Securing the Web with HSTS-Enforced

Aaron van Diepen
Delft University of Technology
Delft, The Netherlands

Adrian Zapletal
Delft University of Technology
Delft, The Netherlands

Fernando Kuipers
Delft University of Technology
Delft, The Netherlands

***Abstract*—TLS stripping attacks expose sensitive web traffic by forcing secure HTTPS connections to fall back to unencrypted HTTP. At present, protection against these attacks relies on website operators explicitly opting into security by deploying mechanisms such as HTTP Strict Transport Security (HSTS) headers. These mechanisms have significant limitations: some are weak or difficult to configure, which raises the risk of misconfiguration and reduces practical adoption; others violate HTTP backward compatibility; at least one can even be abused to enable unintended user tracking.**

We introduce HSTS-Enforced, a mechanism that eliminates the remaining attack surface for TLS stripping while still allowing operators to securely specify that their websites need to be accessed over HTTP when necessary, thereby maintaining accessibility. To achieve this, we flip the current opt-in security model to an opt-out model: all connections default to HTTPS, and operators can explicitly opt out if their websites require HTTP using so-called HTTP-Required indicators. We propose two such HTTP-Required indicators: a new DNS record and an HTTP-Required Preload list. We evaluate HSTS-Enforced under multiple deployment scenarios, demonstrating that it blocks all practical TLS stripping attempts while maintaining compatibility for sites that require HTTP – without introducing overhead in the typical case. Finally, we outline a practical transition path to accelerate global adoption.

***Index Terms*—Transport Layer Security (TLS), HTTPS, HTTP Strict Transport Security (HSTS), Domain Name System (DNS), SSL stripping, Downgrade attacks, Web security, Secure-by-default**

I. Introduction

The initial milestone in web security was the introduction of HTTPS (Hypertext Transfer Protocol Secure) [1], a protocol that employs TLS (Transport Layer Security) to encrypt communications between clients and servers, ensuring data confidentiality and integrity even if transmissions are intercepted.

Despite this, a practical challenge remains: web clients must implicitly determine whether to initiate a secure HTTPS connection or fall back to unsecured HTTP (Hypertext Transfer Protocol). A man-in-the-middle attacker can perform TLS stripping (§II-A), which is an attack that forces connections to use unencrypted HTTP. Several mechanisms exist to protect against TLS stripping. However, each suffers from notable limitations. These limitations motivate the design of a more robust and deployable protocol with the following requirements: strong security guarantees, resilience to misconfiguration, ease of achieving secure configurations, interoperability with HTTP when needed, and preservation of user privacy (§II-B). We review existing approaches and analyze the extent to which current protocols fail to satisfy these objectives (§II-C).

To address the shortcomings of existing approaches, we propose HSTS-Enforced (§III-A), a mechanism that builds on HSTS (HTTP Strict Transport Security) [2]—a policy that instructs web clients to interact with a given website exclusively over HTTPS for a specified duration—by making this behavior the default for all websites. Rather than requiring operators to opt into this state, web clients default to only creating HTTPS connections unless website operators explicitly allow the use of HTTP by specifying an HTTP-Required indicator. We propose two such HTTP-Required indicators (§III-B): (1) a new DNS (Domain Name System) record, HTTPREQ, which leverages the cryptographic chain of DNSSEC (DNS Security Extension) to prove that a website requires HTTP, and (2) an HTTP-Required Preload list, which pre-configures websites to allow HTTP. To improve the accessibility of websites, we use the additional security provided by HSTS-Enforced to optimize the connection process of web clients (§III-C). We implement HSTS-Enforced, the proposed indicators, and the optimized connection process in the Chromium browser and several DNS software suites (§IV) and use it to show that HSTS-Enforced prevents TLS stripping without inducing significant overhead (§V). Finally, we outline a practical transition path to accelerate global adoption of HSTS-Enforced (§VI). Additional design details and evaluation results are provided in an extended version of this paper [3].

II. Background & Related Work

A. TLS Stripping Attacks

TLS stripping is a downgrade attack in which an adversary forces a web client to communicate over unencrypted HTTP even though the target website supports HTTPS [4], [5]. The attack exploits the common web client behavior of reverting to HTTP when an HTTPS connection attempt fails. By actively blocking HTTPS traffic, a man-in-the-middle can prevent the establishment of a TLS session and coerce the client into using plaintext HTTP.

Once the connection is downgraded, the attacker can eavesdrop on sensitive data such as authentication credentials and personal information or modify the traffic in transit to inject malicious content. Depending on the attacker's network position, these attacks can target all users accessing a specific website or all websites accessed by a particular user or group of users.

Our threat model assumes an attacker positioned anywhere along the network path between client and server (e.g. a malicious Wi-Fi hotspot operator, a compromised ISP router, or a nation-state interceptor). The attacker can intercept, block, delay, and replay encrypted traffic; if traffic is unencrypted, they can also read, modify, and forge messages. The attacker

ISBN 978-3-903176-82-9

has no direct control over either endpoint: they cannot compromise the user's device or the remote server, install software, or steal cryptographic keys.

This threat model gives the attacker several concrete capabilities. They can perform TLS stripping to silently downgrade HTTPS connections to HTTP as described in §II-A. They can also tamper with DNS responses, since these travel in plaintext. DNSSEC mitigates this by cryptographically signing responses, making silent modification detectable. However, the attacker can still drop responses to cause lookup failures. DoH (DNS-over-HTTPS) encrypts DNS queries in transit but does not solve the underlying problem: it merely shifts trust to the resolver, which could return fraudulent responses to the client. Furthermore, DoH requires the user to both explicitly enable it and select a resolver they trust. We consider this an unreasonable burden to place on general users, who should not need to reason about the trustworthiness of DNS infrastructure to browse the web securely.

B. The Goal

We seek a solution that *fully* secures web connections against TLS stripping attacks and all subsequent attacks against HTTP by the attacker outlined in our threat model. The strawman solution would be to never use HTTP at all. However, there are valid reasons for using HTTP: old systems may have performance limitations that impede cryptographic operations; website operators may require HTTP for testing purposes; some website operators are simply not willing to acquire an X.509 certificate; and finally, local routing devices, such as routers and wireless extenders, commonly provide a configuration interface that is hosted directly on the device, and including an X.509 certificate would allow malicious users to extract the private key linked to the certificate.

The desiderata for a mechanism that prevents TLS stripping attacks are as follows:

① *Maximum security.* The mechanism should comprehensively prevent TLS stripping attacks.

② *Resilience against misconfigurations.* Incorrect configurations should not lead to a loss of security.

③ *Minimum effort.* Operators should be able to reach the maximum level of security as easily as possible.

④ *No strict enforcement.* The mechanism shall allow connections over HTTP when explicitly enabled by operators and for some special exceptions, provided this is done in a secure and controlled manner.

⑤ *No user tracking.* The mechanism should not enable websites to track users via user-specific responses.

C. Related Work

This section mostly focuses on prior solutions that aim to prevent TLS stripping and evaluates whether they fulfill our desiderata (§II-B). Table I summarizes the results.

HSTS Headers [2] are communicated as part of an HTTPS response, after which a web client stores them in their cache. This enables HSTS for that website, which enforces connections to a website to use HTTPS with a trusted certificate. Nevertheless, HSTS headers *fail to provide maximum security* ①; although they secure subsequent connections, the first connection to a website is still vulnerable to attacks.

TABLE I: Comparison of TLS Stripping Prevention Mechanisms Against Defined Desiderata (§II-B).

Mechanism	Desiderata				
	①	②	③	④	⑤
HSTS Headers [2]	✗	✗	✗	✓	✗
HSTS Preloading [6]	✓	✗	✗	✓	✓
HTTPS Records [7]	✓	✗	✗	✓	✓
HTTPS-First Mode [8]	✗	✓	✓	✓	✓
HTTPS-Only Mode [9]	✓	✓	✓	✗	✓
HSTS-Enforced	✓	✓	✓	✓	✓

To prevent accessibility issues, HSTS headers specify how long they should be cached; after they expire, connections become vulnerable again. Additionally, the cache is considered part of the client's history; deleting the history clears the stored HSTS headers. They are also *not resilient against misconfigurations* ②: many websites send erroneous HSTS headers, causing user agents to not enable HSTS for these websites [10]–[13]. Additionally, configuring HSTS headers requires more than *minimum effort* ③, causing many websites to not adopt them [10]–[12], [14]–[17]. Measurement results range from an adoption rate of 0.6% on a list of websites obtained by scanning IP addresses for open HTTPS ports [11] to 46% on a list of the top one thousand websites [12]. Finally, because HSTS headers are part of HTTPS responses, which can differ per user, they *unwittingly facilitate user tracking* ⑤ [18].

HSTS Preloading [6] relies on the HSTS Preload list, which is a list maintained by the Chromium project [19]. Any website included in the list is pre-configured to always enable HSTS. The list removes the need for HSTS headers to be communicated using HTTPS, which mitigates some vulnerabilities of HSTS headers. However, the list does *not provide resilience against misconfigurations* ② and *fails to require minimum effort* ③: website operators can request their domain to be added to the HSTS Preload list using a registration website maintained by Chromium [6], but the registration process requires correct configuration of website redirects and HSTS headers, causing the HSTS Preload list to see very low adoption rates by website operators [10], [11]. Through manual scanning, we found that the websites of most major international banks are not on the list, although banking websites are precisely the ones that should protect against TLS stripping attacks. Additionally, if the configuration required for addition to the list is ever broken, anyone can request the domain's removal. Currently, there are only around 165.000 domains on the list that have not broken their configuration [20], which is a relatively low number considering the size of the Internet.

HTTPS Records [7] are DNS records that indicate which HTTPS versions a domain supports. Theoretically, a DNSSEC-verified negative response for such records could be treated as an opt-out of HSTS. However, this mechanism would require a broad adoption of HTTPS records by all websites that do not wish to opt-out, and the accidental removal of these records would lead to a loss of security. These issues have been shown to be common [21]–[23]. Therefore, HTTPS records do

not provide *resilience against misconfigurations* ② and do not require *minimum effort* ③. Similarly, any other DNS record that operates under an opt-in security model, such as for example using *DANE (DNS-based Authentication of Named Entities) records* would fail the same desiderata.

HTTPS-First Mode [8] is a client-side policy that attempts to establish every connection over HTTPS before falling back to HTTP. This mechanism fails to provide *maximum security* ①. Its security is opportunistic rather than absolute. An active adversary can exploit this by blocking the HTTPS connection, causing the client to proceed over HTTP.

HTTPS-Only Mode [9] is another feature mostly integrated into browsers. Previously known as the browser extension *HTTPS Everywhere* [24], it blocks all HTTP connections using a user-facing warning. Unlike HTTPS-First mode, HTTPS-Only mode is not enabled by default in most browsers and can be enabled through the settings if desired. This mechanism fulfills all desiderata but breaks accessibility on websites that are restricted to HTTP. Thus, it fails to *not strictly enforce HTTPS* ④. The only difference to our strawman solution (§II-B) is that users can still ignore warnings to access insecure websites. However, doing so effectively leads to the loss of *maximum security* ①. The primary limitation of this mechanism is its heavy reliance on users to determine whether it is acceptable to disregard a given warning.

Combinations of mechanisms can be and are being used. However, for a combination to provide *resilience against misconfigurations* ②, require *minimum effort* ③, and *not strictly enforce HTTPS* ④, all mechanisms used in the combination must fulfill these three desiderata. There exists no combination of two mechanisms such that both mechanisms fulfill these three desiderata. Hence, no combination fulfills all five desiderata.

III. HSTS-Enforced

We propose HSTS-Enforced, a solution where websites default to HSTS and can explicitly opt out of security rather than opting in. In what follows, we provide an overview of HSTS-Enforced and explain how it fulfills our goals (§III-A), propose two indicators used to opt out of security (§III-B), and use the additional security to optimize the connection process of HSTS (§III-C).

A. Overview

By default, HSTS-Enforced enables HSTS for all websites. If the use of HTTP is desired, operators can explicitly opt out of HSTS. By ensuring opt-outs are unspoofable, we achieve *maximum security* ①. By defaulting to HSTS, we *prevent misconfigurations from stifling security* ②. Moreover, because operators no longer need to explicitly opt into security, HSTS-Enforced requires *minimum effort to secure web connections* ③. *We do not force all connections to use HTTPS* ④: websites can indicate that they opt out of security. Finally, because HSTS-Enforced prevents user-specific HSTS settings, it *prevents websites from using HSTS state to track users* ⑤.

In what follows, we provide an overview of our two proposed HTTP-Required indicators and explain how they maintain security and prevent introducing new tracking vectors (§III-B) Finally, we detail an optimized connection process that minimizes the instances requiring resolution of an indicator and maximizes website accessibility (§III-C).

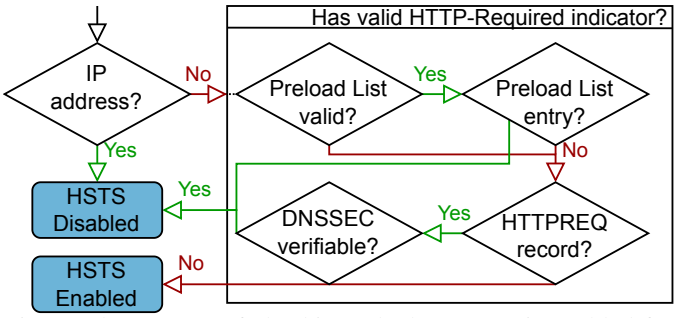

Fig. 1: The process of checking whether HSTS is enabled for a domain with HSTS-Enforced.

B. HTTP-Required Indicators

To opt out of HSTS, websites should use HTTP-Required indicators. Such indicators must satisfy two demands: (I) *the indicators must not be spoofable*, and (II) *web servers cannot provide a unique (set of) indicator(s) per user*. These properties prevent the reintroduction of weaknesses that could enable TLS stripping or user tracking. We propose two such indicators: the HTTP-Required Preload list and the HTTPREQ DNS record. When scanning for HTTP-Required indicators, the order of checking indicators should be optimized to reduce network traffic and latency. For our proposed indicators, HSTS-Enforced first reads the HTTP-Required Preload list and then checks for the presence of an HTTPREQ record, as this order minimizes the amount of DNS traffic. Fig. 1 illustrates the process of checking whether HSTS is enabled or disabled with our proposed indicators.

The HTTP-Required Preload list functions like the HSTS Preload list in reverse: it lists domains that opt out of HSTS. The list is frequently redistributed and contains an expiration date; entries are re-verified before each release. Operators request inclusion via a registration website, which verifies from multiple secure vantage points that HTTP responses carry an `HTTP-Required` header, analogously to Let's Encrypt's domain validation [25], which prevents attacks against this indicator (I). Similar to the HSTS Preload list, the HTTP-Required Preload list is integrated directly into clients, preventing user-specific indicators (II). If the list expires without a client update, access to HTTP-only sites is blocked—a preferable trade-off to compromised security.

HTTPREQ DNS records use the cryptographic chain of DNSSEC to create a secure indicator. The presence of a record, combined with a valid DNSSEC signature, serves as the indicator. To make the indicator unspoofable, web clients verify the full cryptographic chain locally (I). DNS propagates records consistently, ensuring that all receivers access the same information (II).

C. Connection Process

HSTS-Enforced decides which connections should be made based on which scheme is specified in the URL and on the HSTS state of a website. In all cases, HSTS-Enforced delays checking the HSTS status until right before making an unsecure connection. In the common cases of *no scheme*

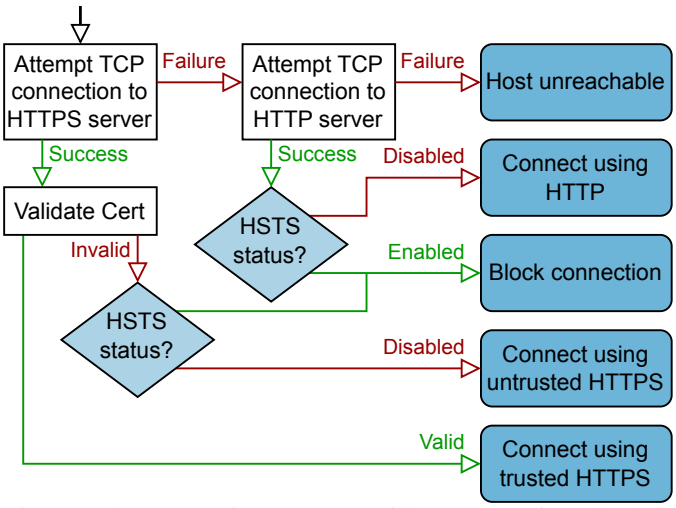

Fig. 2: The connection process with HSTS-Enforced when HTTPS or no protocol are specified in the URL scheme. Nodes labeled "HSTS status?" perform the process shown in Fig. 1.

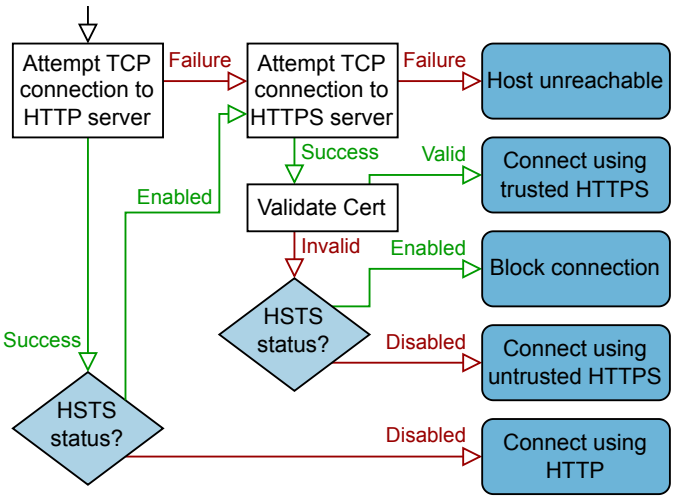

Fig. 3: The connection process with HSTS-Enforced when HTTP is specified in the URL scheme. Nodes labeled "HSTS status?" perform the process shown in Fig. 1.

or *HTTPS*, HSTS-Enforced always first attempts to establish a connection to the website's HTTPS server using a trusted certificate and falls back to untrusted HTTPS and HTTP only if HSTS is disabled (Fig. 2). When *HTTP* is explicitly specified in the URL, HSTS-Enforced follows this directive by first trying to establish an HTTP connection if allowed and will always attempt to fall back to HTTPS if unsuccessful (Fig. 3). Attempts to establish a trusted HTTPS connection will often succeed [26]. In case an HTTPS server is found but the certificate it replies with is untrusted, HSTS-Enforced will always check whether HSTS is disabled before establishing such connections. When attempting an HTTP connection, HSTS-Enforced will first check if an HTTP server even exists. Since HTTP relies on TCP, checking if an HTTP server exists does not require possibly sensitive data to be sent but only requires an attempt at establishing a connection. If an HTTP server can not be found, HSTS-Enforced can skip checking the HSTS status in favor of immediately attempting trusted HTTPS connections if not attempted before. If an HTTP server is found, it continues by checking the HSTS status and only allows the connection if it is disabled. If HSTS-Enforced cannot reach the HTTP server or if HSTS is enabled, it switches to HTTPS.

IV. Implementation

To demonstrate the feasibility and assess the performance of HSTS-Enforced, we provide a complete implementation. This implementation includes a modified browser that uses HSTS-Enforced to initiate web connections and can request and verify the proposed HTTP-Required indicators. Additionally, we implement the DNS-based HTTP-Required indicator within a DNS server and resolver. The full source code is available in the accompanying Git repository [27].

A client supporting HSTS-Enforced was implemented by modifying the Chromium browser [19]. We removed all code related to HSTS headers, the HSTS cache, and the HSTS Preload list, as it conflicts with HSTS-Enforced. Afterwards, we added the functionality for checking HTTP-Required indicators and modified Chromium's connection setup to follow the procedure described in §III-C.

A web service for website operators to request addition to the HTTP-Required Preload list was implemented, including a REST API and a website with a graphical user interface. The service accepts requests for addition and verifies their validity. It stores accepted entries in a database. Moreover, a recurring task checks all entries in the database for compliance every six weeks. Thereafter, it publishes a new version of the list.

The HTTP-Required Preload list was integrated into Chromium. Its implementation is similar to that of the HSTS Preload list: it represents the list as a Huffman-encoded trie [28], [29]. This data structure uses little memory and enables searching the list in linear time relative to the length of the domain name.

HTTPREQ DNS record support was added to three DNS projects: PowerDNS [30], BIND [31], and Unbound [32]. We chose PowerDNS for its flexible backend database support, which allows seamless integration with most web-based DNS management panels. Our BIND implementation serves as a secondary server and provides the `dig` and `delv` tools for testing server configurations. Unbound offers *libunbound*, a library that can be easily integrated into other projects to facilitate DNS resolution and signature validation.

HTTPREQ record resolution was added to Chromium by integrating the *libunbound* library, as Chromium does not natively support DNSSEC verification and DoH does not suffice. We create a single shared *libunbound* context in Chromium's network process, configured with the default and any user-specified DNSSEC trust anchors. When the presence of a HTTPREQ record needs to be checked, the connection instantiator uses this *libunbound* context to resolve the record and validate its signature.

V. Evaluation

We verified that HSTS-Enforced improves web security as intended and typically induces no overhead. However, upon attempting to visit an unsecure website, the DNSSEC

resolution and validation process can slightly prolong the connection setup. We detail our experiments hereafter.

A. Setup

We deployed web servers that support HTTP, trusted HTTPS, or untrusted HTTPS, as well as recursive and authoritative DNS servers by deploying them in Docker containers. Our modified Chromium served as a user agent that ran directly on our system and connected to the servers inside the Docker containers. We simulated an RTT (Round-Trip Time) of 20 ms between the user agent and the recursive DNS server functioning as their external resolver by adding 10 ms delay in each direction using netem. Moreover, we registered a public domain, created and registered the required DNSSEC keys, requested a trusted HTTPS certificate, and self-signed another certificate to be able to test all aspects of HSTS-Enforced.

B. Security and Effectiveness

We verified that *HSTS-Enforced correctly prevents TLS stripping attacks* by blocking connections that use HTTP or untrusted HTTPS when no valid HTTP-Required indicator is present. It allows connections to such servers if and only if an indicator is present and always allows connections to trusted HTTPS servers. The mechanism also respects user preferences: if permitted through an indicator, it employs HTTP when explicitly specified through the scheme. Furthermore, when an indicator is present and the website is only served over a single connection type, a web client using HSTS-Enforced consistently uses that connection type, irrespective of the specified scheme.

C. Connection Delay

In most cases, *HSTS-Enforced does not induce additional delay*. It only adds some delay when verifying an HTTPREQ indicator because the DNSSEC resolution can take one or more RTTs. *This delay only occurs when one attempts to initiate an unsecure connection to a website that supports such connections and the domain has an HTTPREQ record*. It is the result of a trade-off between fast access to an unsecure website and protecting against TLS stripping attacks. In the case of a negative response for an HTTPREQ record or any DNSSEC keys, we assume no indicator to be present. Fig. 4 illustrates how many RTTs the resolution takes depending on which records must be retrieved and the actual times it took in our measurements, which are slightly longer than multiples of the RTT due to processing time. In the worst case, the resolution could take six RTTs – in our measurements, the worst case took 134 ms with an RTT of 20 ms. This worst case should rarely occur in actuality, because most of the records that are part of the DNSSEC chain are cached locally from previous requests. The likelihood that a record is cached locally increases for records higher up in the chain. Furthermore, DS and DNSKEY pairs are always used in tandem. If one of them is cached, usually both are. In a typical scenario, the DNSKEY for root is already cached and so are the DS and DNSKEY for the TLD (top-level domain). Hence, the typical verification process when visiting an unsecure website the first time should take around three RTTs to the resolver. Additionally, because DNS response packets are small (a few hundred bytes at most), caching many of them is inexpensive.

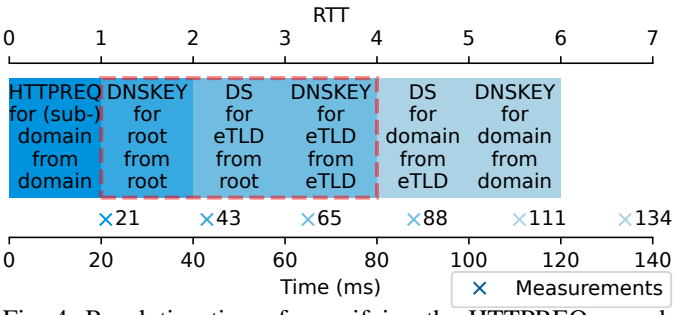

Fig. 4: Resolution times for verifying the HTTPREQ record, depending on which records are not cached and must be requested. Equal colors denote linked records, which are typically both cached if one of them is. The records outlined by the dashed red line are typically already cached due to previous resolutions.

