The State of https Adoption on the Web

Christoph Kerschbaumer, Frederik Braun, Simon Friedberger, Malte Jürgens
Mozilla Corporation
{ckerschb,fbraun,sfriedberger,mjurgens}@mozilla.com

Abstract—**The web was originally developed in an attempt to allow scientists from around the world to share information efficiently. As the web evolved, the threat model for the web evolved as well. While it was probably acceptable for research to be freely shared with the world, current use cases like online shopping, media consumption or private messaging require stronger security safeguards which ensure that network attackers are not able to view, steal, or even tamper with the transmitted data. Unfortunately the Hypertext Transfer Protocol (http) does not provide any of these required security guarantees.**

The Hypertext Transfer Protocol Secure (https) on the other hand allows carrying http over the Transport Layer Security (TLS) protocol and in turn fixes these security shortcomings of http by creating a secure and encrypted connection between the browser and the website. While the majority of websites support https nowadays, https remains an opt-in mechanism that not everyone perceives as necessary or affordable.

In this paper we evaluate the state of https adoption on the web. We survey different mechanisms which allow upgrading connections from http to https, and provide real world browsing data from over 140 million Firefox release users. We provide numbers showcasing https adoption in different geographical regions as well as on different operating systems and highlight the effectiveness of the different upgrading mechanisms. In the end, we can use this analysis to make actionable suggestions to further improve https adoption on the web.

I. Introduction

Over 30 years ago, on August 6th 1991, Tim Berner's Lee published a blog post sharing information about his World Wide Web project [47]. With the words *"Try it"* he encouraged the world to make use of the web. Since then billions have and for most of us, the web has become an integral part of our daily lives.

Since its inception, the fundamental protocol through which web browsers and web servers communicate is http [13]. However, data transferred by the regular http protocol is unprotected and transferred in the clear, granting a network attacker the ability to view or even modify it. While in the early days of the web this security drawback probably was an acceptable trade-off, the web has started to host not only considerably more content but also more sensitive user data. Eventually the web was not only used by scientists to share information, but the web started to host online shopping sites where end users entered their credit card numbers. To avoid fraud, these numbers had to be transferred securely. Nowadays, not only credit card numbers, but literally all kinds of sensitive data is transmitted over the wire and needs to remain confidential from third parties.

Creating a secure and encrypted connection between the browser and the web server provides the necessary security guarantees and overcomes the security issues of http. More precisely, carrying http over the Transport Layer Security (TLS) protocol [14] enables the browser to authenticate the identity of the web server to the browser, and ensures that messages sent between the browser and the server are kept confidential from all other parties.

Over the years we have witnessed tremendous progress towards migrating the web to use https by default instead of the outdated and insecure http protocol. Browser vendors as well as the web community have initiated various efforts to increase the ratio of websites delivering content over https connections. Most notably, HTTP Strict Transport Security (HSTS) [15] allows a server to signal to a browser to always rely on https connections. Also, the Secure Contexts specification [51] which states to only expose powerful web APIs to webpages loaded over https, has helped put pressure on site operators resulting in increased https connections on the web.

Another very important milestone towards increasing the availability of https was the launch of the *Let's Encrypt* initiative [22]. Let's Encrypt was the first global Certificate Authority (CA) service that allowed website owners to automatically obtain free browser-trusted certificates, needed to provide https connections for their websites. Presumably, search engines boosting a site's rank when available over https [11] also had a positive effect.

Furthermore, browser vendors have started to build various customized flavors of upgrading mechanisms directly into their engines. Chromium, the rendering engine powering Chrome, Brave, Edge and more, Firefox with its underlying engine Gecko, as well as Safari on top of Webkit have started to deploy upgrading mechanisms [3], [20], [28], [59]. All of these efforts highlight the importance for the web community to continuously deliver more content over a secure and encrypted connection.

Generally a web page is transferred by first, a top-level document load, followed by various sub-resource loads. A

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2025
28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-9-0
https://dx.doi.org/10.14722/madweb.2025.23001
www.ndss-symposium.org

top-level load, or also document load or page load, refers to the request of a top-level `HTML` document. Sub-resource loads are all the resources (images, stylesheets, scripts, iframes, etc.) contained within the top-level document which together provide the content for the web page. Loading both top-level documents and sub-resources over `https` is equally important.

For this work we solely focus on top-level document loads and on upgrading those. Please note, that when a top-level document loads over `https` then all major browsers (Chrome, Edge, Firefox, Safari) either upgrade insecure sub-resources, like images, audio and video, or block insecure sub-resources, like scripts and iframes. This behavior is defined in the latest version of the Mixed Content Specification [48] which ensures there is no `http` content within a top-level `https` document anymore. This new behavior of either upgrading or blocking mixed content eliminates the problem of exposing sub-resources to an active network attacker as long as the top-level document is loaded over `https`, thus putting even more importance on the security of the top-level document load.

In this paper we survey the state of `https` connections on the web and also highlight the different upgrading mechanisms available on the server-side as well as on the client-side. On the client-side we focus our survey on the web browser Firefox specifically, due to our access to Firefox telemetry data. In detail, we contribute the following:

- In Section II we survey the different upgrading mechanisms available on the server side and provide insights into current client side upgrading mechanisms within the Firefox browser (version 134).
- In Section III we provide the current state of `https` adoption on the web by evaluating real world data reported by over 140 million Firefox release users. We provide insights into `https` adoption for different geographical regions and operating systems, and we examine the effectiveness of the various upgrading mechanisms.
- In Section IV we provide a discussion around our gathered results as well as an outlook of having the entire web rely exclusively on `https` connections in the future.

II. A Survey of Upgrading Mechanisms for Top-Level Loads

Over the past two decades various mechanisms have been proposed to favor `https` connections whenever possible. Within this section we survey the different upgrading mechanisms and highlight their effectiveness to upgrade top-level document loads from `http` to `https`. Again, a top-level load is the connection used to fetch the main `HTML` document for the website you are trying to visit. Put simply, a top-level load is the load of the URL that the browser displays in the address bar.

Before we analyze the different upgrading mechanisms and highlight their benefits and trade-offs it is important to distinguish between upgrading mechanisms initiated on the server-side by the web server and those which are available on the client-side in the browser (See Table I).

Upgrading Mechanism	*Server-Side*	*Client-Side*
CSP `upgrade-insecure-requests`	●	
HTTP Strict Transport Security (HSTS)	●	
HTTPS Resource Records (HTTPS RR)	●	
Web Extensions		●
HTTPS-Only Mode (Firefox built-in)		●
HTTPS-First Mode (Firefox built-in)		●

TABLE I: A comparison of different upgrading mechanisms available on the server and client side - in particular upgrading mechanisms available on the client in Firefox (version 134).

While implementations of server-side upgrading mechanisms are basically identical in all browsers, built-in client-side upgrading mechanisms differ in their implementations and mechanics. This is due to server-side upgrading mechanisms following standards from the World Wide Web Consortium (W3C) [50] or the Internet Engineering Task Force (IETF) [18] while client-side upgrading mechanisms are not standardized yet. The fact that server-side upgrading mechanisms range from DNS, to HTTP and HTML showcases how many protocol layers contribute to establishing a secure connection.

