HTTPS-First as the New Standard

Doğa Ninsun Gezer, Lion Steger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: ninsun.gezer@tum.de, steger@net.in.tum.de

***Abstract*—The modern web has largely shifted to encrypted connections, with the vast majority of traffic using the Hypertext Transfer Protocol Secure (HTTPS). This paper investigates how current web browsers initiate connections to servers and the use of encrypted channels for the initial contact. We analyze browser behavior empirically, revealing a shift towards HTTPS-First behaviour, but with varied implementations including direct HTTPS attempts, HTTP probes, and differing QUIC/HTTP3 adoption. Crucially, most retain an insecure fallback to HTTP. We categorize server-side upgrade mechanism like HTTP Strict Transport Security (HSTS) and HSTS preloading, and client-side upgrade mechanisms like HTTPS-Only Mode, analyzing their specific security guarantees and evaluating their effectiveness. By studying attacker methods like SSL stripping, we demonstrate how an insecure fallback behaviour to HTTP can be exploited. We explore the underlying reasons for HTTP persistence, rooted in backward compatibility and local network constraints.**

***Index Terms*—HTTPS, HTTP, HSTS, Network Security**

1. Introduction

The Hypertext Transfer Protocol Secure (HTTPS) is a well-established secure alternative to the Hypertext Transfer Protocol (HTTP). HTTPS provides confidentiality and integrity by using SSL/TLS protocols to establish an encrypted channel between the client and the server. [1] When a user enters a domain name without specifying a protocol (e.g., example.com), the browser's choice of default protocol becomes a critical security decision.

Historically, browsers defaulted to an insecure HTTP connection [**?**], exposing the initial interaction to Man-in-the-Middle (MitM) attacks before potential redirection. [2] Mitigation strategies like HSTS and HSTS preloading [2] were developed to address this vulnerability, signaling to browsers that all communication in the future should be through HTTPS. Consequently, all major browsers have shifted from the HTTP-First approach and now use HTTPS-First connection models.

This paper investigates the practical security implications within the web landscape. We first explain the context of HTTP's persistence and the available upgrade mechanisms. Then, we empirically analyze how major browsers currently implement their initial connection strategies. We subsequently analyze the specific risks arising from the HTTP fallback mechanism. Finally, we synthesize these findings to understand the current security posture regarding initial web connections.

2. Background

This section explains why HTTP persists despite HTTPS, clarifies historical context, and details the mechanisms aimed at upgrading connections, addressing the questions of why browsers cannot simply always use HTTPS and what problems each upgrade mechanism solves.

2.1. The Persistence of HTTP

Despite HTTPS becoming the new standard for the web [3], HTTP continues to persist as a fallback. The complete deprecation of HTTP is prevented by two primary challenges: backward compatibility and the unsolved problem of local network encryption. This section first examines these primary challenges. It then surveys the key mechanisms developed to enforce secure connections, distinguishing between server-side and client-side approaches [4].

2.1.1. Backward Compatibility. A primary reason for retaining HTTP support is backward compatibility and the persistent high prevalence of HTTPS configuration errors. We define these errors as issues within the SSL/TLS certificate, such as being from an untrusted Certificate Authority (CA), a mismatching domain name, or being expired, as well as mixed content warnings [5]. Early research, such as Jackson et al. (2008), mentions the "unknown intent" problem, where distinguishing between self-signed certificates on low-security sites and malicious attacks is difficult [6].

A 2022 study of 1,562 popular websites by Alabduljabbar et al. found that TLS certificate issues were significantly more common on free content sites (35.85%) than on paid content sites (6.99%). These problems included invalid certificates (17.42%), domain name mismatches (11.47%), and expired certificates (6.97%). This demonstrates that many websites still fail to implement HTTPS correctly, reinforcing the need for browsers to handle such cases without completely blocking access [5].

2.1.2. The Local Network Problem. The web's security model, based on the Public Key Infrastructure, does not extend to private networks. CAs are unable to issue trusted certificates for private IP addresses (e.g., 192.168.1.1) or reserved local hostnames (e.g., router.local) [7]. This makes HTTP the only practical protocol for essential tasks like configuring home routers, accessing corporate intranets, and running local development servers, forcing browsers to retain HTTP support.

 doi: 10.2313/NET-2026-05-1_05

2.2. Mechanisms for Upgrading to HTTPS

There are several mechanisms used to upgrade from HTTP to HTTPS. Implementations of server-side mechanisms are generally identical across the web as they follow standards from the World Wide Web Consortium or the Internet Engineering Task Force, whereas client-side mechanisms differ in their implementation in browsers [4].

Mechanism	Share of Upgrades (%)
HSTS (incl. Preload)	62.4
HTTPS-First (Browser Default)	30.4
Web Extension	3.3
HTTPS-Only (Opt-in)	2.3
HTTPS RR (DNS)	1.3
CSP upgrade-insecure-requests	0.3

TABLE 1: Effectiveness of Top-Level HTTP-to-HTTPS Upgrade Mechanisms (Source: [3])

HSTS, as specified in RFC 6797, allows a server to send a `Strict-Transport-Security` header over an HTTPS connection [2]. This instructs the browser to communicate with that domain exclusively over HTTPS in the future. However, HSTS relies on the principle of Trust On First Use, leaving the client's first connection unprotected from the MitM vulnerability [6].

To solve the first-visit problem, browser vendors maintain a list of domains that are hardcoded as HTTPS-Only [8]. For these domains, the browser enforces HTTPS from the very first connection, eliminating the window for SSL stripping attacks. While the list covers most high-traffic sites and together with HSTS is responsible for the majority (62.4%) of all successful HTTP-to-HTTPS upgrades as shown in the TABLE 1, it is not exhaustive [4]. The preload list could potentially include a wildcard for every top-level domain, but that would still create backward-compatibility issues [9].

The HTTPS-First Mode is the default behavior in most modern browsers. The browser first attempts to establish a secure HTTPS connection. If this fails (e.g., the server does not respond on port 443 or has an invalid certificate), it silently and automatically falls back to an insecure HTTP connection [10]. This opportunistic approach is the second-most effective upgrade mechanism, responsible for 30.4% of all upgrades as shown in TABLE 1, but its fallback behavior is an exploitable weakness [4].

The HTTPS-Only Mode is offered as a stricter, opt-in feature in major browsers, this mode also attempts to upgrade all connections but does not automatically fall back. If a secure connection fails, it blocks the navigation and presents the user with a security warning, prioritizing security over compatibility [10].

The `upgrade-insecure-requests` directive within a Content-Security-Policy (CSP) can upgrade same-origin navigations, while DNS-based HTTPS Resource Records (HTTPS RR) can signal HTTPS support during lookup [11], [12]. However, as shown in TABLE 1 their real-world impact on top-level upgrades is minimal, at 0.3% and 1.3% respectively [4].

3. Attacks Targeting the HTTPS-First Fallback

While HTTPS-First policies secure the data channel, they do not address all threats. Application-layer vulnerabilities like Cross-Site Scripting [13], client-side malware, and phishing attacks using valid TLS certificates remain potent threats. This analysis, however, focuses specifically on protocol downgrade attacks, which exploit the persistence of HTTP as a fallback mechanism. These attacks assume an adversary has a privileged MitM position, achievable by controlling a Wi-Fi access point, ARP spoofing, or DNS hijacking [14].

The primary threat in this context is SSL stripping, first demonstrated by Moxie Marlinspike [15]. A modern variant of this attack weaponizes the very compatibility feature designed to make HTTPS-First practical: its silent fallback to HTTP. An attacker can intentionally disrupt the browser's initial attempt to connect via HTTPS, for instance, by blocking traffic to port 443. This failure triggers the browser's automatic and silent fallback to insecure HTTP, handing the attacker the unencrypted request needed to initiate the attack.

Critically, this attack vector circumvents dynamic HSTS by exploiting its "Trust On First Use" vulnerability. If an attacker intercepts the very first connection, the browser never receives the `Strict-Transport-Security` header over a trusted channel. The attack itself prevents the deployment of the primary mechanism designed to stop it, as browsers ignore the header when sent over an insecure connection [2].

The attack leverages the browser's predictable fallback logic in three steps:

1) **Disrupt Initial HTTPS Attempt:** The MitM attacker actively prevents the browser's initial connection attempt to port 443 from succeeding. This can be done by blocking the traffic.
2) **Trigger Silent Fallback:** The browser, encountering a failure on port 443 and operating in the default HTTPS-First mode, automatically initiates a new connection attempt to port 80 (HTTP) without user notification.
3) **Intercept and Downgrade (SSL Stripping):** The attacker intercepts the unencrypted HTTP request. They establish their own secure HTTPS connection to the legitimate server, receive the content, strip all security elements (e.g., changing `https://` links to `http://`), and forward the modified, insecure HTTP version to the user [15].

From the user's perspective, the site simply loads over HTTP without any error, defeating the purpose of HTTPS. This demonstrates how a feature designed for compatibility (the silent fallback) becomes an exploitable flaw. Consequently, the HSTS Preload List remains the only mechanism capable of fully protecting users from this protocol downgrade attack on their first visit, as its policy is known to the browser before any connection is made [8].

4. Analysis of Modern Browser Behavior

To investigate the practical deployment of HTTPS-First connection strategies, this study analyzed the net-

 doi: 10.2313/NET-2026-05-1_05

work traffic of five widely used browsers as of September 2025 [16]: Google Chrome, Safari, Edge, Opera, and Firefox. Using the latest stable version of each browser after a clean installation, all network connections initiated when rendering page content were captured. .

Our experimental setup employed Wireshark, an open-source network protocol analyzer, to capture and inspect traffic. For each test, we entered the schemeless domain `example.com` into the browser's address bar. This domain was selected because it is accessible via both HTTP, HTTPS and QUIX but is not on Chromium's HSTS preload list [8]. We analyzed the sequence and destination ports (80 vs. 443) of initial TCP SYN packets, any subsequent UDP (QUIC) connection attempts, and the protocol used for the first application data transmission (TLS vs. HTTP vs. QUIC) The browser's behavior was categorized based on the first application-layer data packet sent:

- HTTPS-First Behavior: The browser sends a `TLSv1.x Client Hello` over a TCP connection to port 443. Any connections to port 80 are either closed or remain idle.
- HTTP-First Behavior: The browser sends an `HTTP GET` request over a TCP connection to port 80.

Following the initial connection, browsers showed differing tendencies to upgrade to QUIC/HTTP3 [**?**]. *Aggressive Upgraders (Chrome, Edge, Opera):* The Chromium-based browsers actively attempted QUIC connections to `example.com`, indicating a strategy favoring potential QUIC performance benefits. *Conservative Adopters (Firefox, Safari):* Firefox and Safari defaulted to using the established TCP/TLS connection for `example.com`. Firefox demonstrated QUIC capability via background traffic but applied it selectively, perhaps prioritizing TCP's maturity or specific performance heuristics not met by `example.com`. Safari generally adopts new protocols more cautiously.

This empirical analysis connects the need for HTTP support (Sec 2) to concrete browser implementation choices and their inherent trade-offs, leading into a discussion of the overall implications.

4.1. Google Chrome

Google Chrome began implementing an "HTTPS-First" mode in 2021 [17].The test of Google Chrome (v140.0.7339.213) showed that upon entering the schemeless URL, the browser initiated three parallel TCP connection attempts: two to port 80 (HTTP) and one to port 443 (HTTPS). While the TCP three-way handshakes completed for all three connections, the `TLSv1.3 Client Hello` was sent exclusively over the port 443 connection. The two connections to port 80 were established but remained idle, with no `HTTP GET` requests sent. Later Chrome also attempted to establish a QUIC connection for `example.com`.

This behavior is a strategy, where parallel connections to port 80 function as a possible browserside internal optimization. By establishing these connections preemptively, Chrome prepares for a potential fallback scenario where a server may not support HTTPS, aiming to improve performance without waiting for a server-side redirect. This is one of several security measures in the browser, which also include enforcing the HSTS Preload List [8] and adhering to the Mixed Content Specification by blocking or upgrading insecure sub-resources within a secure page [18].

4.2. Firefox

Firefox, developed by the Mozilla Foundation, is an open-source browser that operates on its own Gecko engine. Firefox has implemented security features such as an HTTPS-Only Mode and an HTTPS-First Mode [10], with the latter being the default in its Private Browsing mode since version 91 [19].

The analysis of Firefox (v143.0.1) demonstrated a strategy that initiates connections exclusively over HTTPS. Upon entering `example.com`, Firefox made no attempt to connect to port 80 (HTTP). Instead, it immediately opened two parallel TCP connections directly to port 443 (HTTPS). The `TLSv1.3 Client Hello` was sent over the first of these two secure connections to complete the handshake. Firefox used the established TCP/TLS connection and did not attempt a QUIC upgrade, although background traffic indicated QUIC capability for other services.

This method differs from the speculative fallback optimization observed in other browsers, as it completely bypasses the insecure protocol during the initial connection phase. This approach improves security by eliminating the brief window where an attacker might interact with an insecure connection. The HTTPS-First mode in Firefox is designed to be opportunistic, attempting an upgrade to HTTPS and only falling back to HTTP if a secure connection cannot be established [10]. The use of two parallel connections to port 443 is an internal performance optimization to ensure a fast and reliable TLS handshake.

4.3. Safari

In our test of Safari (v18.5), the browser implemented an HTTPS-First protocol by initiating two parallel TCP connection attempts: one to port 80 (HTTP) and one to port 443 (HTTPS). Following the successful completion of both TCP handshakes, Safari sent the `TLSv1.3 Client Hello` exclusively over the port 443 connection. The connection on port 80 was established but remained idle. Safari did not attempt to upgrade to QUIC/HTTP3 for `example.com`.

This behavior is a HTTPS-First approach that uses a speculative optimization, similar to Chromium-based browsers but with fewer parallel connections. By preparing a connection to port 80, Safari is ready to fall back to HTTP if the HTTPS attempt fails, but it prioritizes the secure connection. Safari also incorporates security features such as support for the HSTS Preload List [8] and mixed content blocking [18], aligning with modern web security practices.

4.4. Microsoft Edge

In 2021, Microsoft introduced an HTTPS upgrade feature termed "Automatic HTTPS" [20]. Our analysis of Microsoft Edge (v140.0.3485.94) confirmed that its

connection strategy is identical to that of Google Chrome. Upon entering the schemeless URL, Edge initiated three parallel TCP connection attempts: two to port 80 and one to port 443. All three handshakes completed, but the `TLSv1.3 Client Hello` was sent only over the port 443 connection. The other two connections to port 80 remained unused. Edge also attempted to establish a QUIC connection for `example.com` after the initial TLS handshake.

This behavior is a direct result of its Chromium foundation. The parallel connections serve as a speculative optimization, allowing the browser to proceed with a secure TLS handshake while being prepared for a fallback scenario. As part of the Chromium ecosystem, Edge also implements security features such as the HSTS Preload List and mixed content blocking [8], [18].

4.5. Opera

The analysis of Opera One (v122.0.5643.71) revealed a connection strategy identical to that of Firefox. Opera made no attempt to connect to port 80. Instead, it immediately opened two parallel TCP connections directly to port 443 (HTTPS) and sent the `TLSv1.3 Client Hello` over the first available secure connection. Opera also attempted a QUIC upgrade for `example.com` after establishing the TLS connection.

This finding shows a divergence from other Chromium-based browsers in the study. Opera has implemented a networking strategy for initial connections that aligns with Firefox's approach, prioritizing a secure-only initial attempt. This choice enhances security by removing the opportunity for downgrade attacks during the connection phase. While it customizes this initial step, Opera still utilizes the broader security architecture of the Chromium engine, including its handling of mixed content and support for HSTS.

TABLE 2: Observed Browser Behavior Summary for `example.com`

Browser	Initial TCP SYNs	QUIC Attempt
Chrome	1x HTTPS, 2x HTTP	Yes
Edge	1x HTTPS, 2x HTTP	Yes
Safari	1x HTTPS, 1x HTTP	No
Firefox	2x HTTPS only	No
Opera	2x HTTPS only	Yes

5. Conclusion

The web has undergone a significant but incomplete transition towards a secure-by-default model. Our analysis shows that the default behavior for most major browsers has shifted from HTTP-First to a more secure HTTPS-First model. And we can cofnidently say that HTTPS has become the new standard for the web. However, this is fundamentally a web compatibility measure, not a security guarantee. The reliance on a silent, insecure fallback provides a clear and exploitable entry point for man-in-the-middle attacks like SSL stripping.

Truly reliable protection is only achieved through explicit, server-declared policies. HTTP Strict Transport Security, particularly when combined with the HSTS Preload List, remains the gold standard, offering the only comprehensive solution to the "first-visit" vulnerability that plagues all other mechanisms.

The complete deprecation of HTTP, while a desirable end-goal, is currently infeasible. The need to maintain access to legacy content and the fundamental architectural challenge of extending the web's trust model to private, local networks ensures that HTTP will persist as a necessary fallback for the foreseeable future. