If the "Block all unencrypted requests" box is selected from the H-E toolbar GUI, then like HSTS, it will not be possible to load unencrypted connections. (Unlike HSTS, it will still be possible to grant an exception for TLS certificates from unknown issuers by going into "Advanced" options in the browser warning.) Like the HSTS `includeSubDomains` directive, one can write H-E rules that require connections to any subdomains of a given domain to always be via HTTPS. H-E rules are more flexible than HSTS, and are much faster and easier to roll out, on the order of days. In fact, HTTPS Everywhere rules used to be updated only when a new version of the extension was released; however, recently the EFF announced "continual ruleset updates" for H-E, meaning that the extension will check with EFF to see if a new ruleset list is available without having to wait for an extension update [1].

Currently, HTTPS Everywhere encourages site owners to favor HSTS and joining the HSTS preload list over adding HTTPS Everywhere rules for their sites [11]. The stated reason is because HSTS is built into all major browsers, thus works by default for nearly all clients. H-E, on the other hand, is an optional extension, thus has a much smaller userbase. While that is true, given the risk noted in the last section during the month(s) until a request to put a site on the HSTS preload list is reflected in browsers, it would be wise to add an H-E rule to the default ruleset at least until the site has been on the HSTS preload list for a bit.

The most important difference between HSTS and H-E is that H-E does not allow servers to selective create new settings at clients dynamically per connection and/or based on activity during a connected session. Thus H-E is not subject to the tracking or censorship that we have seen is possible via HSTS. HTTPS Everywhere

rules are retrieved from the H-E ruleset given to all clients. The new "continual ruleset" change does allow customized rulesets or "update channels" that third parties can use [1]. Care must be taken to evaluate the tracking potential of allowing multiple update channels, the number of clients using a given update channel, and the frequency of updates. Nonetheless, these updates are not created in response to timing or pattern of access to particular domains. The HTTPS Everywhere interface allows creation of custom rules for sites, but these are created by the user, not given by the server.

HSTS settings are also not easy for users of most browsers to discover, remove, or determine if they have been removed. On the other hand, H-E rules, which apply to all HTTPS Everywhere enabled clients, are easily discoverable. And a simple checkbox on the interface allows users to turn HTTPS Everywhere on and off.

6 Conclusions and Recommendations

It may be possible to alter browsers to counter particular attacks set out in Section 3, but the fundamentally interactive nature of dynamic HSTS makes it very unlikely that there is a way to avoid the capability for significant censorship and tracking of client behavior by site owners or cross-domain by third parties. Users for whom the tracking of online activity may have operational or personal security implications may wish to completely avoid accepting of HSTS headers, as might users simply uncomfortable with the possibility that HSTS might be used for tracking or to alter the content/service options they are given. One might thus expect that Tor Browser or other browsers placing a premium on safety will want to block HSTS headers in the future, but this would be a security trade-off not a pure win as long as many sites remain accessible by both HTTP and HTTPS. It may also be desirable to have a default such as H-E's GUI option to block all unencrypted requests. That remains a security/functionality trade-off on today's Internet, though it seems a diminishing one.

At the very least, browsers should provide a simple user-friendly way both to learn HSTS settings for a domain and to remove them if desired. Chrome appears to be the only major browser to facilitate this at all (by visiting `chrome://net-internals/#hsts`), but not in the same place as are options for clearing history, cookies, or cached images (`chrome://settings/clearBrowserData`), and it does not provide an interface to simply clear all dynamically set HSTS state. Removal of all HSTS settings in Tor Browser occurs when "New Identity" is selected from the onion popup UI. More generally, it should also be possible and easy for users to choose to have their browsers ignore HSTS headers, either all headers or for specific domains. None currently permit ignoring HSTS headers, though all will drop HSTS state added in private browsing mode, once *all* private browsing windows are closed.

Limiting HSTS to preloaded sites would prevent dynamic attacks while failing safe for clients attempting to access such sites insecurely. As we have noted, however, it currently requires substantial lead time for a site to be added to this list. It is thus reasonable to add the site to the H-E ruleset so that those clients with H-E installed can be protected until the preload update takes effect.

If we do limit to a preload list, scaling becomes another concern. Currently the HSTS preload list includes on the order of fifty thousand domains, and the H-E ruleset covers about half as many. These cannot scale up to Internet size: while as of Feb 2018 about 10% of Alexa Top 1M sites were using HSTS [6], as of June 2017, only .337% of Alexa Top 1M sites were on the HSTS preload list, though this was more than double the fraction as of April 2016 [13]. Perhaps by the time scale is an issue, the portion and significance of the web that is unencrypted will be small enough that it can be ignored or only permitted access by experts (or at least behind sterner warnings and barriers).

Even if one chooses to live with the tracking and censorship threats of dynamic HSTS (possibly with management options as we have described), this does not protect against the first connection MitM threat for those sites. One can install H-E and set it to block all unencrypted requests. Besides the usability impact of this choice already noted, as long as dynamic HSTS is accepted, an adversary can still create ways to track clients by noting which subset of requests are received at all.

Forcing all connection requests to attempt HTTPS, only reverting to HTTP if this fails, could maximize both hijack and tracking/censorship protection with less likelihood of complete failure for unecrypted connections. This will likely have a usability-impacting performance cost for any resources still not reachable by HTTPS. And, this may simply break functionality in some cases, including where the HTTPS connections for a site are at a different URL than the HTTP option (something that H-E does support and for which rules currently exist). Protocol and system changes to better provide clients secure initial connections to websites without significant tracking, censorship, functionality, or scaling issues will require additional research.

Acknowledgments

The authors would like to thank the anonymous reviewers as well as Bill Budington, Jennifer Helsby, Georg Koppen, Eric Mill, Seth Schoen, and Ryan Wails for helpful comments on drafts of this paper that greatly improved it.

References

[1] BUDINGTON, B. HTTPS Everywhere introduces new feature: Continual ruleset updates. https://www.eff.org/deeplinks/2018/04/https-everywhere-introduces-new-feature-continual-ruleset-updates, April 3 2018.

[2] FULGHAM, B. Protecting against HSTS abuse. https://webkit.org/blog/8146/protecting-against-hsts-abuse/, March 18 2018.

[3] GARRON, L., BORTZ, A., AND BONEH, D. The state of HSTS deployment: A survey and common pitfalls. https://garron.net/crypto/hsts/hsts-2013.pdf, 2013.

[4] Chrome fixes STS privacy issue. https://web.archive.org/web/20100417094217/http://ha.ckers.org/blog/20100413/chrome-fixes-sts-privacy-issue/, April 13 2010.

[5] HELME, S. Testing the HSTS preload process. https://scotthelme.co.uk/hsts-preload-test/, July 22 2016.

[6] HELME, S. Alexa Top 1 Million analysis - February 2018. https://scotthelme.co.uk/alexa-top-1-million-analysis-february-2018/, February 26 2018.

[7] HODGES, J., JACKSON, C., AND BARTH, A. Strict transport security. https://lists.w3.org/Archives/Public/www-archive/2009Sep/att-0051/draft-hodges-strict-transport-sec-05.plain.html, September 9 2009.

[8] HODGES, J., JACKSON, C., AND BARTH, A. HTTP Strict Transport Security (HSTS). https://tools.ietf.org/html/rfc6797, November 2012.

[9] HSTS preload list submission. https://hstspreload.org/.

[10] HTTPS Everywhere. https://www.eff.org/https-everywhere.

[11] How do I add my own site to HTTPS Everywhere? https://www.eff.org/https-everywhere/faq/#how-do-i-add-my-own-site-to-https-everywhere.

[12] JAGGARD, A. D., AND SYVERSON, P. Onions in the crosshairs: When The Man really is out to get you. In *ACM Workshop on Privacy in the Electronic Society (WPES '17)* (Dallas, Texas, USA, October 2017), ACM.

[13] KING, A. Analysis of the Alexa Top 1M sites. https://blog.mozilla.org/security/2017/06/28/analysis-alexa-top-1m-sites/, June 28 2017.

[14] SELVI, J. Bypassing HTTP strict transport security. In *Black Hat Europe* (2014). https://www.blackhat.com/docs/eu-14/materials/eu-14-Selvi-Bypassing-HTTP-Strict-Transport-Security-wp.pdf.

[15] WILANDER, J. Intelligent tracking prevention 2.0. https://webkit.org/blog/8311/intelligent-tracking-prevention-2-0/, June 4 2018.