For client-side upgrading mechanisms our work focuses on the technical details of the different built-in upgrading mechanisms available within Firefox. We decided to focus on Firefox for the available client-side upgrading mechanisms because we have access to telemetry data reported by real world Firefox release users and can therefore measure, cross-reference, and evaluate their effectiveness later within Section III.

A. The `upgrade-insecure-requests` directive in a Content-Security-Policy

The Content Security Policy (CSP) [49] provides an added layer of security for web pages that helps to detect and mitigate certain types of attacks. While CSP is mostly known for its protection against Cross-Site-Scripting (XSS) and other injection attacks, the `upgrade-insecure-requests` directive [46] provides a security feature that can ensure that all insecure `http` requests are automatically upgraded to secure `https` requests within a website.

To enable CSP, a web page needs to configure their web server to return the `Content-Security-Policy` response header in server responses, similar to the following:

```
Content-Security-Policy: upgrade-insecure-requests
```

Alternatively, a website can deliver a policy leveraging the `<meta>` element in the HTML markup of a website.

Mostly, a web page would make use of this directive to upgrade insecure sub-resources such as images, scripts, or stylesheets, to secure requests. It is worth noting, that this directive instructs the browser to brute force upgrades for all sub-resources, regardless of their origin within the current document. Hence, using this directive can cause sub-resources on a website to result in broken loads if the upgrade from `http` to `https` does not succeed.

More importantly in the context of this paper, the `upgrade-insecure-requests` directive also applies to same-origin top-level navigations and those which submit form data [45]. In more detail, imagine you visit the website `https://example.com`, and then you click a link on that web-page, that is same-origin with `http://example.com`, modulo the scheme portion of the URL, then the browser will upgrade the connection for this top-level load from `http` to `https`. Websites utilizing this Content-Security-Policy directive can hence ensure that the browser will automatically upgrade `http` links on their site to `https` and thus provide an additional layer of security against machine-in-the-middle attacks on their sites [1], [5], [8], [40], [41].

Another interesting property of this feature is that CSP works on a per-request, per-document basis. HSTS, on the other hand, is a setting for the whole domain. This makes the `upgrade-insecure-requests` directive a useful stepping stone towards a site-wide HSTS deployment.

B. HTTP Strict Transport Security (HSTS)

HTTP Strict Transport Security (HSTS) [15] allows a website to signal that browsers should only interact with the website using secure `https` connections and never with insecure `http` connections. For HSTS-enabled websites, the browser will always establish a secure and encrypted `https` connection even when following a non-secure `http` URL.

A website implements an HSTS policy by sending the `Strict-Transport-Security` response header in server responses for `https` connections, similar to the following:

```
Strict-Transport-Security: max-age=<expire-time>;
     includeSubDomains
```

The presence of the header indicates that the browser should automatically upgrade `http` links to resources on the site to the corresponding `https` links. For HSTS to work, the header needs to contain a duration (`max-age`) in seconds for how long the browser should remember this setting and may include additional optional keywords. Such keywords for example allow specifying whether the setting should apply across all subdomains (by using `includeSubDomains`) in addition to the main domain.

When the browser receives an HSTS header over `https`, it caches that signal to upgrade all future requests to that website to `https`. This allows the browser to automatically turn any non-secure `http` link for a website into a secure `https` link. For example, suppose that the site `exampleB.com` deploys an HSTS header, and the browser has previously visited `exampleB.com` over `https`. Then suppose that `https://exampleA.com/index.html` contains a link to `http://exampleB.com/index.html`. Because the browser has previously visited `exampleB.com` and cached the HSTS header, it will load the page over `https` despite the `http` link.

A downside of HSTS is that browsers will ignore the header if it is received over non-secure `http`. Hence, web servers utilizing HSTS must first redirect the non-secure `http` request and upgrade it to a secure `https` request (e.g. with configurations that redirect all `http` requests to their `https` equivalents). This requirement showcases a design limitation: before a browser has visited a website and received HSTS information, a request may still occur using plain `http` and is therefore still vulnerable to downgrade attacks by tools such as SSLStrip [25]. Hijacking this initial request suffices for an attacker to perform a machine-in-the-middle attack, which in turn allows them to downgrade the connection and eavesdrop on, or modify data exchanged between client and server.

To prevent this kind of downgrade attack, modern browsers introduced a mechanism known as the `HSTS Preload List` - a static list of HSTS supporting domains that is integrated and shipped with the browser [4], [26]. A website can submit an inclusion request to the `HSTS Preload List`. If accepted, the website's domain gets added to the list and browsers utilizing the list will exclusively make secure `https` connections to it. Currently, this list contains slightly over 106,000 entries as inspected in the Firefox source code [32]. Obviously such a list based approach cannot scale to the size of the internet which consists of billions of sites. However, given that most of the web traffic on the internet occurs on a small number of popular sites [39], this mechanism still enables top sites to protect their end-users against downgrade and machine-in-the-middle attacks.

C. HTTPS Resource Records (HTTPS RR)

Traditional methods of establishing secure connections, like HSTS, often necessitate multiple network round trips between the client and the server, resulting in latency. Therefore, engineers in the IETF started developing a solution based on the Domain Name System (DNS) system. HTTPS Resource Records (or HTTPS RR [17]) is a DNS record type designed to facilitate the discovery and connection to `https`-enabled services before the first connection. Providing the additional metadata directly within the DNS can reduce the number of round trips required for a secure connection, as the browser can consider the upgrade before establishing the actual connection.

A site administrator can set up HTTPS records for their domain `example.com` as follows:

```
www.example.com 1800 IN HTTPS 1 . port=8002 alpn=h3
```

where `www.example.com` is the domain name, `1800` is the Time to Live (TTL) for caching, and `https` specifies the record type.

There are multiple extra fields in the DNS record that we will not explain in full detail. Suffice to say that it allows to direct traffic to other host names, ports and also to immediately switch to different `HTTP` protocol levels (like HTTP/2, HTTP/3) using the TLS Application-Layer Protocol Negotiation Extension (ALPN) [16].

Therefore, a browser requesting the HTTPS record from the DNS server receives all necessary information to establish an encrypted and fast connection to `https://example.com` by removing a lot of additional overhead that is otherwise required within the upper layers of the `http` protocol to negotiate an encrypted session. By supporting ALPN directly in HTTPS RR, the browser is able to avoid further overhead that is usually spent on negotiating the HTTP protocol version by jumping straight to the HTTP/2 or HTTP/3 endpoints.

Like HSTS, this feature provides a control mechanism that applies to the whole site and therefore requires all content to be available over `https`. Notably different from HSTS, the HTTPS Resource Record requires changes in the HTTP stack as well as the DNS code of the client. Whereas HSTS relies on an attacker not being able to attack the first request/response pair, HTTPS RR also requires the DNS traffic to be secure for the browser to receive accurate information about the server's `https` configuration.

D. Upgrades from Web Extensions

A web extension [23] allows adding features and functions to a web browser and hence allows enriching the browsing experience of end-users. Generally web extensions are also created using familiar web-based technologies, such as HTML, CSS, and JavaScript. A web extension can further take advantage of a set of privileged JavaScript APIs provided by the browser which allow it to intercept and modify outgoing network requests and therefore enables it to upgrade a connection from `http` to `https`.