VI. Transition Strategy

We propose a phased transition to HSTS-Enforced that allows website operators time to adjust before strict enforcement. During this period, operators of websites that support only HTTP connections must deploy one of the HTTP-Required indicators or upgrade their site to HTTPS. The transition could be supported by a service enabling operators to verify required actions for their sites. Additionally, a user-assisted service could, through browser extensions, gather a list of websites needing configuration updates, allowing a central entity to reach out to their operators.

After a transitional period, which should be long enough for operators to implement the necessary measures, HSTS-Enforced can be rolled out via browser updates. We anticipate its adoption extending beyond web browsers to all web clients that support HTTP. This includes command-line tools (e.g., wget and curl) and web request libraries (e.g., Python's requests, Java's HttpClient, Node.js axios, and Go's net/http), all of which are susceptible to TLS stripping when an incorrect scheme is specified, because prevailing implementations prioritize the user's explicit selection of HTTP over secure defaults.

VII. Discussion

Adoption of DNSSEC is not a significant limitation for HSTS-Enforced. According to a recent measurement [26], approximately 92.1% of web visits are served over HTTPS, making HSTS enforcement safe and advantageous for most sites. DNSSEC only has to be enabled for those websites that can not adopt HTTPS and want to deploy the DNS based indicator. For most of these websites, DNSSEC is a viable option to deploy: all 2820 ICANN-accredited registrars [33] and 93.7% of the TLDs present in the root zone support DNSSEC [34]. In total, there are 91 TLDs that do not support DNSSEC. These TLDs mostly host only government or organizational sites that likely support HTTPS. If deployment of HTTPS and this indicator is unfeasible, operators can fall back to another indicator such as the HTTP-Required Preload list.

The length of the HTTP-Required Preload list is anticipated to be significantly smaller than the HSTS Preload list. The HTTP-Required Preload list would primarily cover the limited

set of non-local, non-DNSSEC domains that have a hard requirement for HTTP. Given that these are relatively unpopular TLDs and only 2.4% of web visits rely on HTTP [26], we anticipate that the HTTP-Required Preload list may be significantly smaller than the HSTS Preload list.

VIII. Future work

Governance of the HTTP-Required Preload list could be decentralized or blockchain-based which would add transparency but also introduce operational complexity. Therefore, governance by a neutral instance such as a standards organization or non-profit Internet governance body with secondary verification by independent entities such as browser developers is desirable.

Alternative HTTP-Required indicators could be added; however, these must fulfill the requirements outlined in §III-B. During the development of HSTS-Enforced, we evaluated several additional candidate indicators, such as signed HTTP headers, opt-in HTTPS headers, or an X.509 certificate extension. The first two of these indicators fail because a web server can vary them per user, enabling potential tracking. X.509 certificate extensions fulfill the requirements; however, it requires an HTTPS server to serve the X.509 certificate, which is often missing when HTTP is needed, making deployment impractical.

IX. Conclusion

We have described HSTS-Enforced, a mechanism where web connections are secured by default, and operators can use simple yet robust methods to opt out of security if they require HTTP. We proposed two HTTP-Required indicators to enable this opt-out while leaving room for additional indicators. As shown, HSTS-Enforced secures web connections against attacks that remain possible with the current system around HSTS with limited overhead. We hope that HSTS-Enforced will be adopted as the next step in securing the web.

Acknowledgements

We thank the anonymous reviewers and Chenxing Ji for their comments, and we acknowledge the open-source tools and frameworks used in this work, whose continued development and maintenance supported the implementation and evaluation of the proposed mechanism. This research was supported by the National Growth Fund through the Dutch 6G flagship project "Future Network Services" and the Netherlands Organisation for Scientific Research (NWO) under the CATRIN project.

References

[1] E. Rescorla, "HTTP Over TLS," RFC 2818, 2000.
[2] J. Hodges, C. Jackson, and A. Barth, "HTTP Strict Transport Security (HSTS)," RFC 6797, 2012.
[3] F. K. Aaron van Diepen, Adrian Zapletal, "Securing the Web with HSTS-Enforced," *arXiv preprint*, 2026.
[4] M. Marlinspike, "New Tricks for Defeating SSL in Practice," Black Hat DC. https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf, 2009.
[5] ——, "More Tricks for Defeating SSL in Practice," Black Hat USA. https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf, 2009.
[6] The Chromium Authors, "HSTS Preload List Submission," HSTS Preload List Submission. https://hstspreload.org/, 2012.
[7] B. M. Schwartz, M. Bishop, and E. Nygren, "Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records," RFC 9460, 2023.
[8] Mozilla Support Community, "HTTPS-First Upgrades to Secure Connections," Mozilla Support. https://support.mozilla.org/en-US/kb/https-first, 2025.
[9] ——, "HTTPS-Only Mode in Firefox," Mozilla Support. https://support.mozilla.org/en-US/kb/https-only-prefs, 2025.
[10] J. Amann *et al.*, "Mission Accomplished? HTTPS Security after DigiNotar," in *ACM IMC*, 2017.
[11] I. S. Petrov *et al.*, "Measuring the Rapid Growth of HSTS and HPKP Deployments," in *IET CANS*, 2017.
[12] S. Roth *et al.*, "The Security Lottery: Measuring Client-Side Web Security Inconsistencies," in *USENIX Security*, 2022.
[13] H. Siewert, M. Kretschmer, M. Niemietz, and J. Somorovsky, "On the Security of Parsing Security-Relevant HTTP Headers in Modern Browsers," in *IEEE S&P Workshops*, 2022.
[14] W. Buchanan, S. Helme, and A. Woodward, "Analysis of the Adoption of Security Headers in HTTP," *IET Information Security*, vol. 12, no. 2, 2017.
[15] A. Lavrenovs and F. J. R. Melón, "HTTP Security Headers Analysis of Top One Million Websites," in *International Conference on Cyber Conflict (CyCon)*, 2018.
[16] A. Mendoza, P. Chinprutthiwong, and G. Gu, "Uncovering HTTP Header Inconsistencies and the Impact on Desktop/Mobile Websites," in *ACM WWW*, 2018.
[17] S. de los Santos and J. Torres, "Analysing HSTS and HPKP Implementation in Both Browsers and Servers," *IET Information Security*, vol. 12, no. 4, 2018.
[18] P. Syverson and M. Traudt, "HSTS Supports Targeted Surveillance," in *USENIX FOCI*, 2018.
[19] The Chromium Projects, "Chromium," Chromium. https://www.chromium.org/home/, 2024.
[20] Mozilla, "Firefox Preload List," https://searchfox.org/mozilla-central/source/security/manager/ssl/nsSTSPreloadList.inc, 2024.
[21] T. Chung *et al.*, "A Longitudinal, End-to-End View of the DNSSEC Ecosystem," in *USENIX Security*, 2017.
[22] T. Dai, H. Shulman, and M. Waidner, "DNSSEC Misconfigurations in Popular Domains," in *CANS*, 2016.
[23] H. Shulman and M. Waidner, "One key to sign them all considered vulnerable: Evaluation of DNSSEC in the internet," in *USENIX NSDI*, 2017.
[24] Electronic Frontier Foundation (EFF), "HTTPS Everywhere," Electronic Frontier Foundation. https://www.eff.org/https-everywhere, 2010.
[25] J. Aas, D. McCarney, and R. Shoemaker, "Multi-Perspective Validation Improves Domain Validation Security," Let's Encrypt. https://letsencrypt.org/2020/02/19/multi-perspective-validation.html, 2020.
[26] C. Kerschbaumer, F. Braun, S. Friedberger, and M. Jürgens, "The State of HTTPS Adoption on the Web," in *MADWeb*, 2025.
[27] A. van Diepen, A. Zapletal, and F. Kuipers, "HSTS-Enforced Artifacts," https://artifacts.aaronvdiepen.nl/HSTS-Enforced, 2026.
[28] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," in *Institute of Radio Engineers*, 1952.
[29] E. Fredkin, "Trie Memory," *Communications of the ACM*, vol. 3, no. 9, 1960.
[30] PowerDNS Team, "PowerDNS," PowerDNS. https://www.powerdns.com/, 2024.
[31] Internet Systems Consortium (ICS), "BIND," Internet Systems Consortium. https://www.isc.org/bind/, 2024.
[32] NLnet Labs, "Unbound," NLnet Labs. https://unbound.docs.nlnetlabs.nl/en/latest/, 2024.
[33] Internet Corporation for Assigned Names and Numbers (ICANN), "List of Accredited Registrars," https://www.icann.org/en/accredited-registrars?view-all=true, 2024.
[34] Internet Assigned Numbers Authority (IANA), "Root Zone File," InterNIC. https://www.internic.net/domain/root.zone, 2024.