The list of web extensions that intend to secure web traffic is long. The most prominent browser extensions for upgrading connections is likely *HTTPS Everywhere* [6], which allows browsers to encrypt communications with many major websites. Extensions vary in how they balance strictness and usability. While some extensions completely disallow `http` requests and oftentimes simply block them, others warn or upgrade requests opportunistically without any kind of extra indication. Further, some extensions remember sites that are incapable of serving `https` requests while others try to upgrade every time.

For a while there have been extensions that reported 'seen certificates' for visited websites and therefore provided a service that allowed others to verify which certificates are legitimate when making a secure connection. These extensions would warn users if a website changed its certificates or when it provided different certificates to different end users. [35], [60]. While this approach worked well for long-lived certificates, the shift towards shorter validity periods [42] and trends in Cloud and Edge Computing have made these approaches less reliable.

In contrast to the other mechanisms discussed here, site authors have little to no influence over whether their users will manually go through the steps of installing a web extension. Given that most browser users are not security experts, the web extension approach by itself is already limited to people who have the required security background, and are further aware that these kinds of extensions exist in the first place.

Additionally, we can see that the popularity of these extensions has waned in the light of other mechanisms becoming available. In fact, the development of the *HTTPS Everywhere* extension from the Electronic Frontier Foundation (EFF) has been paused in 2021, as a result of upcoming browser architecture changes. The authors cite the ubiquity of `https` enforcement mechanisms in various browsers, like *HTTPS-Only Mode* in Firefox [20], *Automatic HTTPS* in Microsoft Edge [24] or *HTTPS-First* in Google Chrome [3], [7].

E. HTTPS-Only Mode (Firefox built-in)

HTTPS-Only Mode [20] is an opt-in security feature in Firefox. Chrome has a similar setting "Always use secure connections" [10], while Safari requires one of the previously mentioned extensions.

HTTPS-Only Mode upgrades all connections, whether top-level or sub-resources loads, by rewriting the scheme portion of a URL from `http` to `https`. If the browser cannot establish a top-level `https` connection, e.g. because the web server does not support `https`, the end user gets prompted with an error message which explains the security risk. The user then has the choice to either abandon the page visit or provide explicit consent to grant Firefox the permission to visit the site using an `http` connection.

For sub-resource loads, HTTPS-Only Mode enforces a "brute force" approach - every connection for a sub-resource load gets upgraded. If the upgraded load does not succeed, there is no fallback mechanism. Measurements in previous work [20] indicate that if a top-level connection can be established using `https`, then it is extremely likely that sub-resources within that page are also available over `https`.

While HTTPS-Only ensures that only `https` connections are used, it potentially downgrades the browsing experience of a regular end user. Because of its strictness it finds wide usage with users who put a high value on privacy and security and, due to the shared codebase, in the Tor Browser [54], however it's likely not suitable for typical users. In fact, it changes the default behavior in a web browser, thereby affecting the expectations of web developers that their content is behaving according to widely accepted standards. Such behavior changes can lead to websites working improperly in one browser while working fine in another. This kind of inconsistency is captured under the term "web compatibility", or "webcompat" for short. Finding the right balance between security, user experience and web compatibility requires additional efforts in research and development to roll such features out by default to all users in release versions of browsers.

F. HTTPS-First Mode (Firefox built-in)

Eventually, Firefox introduced HTTPS-First Mode to improve upon HTTPS-Only Mode. Internally HTTPS-First Mode relies on the same security principles as HTTPS-Only Mode, though it does not prompt the user for explicit permission to visit an `http` URL in case the automatic upgrade fails. Instead, the underlying algorithm of HTTPS-First Mode allows the connection to automatically fall back to `http` for cases where a secure and encrypted connection cannot be established. Additionally, and as previously mentioned, the new security guarantees provided by the Mixed Content Protection [48] only requires HTTPS-First Mode to act upon top-level loads and leave the upgrading or blocking of sub-resources to the implementation of the Mixed Content Specification.

An internal evaluation has shown that end users care about their security, though they care slightly more about an uninterrupted browsing experience. Only about two million Firefox users have ever opt-ed into enabling HTTPS-Only Mode. Of those, about a third have decided to disable it again. This shows that a smooth browsing experience is preferable over warning dialogs. And, a probably even more important insight, that end users need built-in security by default. They do not want to adjust the security settings of their browser since most of them have little or no expertise in security.

HTTPS-First Mode provides that usable security feature. With its built-in fallback mechanisms HTTPS-First seems to be more in line with what end users expect from a modern browser. This opportunistic approach of course does not provide as strong security guarantees as HTTPS-Only Mode, however it allows favoring secure and encrypted connections whenever possible. Hence, the automatic upgrade with built-in fallback mechanism that HTTPS-First provides seems like the perfect balance between security and usability for the vast majority of end users. Please note, that security conscious users can still opt into enabling HTTPS-Only Mode which overrules the settings of HTTPS-First Mode thus providing a mechanism for end users to ensure that not a single web page will be loaded using `http` without their explicit consent.

Firefox has been shipping HTTPS-First by default in Private Browsing Mode [28] since version 91. We expect that this feature requires fine tuning to cater to the long tail of the web (cf. web compatibility) before it will be released in early 2025. In that notion, a subset of the functionality provided by HTTPS-First is already generally enabled in release versions of Firefox. For now general navigations remain unaffected in this first wave of a progressive rollout of HTTPS-First Mode, but loads that originate from the address bar already benefit from the security advantages provided by HTTPS-First.

G. HTTPS Upgrades Proposal

Due to the shared interest in introducing additional encryption upgrades, engineers from multiple browsers are working together to standardize their behaviors for upgrading page loads to `https` as part of the WHATWG Fetch Standard [43]. The main discussion forum is a currently unmerged pull request with the name "HTTPS Upgrades" [36]. The proposed change aims to bring closer alignment among browser vendors. It originally defined a behavior similar to HTTPS-First Mode in Firefox, in that it attempts to load a page via secure `https` first and falls back to an unencrypted `http` request when a secure connection cannot be established. There were subtle differences in that HTTPS-Upgrade would apply to all `http` URLs whereas HTTPS-First would only upgrade navigations on their default ports, i.e., switching `http` navigations on port 80 to `https` on port 443 (the respective default ports).

Furthermore, Firefox would respect pages that redirect from `https` to `http` and no longer attempt to upgrade them, in order to not cause endless redirect loops. Recent changes to the proposal have led to much closer alignment. At this point, we can consider the term HTTPS-First for the Firefox implementation and HTTPS-Upgrades for the proposed feature in WHATWG Fetch as two sides of the same coin. While we can expect that the product-specific names in their respective settings may live on, we can assume that the specification and standardization efforts has led to full alignment across browsers and will likely result in converging behavior and increased web compatibility in the future.

III. Evaluation

To provide an accurate picture of the state of `https` adoption on the web we break our evaluation into four subsections. We start by providing measurements of the evolution of `https` adoption over the last ten years, from 2014 to 2024 (Subsection III-B). We then provide insights into `https` adoption in different geographical regions (Subsection III-C) and on different operating systems (Subsection III-D). Lastly, we provide insights into the effectiveness of the different upgrading mechanisms on the web (Subsection III-E). Before however, we briefly provide background information on our data-gathering mechanisms (Subsection III-A) used to provide a stable line of comparison.

A. Background on Data-Gathering

The Firefox browser for Desktop (version 134) is available for Windows, running Windows 10 or later supporting 32-bit or 64-bit, for Mac running macOS 10.15 or later, and for GNU/Linux using glibc 2.17, GTK+ 3.14, libglib 2.42, and libstdc++ 4.8.1 or higher versions thereof. Also, the Firefox browser is available for Android smartphones and tablet devices running Android 5.0 with API level 21 or higher.

The *Mozilla Telemetry Portal* [33] lets us present information gathered during one full Firefox release cycle (version 134), from **January 7th to February 3rd, 2025**. All the collected information used for analysis is subject to *Mozilla's Data Privacy Principles* [27], which only permits collection of non-user-specific data.

Our evaluation period of one month covers **real world browsing data of over 140 million Firefox release users** who agreed to report information back to Mozilla.

B. `https` Adoption on the Web Over the last Decade

Note: *This subsection presents historic data collected as part of Firefox telemetry [22]. For our research we precisely defined what data to collect and installed new telemetry probes. When doing so, we encountered that the data presented in this subsection was historically collected in different situations in regard to error handling, iframes, caching, redirect handling and HTTP methods. While the chosen collection method results in underreporting* `https` *usage compared to our newly installed probes in the other sections, it still captures the general trend of* `https` *adoption over the last decade.*

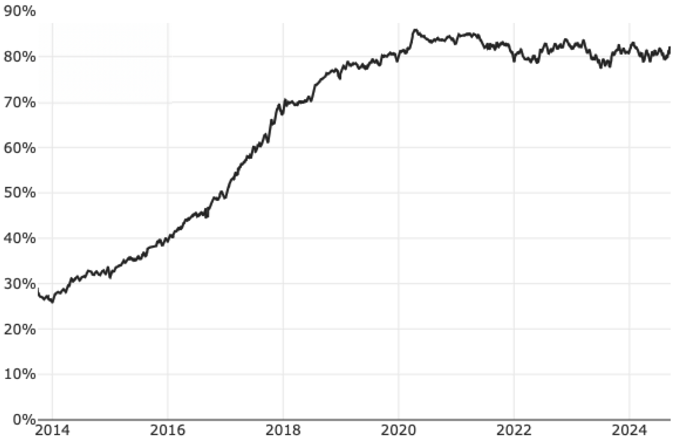

Fig. 1: Evolution of websites relying on `https` for the years 2014 to 2024. (Note: Data is also basis of *Let's Encrypt*'s statistics. [22]).

In 2014, ten years ago, under 30% of web pages were loaded using `https` (Figure 1), leaving more than two thirds of websites vulnerable to machine-in-the-middle attacks, and hence putting user's security and privacy at risk.

In the following years the graph in Figure 1 for `https` supporting websites points steeply upwards. During those years *Let's Encrypt* launched. Again, *Let's Encrypt* is an automated certificate authority that allows website owners to automatically obtain a browser-trusted certificate free of charge to secure their website traffic. Having the ability to get a free certificate for a website paired with an upgrading mechanism like HSTS [15] or also CSP's `upgrade-insecure-requests` directive [46] have certainly contributed to this success. Furthermore, browser vendors following the suggestions from the Secure Contexts specification [51] forced web developers to upgrade their websites security model to use the latest and most powerful web APIs to provide a rich user experience. Additionally, search engines considering `https` availability for search rankings [11] caused website operators to re-think their security model and potentially upgrade their website to support `https`. Whatever combination of the above-mentioned factors made the difference, fact is, the graph in Figure 1 shows a doubling of `https` adoption during the years 2016 to 2019.

Finally, we observe that between the years 2019 and 2024 the graph, and hence the increase of websites loading over `https`, starts to flatten. We reason that this flattening is because more people, also in lesser developed countries started to have access to the internet, which caused the proportion of websites loaded over `https` to flatten. But either way, we note that by the end of 2024, more than 80% of web pages were loaded using `https` (see Figure 1), almost tripling the percentage from 2014. The exact usage rate varies by region as we will discuss in more detail in Subsection III-C.

C. The State of `https` *Adoption in Different Geographical Regions*

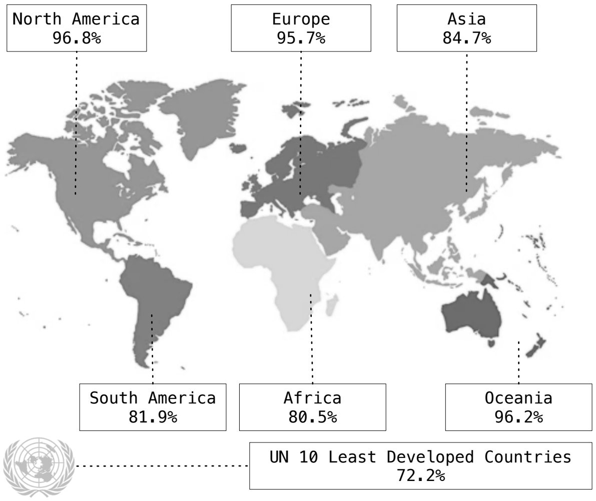

Fig. 2: The state of `https` adoption on different continents as well as for the ten least developed countries in the world.

Continent	*Country Name*	*Geogr. Code*	`https`
North America			**96.8%**
	Canada	CA	97.1%
	United States	US	96.4%
Europe			**95.7%**
	Austria	AT	94.9%
	Belgium	BE	97.0%
	Denmark	DK	98.0%
	France	FR	96.8%
	Germany	DE	95.9%
	Italy	IT	91.1%
	Netherlands	NL	96.1%
	Norway	NO	97.3%
	Portugal	PT	94.8%
	Spain	ES	95.1%
Asia			**84.7%**
	China	CN	77.6%
	India	IN	83.2%
	Indonesia	ID	71.5%
	Iran	IR	84.5%
	Japan	JP	93.8%
	Pakistan	PK	87.4%
	Saudi Arabia	SA	86.8%
	South Korea	KR	91.2%
	Thailand	TH	84.2%
	Turkey	TR	86.9%
South America			**81.9%**
	Argentina	AR	90.0%
	Brazil	BR	87.1%
	Chile	CL	87.5%
	Colombia	CO	81.7%
	Costa Rica	CR	86.2%
	Ecuador	EC	88.2%
	Panama	PA	87.4%
	Paraguay	PY	62.5%
	Peru	PE	78.8%
	Uruguay	UY	66.9%
Africa			**80.5%**
	Algeria	DZ	92.3%
	Angola	AO	81.1%
	Egypt	EG	88.4%
	Kenya	KE	70.5%
	Morocco	MA	70.1%
	Nigeria	NG	83.6%
	Rep. of the Congo	CG	88.4%
	South Africa	ZA	86.8%
	Tunisia	TN	81.4%
	Uganda	UG	62.4%
Oceania			**96.2%**
	Australia	AU	95.2%
	New Zealand	NZ	97.1%
UN 10 Least Developed Countries			**72.2%**
	Bangladesh	BD	77.9%
	Burkina Faso	BF	94.9%
	Cambodia	KH	78.9%
	Ethiopia	ET	66.2%
	Haiti	HT	77.8%
	Mali	ML	66.1%
	Mozambique	MZ	78.0%
	Myanmar	MM	55.2%
	Sudan	SD	51.2%
	Yemen	YE	75.5%

TABLE II: Detailed list of `https` adoption on the different continents separated by selected countries plus `https` adoption for ten of the least developed countries in the world.

To provide an accurate picture of https adoption around the world, we provide numbers on https usage for the different continents. Depending on the continent we evaluate data of up to ten different countries. We cluster information together (see Figure 2) but also provide individual percentage numbers for all the inspected countries (see Table II).

Note: *We randomly selected the countries for each continent. While this selection may introduce a small bias, the presented numbers provide valuable insights about the range of https adoption around the globe.*

Note that we provide the country name and also the geographic code. The Firefox telemetry systems strip a lot of unused information during ingress, for various reasons. In particular, sender IP addresses are resolved into country codes using GeoIP. We provide these geographic codes for reproducibility purposes and because the Mozilla telemetry portal relies on geographic codes for querying specific regional data [34].

Further, we not only want to shed light on the usage of https individually for the global north and the global south, but also evaluate https adoption in countries which might not have the resources to encrypt all of their connections. We want to see if some countries are left behind and need to catch up, and evaluate if there is a correlation between lesser developed countries and https adoption. To accomplish that comparison, we select ten countries from the UN list of the least developed countries [56] and inspect https adoption in these countries specifically.

As illustrated in Figure 2, our measurements show that the global north consisting of North America, Europe and Oceania are able to encrypt over 95% of their top-level connections. In more detail, North America's HTTPS adoption is at 96.8%, Europe at 95.7% and Oceania at 96.2%. In contrast, the global south consisting of Asia, South America and Africa are roughly able to encrypt 82% of their top-level connections. More precisely, Asia 84.7%, South America 81.9% and Africa 80.5%.

This high-level comparison shows that there is roughly a 13% difference in https traffic between the global north and the global south.

When we look more closely at the country specific results from Table II we see that the top three countries for https adoption are Denmark with 98.0%, Norway, with 97.3% and then Canada or also New Zealand, both with 97.1%. On the other end of the spectrum we see that the three countries with the least encryption of top-level loads are Sudan with 51.2%, Myanmar with 55.2% and Uganda with 62.4%.

Comparing the top three countries with an average of 97.5% of https adoption with the three countries that have the least https adoption, namely on average 56.3% we see that there is roughly a 40% difference. Put differently, around six out of 10 web page loads in any of those three countries on the lower end of the https adoption spectrum are not encrypted, exposing network traffic to network attackers and hence putting an end user's security at risk.

Finally, when looking at the list of https adoption for the ten least developed countries in the world (see bottom of Table II), we see that on average only 72.2% of top-level connections rely on a secure and encrypted connection. Given that the top ten countries from the global north are encrypting close to 97% of their connections, we see that there is almost a 25% difference of https adoption between the top ten countries from the global north and the ten least developed countries in the world.

In closing, we can see that there is a correlation whether a country is from the global north or from the global south in regard to https adoption. While https adoption for countries located in the global north is getting closer and closer to 100%, we see that the least developed countries from the global south do not appear to encrypt as many top-level loads and may need further support to increase the security of their data transfer.

D. The State of https Adoption on Different Operating Systems

In this section we provide a comparison of https adoption for users on the different operating systems Windows, macOS, Linux and Android. The following data is reported from over 140 million Firefox release users from all over the world and is not segregated by region. In detail, during our data collection period of one month we recorded data from approximately 107 million Firefox users on Windows, seven million users on macOS, five million users on Linux and 25 million users on Android.

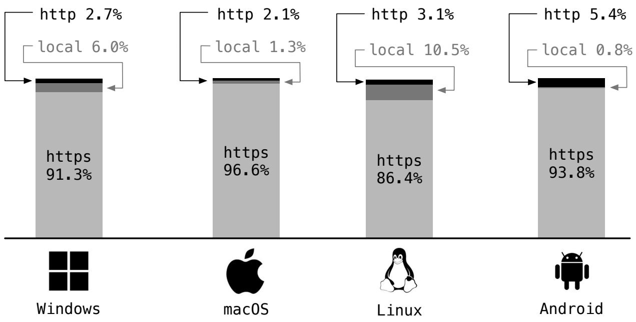

Fig. 3: Adoption of https on the different Operating Systems: Windows, macOS, Linux and Android.

Figure 3 provides aggregated percentage numbers for all top-level loads for the operating systems Windows, macOS, Linux and Android and illustrates how many top-level loads occur over https or http, whereas local is a subset of http, meant to highlight requests that are not as easily upgraded to https.

As illustrated, 91.3% of all top-level web traffic on *Windows* is transmitted over https. The operating system where Firefox manages to establish the highest percentage of secure connections is macOS, where 96.6% of all top-level loads occur over https. Finally, the mobile operating system Android allows Firefox to establish 93.8% of secure and encrypted https connections.

Linux only establishes 86.4% https connections, which at first sight indicates that Linux leaves the most web traffic vulnerable to network attackers. When taking a closer look however Figure 3 illustrates that Linux also establishes the most connections (10.4%) to sites in local IP ranges which are not as easily upgradeable. We reason that the Linux audience has a higher share of developers and system administrators and hence spend a lot of work in local IP ranges.

E. The Effect of `https` *Upgrading Mechanisms on the Web*

Within this section we provide a detailed analysis of all top-level loads that (a) already start out being `https`, (b) loads that the different upgrading mechanisms are able to upgrade, (c) `local http` loads that occur in the local IP range that cannot be upgraded and (d) loads that still occur over insecure and unencrypted `http` connections.

Please note that Firefox telemetry records and reports only the first mechanism that upgrades a request. Browsers implement the upgrading mechanisms in a specified order. Firefox gives precedence to upgrading as specified in the WHATWG Fetch specification [43] and only then applies its own improvements. Only if Firefox cannot detect any upgrading signal originating from the server, does it check, one-by-one, the internal upgrading mechanisms. In more detail, the upgrading order in Firefox is: HSTS, CSP `upgrade-insecure-requests`, HTTPS-RR, and if enabled, HTTPS-Only, HTTPS-First and finally web extension upgrades.

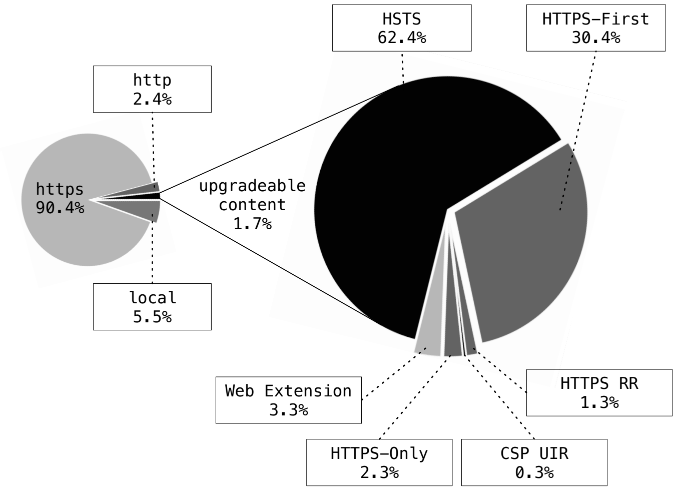

Fig. 4: Comparison for top-level (document) loads, relying on `https`, `http`, `local http` and the effectiveness of the various upgrading mechanisms.

Looking at the left pie chart in Figure 4, we see that 90.4% of all top-level loads recorded during our study already start out as `https`. In other words, there was no need for any of the different upgrading mechanisms to apply and improve the load process because these loads were specified to happen over a secure and encrypted `https` connection to begin with. These may be sites retrieved from the browsing history or bookmarks, or hyperlinks which point to URLs starting with `https`. Further, we found that 5.5% of top-level loads connect to a `local http` address which can not be upgraded because those loads happen in the local IP range. On the other end of the spectrum we found that 2.4% still end up happening on the outdated and insecure `http` protocol, because no upgrade mechanism succeeded.

That leaves 1.7% of top-level loads that are subject to the various upgrading mechanisms that can potentially upgrade a top-level connection from `http` to `https`. The pie chart on the right side of Figure 4 presents a breakdown of those 1.7% of upgraded content and shows which upgrading mechanisms are responsible for those upgrades. The data indicates that HSTS is the leading upgrading mechanism and accounts for about 62.4% of all loads that can be upgraded. Recall that the majority of web traffic happens on top sites, which are all on the `HSTS Preload List`.

HSTS is followed by Firefox's built-in upgrading mechanism HTTPS-First Mode. It is accountable for 30.4% of upgrades. HTTPS-First tries to upgrade a connection from `http` to `https` and also has a built-in fallback mechanism in case the upgrade does not succeed. Approximately 14% of loads which HTTPS-First tries to upgrade, fall back to `http`. In our chart, such a failed upgrade attempt shows up in the left pie chart under `http` loads.

Further, we see 3.3% of upgrades from `http` to `https` are due to web extensions that upgrade connections. 2.3% of upgrades are triggered by the opt-in HTTPS-Only mode. Finally, the data shows that HTTPS Resource Records in DNS and CSPs `upgrade-insecure-requests` directive play a smaller role in the importance of upgrading mechanisms by being responsible for 1.3% and 0.3% of loads respectively.

We want to emphasize that even though 1.7% of upgraded connections seems like a low percentage, it translates to an impressive absolute number of more than 1.4 billion top-level loads that can be upgraded from `http` to `https` for our measurement period and shows the importance of investing in upgrading mechanisms to secure as many loads as possible and keep end users protected from machine-in-the-middle attacks.

IV. Discussion and Outlook

As the data has shown, by now a large majority of the internet's communication is taking place over `https`. More importantly, the ratio of sites relying on `https` continues to grow. However, getting to a fully encrypted web, where every single connection is encrypted does not seem to be in sight quite yet.

While it would be technically feasible for a browser to only allow `https` connections and completely block all `http` traffic, this is not a reasonable step. For one, legacy websites that can only be reached over `http` would be blocked which would bury valuable content forever and non-publicly resolvable domain names such as `.arpa`, `router.local`, or `192.168.1.1` in a local domain could not be accessed either. Second, it would show an unreasonable amount of warnings to users which generally leads to those warnings being ignored.

Security is obviously important, and we should try hard to encrypt everything on the network but removing access to websites completely is undesired. The W3C even devotes a section named *Support Existing Content* in their *HTML Design Principles* [44] to this topic. In it, the authors explain that every design decision on the web should be carefully weighed against the likely cost of breaking sites.

A. A Fork in the Road of the Open Web

The web evolved over time and still holds millions of legacy websites that can only be reached over `http`. By no means should we cut the cord for all these websites. That would severely restrict the free flow of information - something that is widely recognized as unacceptable.

However, we need to start considering not allowing newly created websites to be accessed without a browser-trusted certificate. One option we could envision would be a *fork in the road approach*, which would divide websites that only support `http` into two groups: In the first group we have `http` websites created before a to-be-determined, fixed date. In the second group we have all `http` websites that were created after said date. A browser could then allow legacy `http` connections to websites from the first group but block connections to websites from the second group. The main issue with this approach is how to determine the creation time of a website.

In more detail, the browser connection model could be adapted to use the following approach: If the connection is already `https`, the browser proceeds and connects to load the website. If the connection, however, is `http`, the browser checks the date on which the domain was registered.

We could envision that the browser performs a look-up query to a service similar to a `WHOIS` database which stores information about the creation date. Alternatively, we could envision a path where every browser ships with a built-in `HTTP Legacy List`. This would invert the current opt-in approach for `https` supporting websites by removing the `HSTS Preload List` because `https` is the new default. Including a list that contains legacy websites which are allowed to be loaded over `http`, would make up for the compatibility risks.

This leaves the question of how to build such a legacy list. It is an open research question on how to collect a list of `http` pages without invading user's privacy by making their clients report their visited sites. It is also unclear how to determine the age of such sites without prolonged monitoring. We are confident that the academic community can help to fix those problems eventually.

B. Securing Local Network Traffic

Currently, the Web Public Key Infrastructure (PKI) does not permit a publicly trusted CA to sign a certificate for a local domain. In more detail, non-publicly resolvable domain names or also router configuration pages currently have to make a trade-off decision: either they do not offer `https`, they rely on complicated workarounds, or use self-signed certificates which comes with the downside that end users have to click through security warnings or go through cumbersome setup steps to import and mark these certificates as trustworthy. None of which is desired. Also, we have learned from the success story of *Let's Encrypt*, that once security becomes easy to use it will find adoption.

The good news is that the web and academic community is aware of the problem and it's getting more and more attention. For example, the topic of securing local traffic was discussed in a breakout session at the *W3C Technical Plenary and Advisory Committee meeting* (TPAC) in 2024 [52]. Further, a proposal to support *HTTPS for Local Domains* [53] was submitted at the Network Working Group in 2024 as well. Earlier works also include a collection of *Approaches to Achieving HTTPS in Local Network* [12] by the now closed *HTTPS in Local Network Community Group* [57].

In summary, our measured numbers indicate that `https` has become the new default for the web, and we are confident that the web and academic community will (a) find solutions for legacy `http` sites on the open web and (b) find a way to secure local network traffic to make `https` not only the default, but encrypt everything on the internet.

Further research is also necessary to determine if there are additional issues which need mitigation before `https` can become truly ubiquitous.

V. Related Work

Creating an encrypted `https` connection is fundamental to providing the necessary security guarantees for any kind of communication between a browser and a web server. Various studies [1], [5], [8], [40], [41] conclude that websites not deploying, or only partially deploying, `https` expose sensitive user information to network attackers. The various upgrading mechanisms presented in Section II aim to mitigate the risk of exposing private user information to the network by upgrading and hence encrypting content using `https` whenever possible. Thus, the presented upgrading mechanisms not only drive up `https` usage but also support websites failing to properly deploy `https` for their sites.

Back in 2008, Jackson and Barth were the first to propose a mechanism [19] that allows web servers to force browsers to interact with a website only using secure `https` connections. This mechanism later served as the foundation for the HSTS specification [15] which, if properly deployed [21], allows websites to protect themselves against protocol downgrade attacks. Besides the downside that the initial request remains unprotected from active attacks (as discussed in Section II), HSTS further has the downside that it opens up an additional tracking vector [38], [55]. It leaks a single bit of information, whether the browser has cached HSTS information, corresponding to whether a user has visited a site before. The various in-browser solutions like *HTTPS-Only* or *HTTPS-First* overcome these limitations by automatically upgrading loads, so that websites are protected by `https`, including the initial request.

At this point it is worth pointing out that all auto upgrading mechanisms have to acknowledge that current web architecture permits resource requests to return different content when queried using `http` or `https`. As discussed by Paracha et al. [37] the number of sites that exhibit such behavior is small, and we argue that this problem will eventually vanish entirely because using the `https` protocol is the new default for any kind of web communication.

Statistics gathered by Porter Felt et al. [9], as well as data supported by the discussion in III-B, shows that around 50% of web browsing took place over `https` in 2017. At that time *Let's Encrypt* reported issuing around half a million certificates per day. At the end of 2024, *Let's Encrypt* reports that their free, automated, and open Certificate Authority issues around five million certificates every day and provides TLS certificates to over 450 million websites. These numbers again confirm that `https` is on the rise and support the claim that `https` is becoming the new default on the web.

Browser vendors obviously played a big role in paving the way for `https` to become the new default for the web. For example, every browser eventually deployed a `Mixed Content Blocker` [48] which, within a top-level site loaded over `https`, blocks `http` content like scripts and styles. In 2015, Chen et al. [1] showed that roughly half of the internet's most popular sites exposed mixed content and were thus exposing their users to cookie stealing attacks and injection of malicious code. The latest iteration of Mixed Content proposal [48] suggests to automatically upgrade `http` sub-resources like audio, video and images to `https` when the top-level site is loaded over `https`. This auto-upgrading of sub-resources ensures that no mixed content is loaded on the page and thus avoids the leakage of user sensitive information to a third party. As of early 2025, all major browsers, Safari [58], Chrome [2] and Firefox [31] are shipping this behavior and automatically upgrade or block all sub-resources from `http` to `https`.

Now browsers not only block mixed content sub-resources, but also mixed content downloads [29]. Mixed content download blocking ensures that downloads initiated on an `https` page cannot use an insecure connection. This is especially important considering that the potentially insecure URL is not displayed to the user since there is no address bar for downloads. Also, browser vendors removed support for the `ftp` [30] protocol in the browser. The File Transfer Protocol (`ftp`) was, for a long time, a convenient way to exchange files between computers on a network. However, equivalent to `http`, the `ftp` protocol allowed attackers to steal, spoof and even modify the transmitted data.

There is no single event or mechanism which magically improved `https` adoption. What rather reflects the truth is that the evolution of the web, steadily over time, allowed and supported more `https` connections. The various upgrading mechanisms, either on the server side, or directly baked into the browser, ensure that as many connections as possible use `https`.

VI. Conclusion

We have discussed the state of `https` adoption on the web. Evaluating data from over 140 million Firefox release users, gathered over the course of one month (Firefox 134 in January 2025) allowed us to present that globally 92.1% of top-level connections already happen over a secure and encrypted connection. We showed that 90.4% of those encrypted loads already started out using an `https` connection and further showed that the different client- and server-side mechanisms allow upgrading an additional 1.7% of connections from `http` to `https` which would otherwise still load using the outdated and insecure protocol.

We further highlighted that `https` adoption is on the rise, though currently varies greatly between different continents and countries. We showed that the difference of `https` adoption ranges from the country with the most measured top-level `https` loads Denmark with 98.0% to the country with the lowest `https` adoption Sudan with 51.2%.

In summary, we conclude that the web has undergone a great transformation and while just ten years ago, only 30% of top level connections happened over `https`, nowadays more than nine out of ten connections happen over `https`. While there is still progress to be made to have all connections rely on `https`, we conclude that `https` has already become the new default on the web.

Acknowledgments

We thank everyone in the Security, Release, Networking and Data teams at Mozilla for their support and insightful comments. In particular we are grateful to the following Mozillians: Cameron Boozarjomehri, Martin Balfanz, Valentin Gosu, Kershaw Jang, Tyler Thorne, Neha Kochar, Andrew Overholt and Martin Thomson. Finally, we also thank our anonymous reviewers and all the other contributors who have helped to pave the way for the web we want: free, open and secure.

References

[1] P. Chen, N. Nikiforakis, C. Huygens, and L. Desmet. A Dangerous Mix: Large-Scale Analysis of Mixed-Content Websites. In *Proceedings of the International Conference on Information Security*, 2015.

[2] Chromium. Chrome Security - Quarterly Updates. https://www.chromium.org/Home/chromium-security/quarterly-updates/#q4-2020, 2020. (checked: February, 2025).

[3] Chromium Blog. Increasing HTTPS adoption. https://blog.chromium.org/2021/07/increasing-https-adoption.html, 2021. (checked: February, 2025).

[4] Chromium Project. Check HSTS preload status and eligibility. https://hstspreload.org/, 2012. (checked: February, 2025).

[5] K. Drakonakis, S. Ioannidis, and J. Polakis. The Cookie Hunter: Automated Black-Box Auditing for Web Authentication and Authorization Flaws. In *Proceedings of the Conference on Computer and Communications Security*, 2020.

[6] EFF. HTTPS:// Everywhere. https://www.eff.org/https-everywhere, 2014. (checked: February, 2025).

[7] EFF. HTTPS Is Actually Everywhere. https://www.eff.org/deeplinks/2021/09/https-actually-everywhere, 2021. (checked: February, 2025).

[8] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten. Cookies That Give You Away: The Surveillance Implications of Web Tracking. In *Proceedings of the International Conference on World Wide Web*, 2015.

[9] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz. Measuring HTTPS Adoption on the Web. In *USENIX Security Symposium*, 2017.

[10] Google Chrome Team. Manage Chrome safety and security. https://support.google.com/chrome/answer/10468685, 2024. (checked: February, 2025).

[11] Google Search Central Blog. HTTPS as a ranking signal. https://developers.google.com/search/blog/2014/08/https-as-ranking-signal, 2014. (checked: February, 2025).

[12] HTTPS in Local Network Community Group. Approaches to Achieving HTTPS in Local Network. https://httpslocal.github.io/proposals/, 2019. (checked: February, 2025).

[13] Internet Engineering Task Force (IETF). Hypertext Transfer Protocol – HTTP/1.1). https://tools.ietf.org/html/rfc2616, 1999. (checked: February, 2025).

[14] Internet Engineering Task Force (IETF). HTTP Over TLS. https://tools.ietf.org/html/rfc2818, 2000. (checked: February, 2025).

[15] Internet Engineering Task Force (IETF). HTTP Strict Transport Security (HSTS). https://tools.ietf.org/html/rfc6797, 2012. (checked: February, 2025).

[16] Internet Engineering Task Force (IETF). Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension. https://datatracker.ietf.org/doc/html/rfc7301, 2014. (checked: February, 2025).

[17] Internet Engineering Task Force (IETF). HTTPS RR. https://datatracker.ietf.org/doc/draft-ietf-dnsop-svcb-https/00/, 2020. (checked: February, 2025).

[18] Internet Engineering Task Force (IETF). Introduction to the IETF. https://www.ietf.org/, 2024. (checked: February, 2025).

[19] C. Jackson and A. Barth. Forcehttps: protecting high-security web sites from network attacks. In *Proceedings of the International Conference on World Wide Web*, 2008.

[20] C. Kerschbaumer, J. Gaibler, A. Edelstein, and T. van der Merwe. HTTPS-Only: Upgrading all connections to https in Web Browsers. *Proceedings on Measurements, Attacks, and Defenses for the Web*, 2021.

[21] M. Kranch and J. Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *Network and Distributed System Security (NDSS) Symposium*, 2015.

[22] Let's Encrypt. Let's Encrypt Stats. https://letsencrypt.org/stats/, 2024. (checked: February, 2025).

[23] MDN Web Docs. Browser extensions. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions, 2024. (checked: February, 2025).

[24] Microsoft Edge Team. Available for preview: Automatic HTTPS helps keep your browsing more secure. https://blogs.windows.com/msedgedev/2021/06/01/available-for-preview-automatic-https-helps-keep-your-browsing-more-secure/, 2021. (checked: February, 2025).

[25] Moxie Marlinspike. New Tricks For Defeating SSL In Practice. https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf, 2009. (checked: February, 2025).

[26] Mozilla. Preloading HSTS. https://blog.mozilla.org/security/2012/11/01/preloading-hsts/, 2012. (checked: February, 2025).

[27] Mozilla. Mozilla Data Documentation. https://docs.telemetry.mozilla.org/, 2020. (checked: February, 2025).

[28] Mozilla. Firefox 91 introduces HTTPS by Default in Private Browsing. https://blog.mozilla.org/security/2021/08/10/firefox-91-introduces-https-by-default-in-private-browsing/, 2021. (checked: February, 2025).

[29] Mozilla. Firefox 93 protects against Insecure Downloads. https://blog.mozilla.org/security/2021/10/05/firefox-93-protects-against-insecure-downloads/, 2021. (checked: February, 2025).

[30] Mozilla. Stopping FTP support in Firefox 90. https://blog.mozilla.org/security/2021/07/20/stopping-ftp-support-in-firefox-90/, 2021. (checked: February, 2025).

[31] Mozilla. Firefox will upgrade more Mixed Content in Version 127. https://blog.mozilla.org/security/2024/06/05/firefox-will-upgrade-more-mixed-content-in-version-127/, 2024. (checked: February, 2025).

[32] Mozilla. Source Code for nsSTSPreloadList. https://searchfox.org/mozilla-central/source/security/manager/ssl/nsSTSPreloadList.inc, 2024. (checked: February, 2025).

[33] Mozilla. Telemetry Portal. https://telemetry.mozilla.org/, 2024. (checked: February, 2025).

[34] Mozilla. Telemetry Portal. https://docs.telemetry.mozilla.org/concepts/pipeline/gcp_data_pipeline.html#decoding, 2024. (checked: February, 2025).

[35] Mukunda Modell, Aiko Barz, Gabor Toth, Carlo v. Loesch. Certificate Patrol. http://patrol.psyced.org/, 2011. (checked: February, 2025).

[36] Mustafa Emre Acer and Carlos Joan Rafael Ibarra Lopez. HTTPS Upgrades. https://github.com/whatwg/fetch/pull/1655, 2024. (checked: February, 2025).

[37] M. T. Paracha, B. Chandrasekaran, D. R. Choffnes, and D. Levin. A Deeper Look at Web Content Availability and Consistency over HTTP/S. In *Network Traffic Measurement and Analysis Conference*, 2020.

[38] Paul Syverson and Matthew Traudt. HSTS Supports Targeted Surveillance. In *USENIX Security Symposium*, 2018.

[39] K. Ruth, A. Fass, J. Azose, M. Pearson, E. Thomas, C. Sadowski, and Z. Durumeric. A world wide view of browsing the world wide web. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 317–336, New York, NY, USA, 2022. Association for Computing Machinery.

[40] S. Sivakorn, A. D. Keromytis, and J. Polakis. That's the Way the Cookie Crumbles: Evaluating HTTPS Enforcing Mechanisms. In *Proceedings of the Workshop on Privacy in the Electronic Society*, 2016.

[41] S. Sivakorn, I. Polakis, and A. D. Keromytis. The cracked cookie jar: HTTP cookie hijacking and the exposure of private information. In *Symposium on Security and Privacy*, 2016.

[42] The Certification Authority Browser Forum (CA/Browser Forum). Latest Baseline Requirements. https://cabforum.org/working-groups/server/baseline-requirements/requirements/, 2024. (checked: February, 2025).

[43] The Web Hypertext Application Technology Working Group (WHATWG). Fetch Standard. https://fetch.spec.whatwg.org/, 2024. (checked: February, 2025).

[44] The World Wide Web Consortium (W3C). HTML Design Principles. https://www.w3.org/TR/html-design-principles/#support-existing-content, 2007. (checked: February, 2025).

[45] The World Wide Web Consortium (W3C). Same-Origin Policy (SOP). https://www.w3.org/Security/wiki/Same_Origin_Policy, 2010. (checked: February, 2025).

[46] The World Wide Web Consortium (W3C). Upgrade Insecure Requests. https://www.w3.org/TR/upgrade-insecure-requests/, 2015. (checked: February, 2025).

[47] The World Wide Web Consortium (W3C). 30 years on from introducing the Web to the World. https://www.w3.org/blog/2021/30-years-on-from-introducing-the-web-to-the-world//, 2020. (checked: February, 2025).

[48] The World Wide Web Consortium (W3C). Mixed Content. https://www.w3.org/TR/mixed-content/, 2023. (checked: February, 2025).

[49] The World Wide Web Consortium (W3C). Content Security Policy Level 3. https://www.w3.org/TR/CSP3/, 2024. (checked: February, 2025).

[50] The World Wide Web Consortium (W3C). Making the Web work. https://www.w3.org/, 2024. (checked: February, 2025).

[51] The World Wide Web Consortium (W3C). Secure Contexts. https://www.w3.org/TR/secure-contexts/, 2024. (checked: February, 2025).

[52] The World Wide Web Consortium (W3C). TPAC Meeting Notes: HTTPS For Local Networks. https://www.w3.org/events/meetings/09083118-ca0a-4347-9271-6adf7798c935/, 2024. (checked: February, 2025).

[53] Thomson, M and Ed, D. Wing. HTTPS for Local Domains. https://danwing.github.io/https-local-domains/draft-thomson-https-local-domains.html, 2024. (checked: February, 2025).

[54] Tor. The Tor Browser. https://www.torproject.org/, 2024. (checked: February, 2025).

[55] M. Traudt and P. Syverson. Does Pushing Security on Clients Make Them Safer? https://petsymposium.org/2019/files/hotpets/proposals/pushing-insecurity-hotpets19.pdf, 2019.

[56] UN Trade and Development (UNCTAD) . UN list of least developed countries. https://unctad.org/topic/least-developed-countries/list, 2024. (checked: February, 2025).

[57] W3C Community & Business Groups. HTTPS in Local Network Community Group. https://www.w3.org/community/httpslocal/, 2017. (checked: February, 2025).

[58] Webkit. WebKit Features in Safari 18.0. https://webkit.org/blog/15865/webkit-features-in-safari-18-0/#https, 2024. (checked: February, 2025).

[59] WebKit. WebKit Features in Safari 18.2. https://webkit.org/blog/16301/webkit-features-in-safari-18-2/, 2024. (checked: February, 2025).

[60] D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing. In *USENIX Security Symposium*, 2008.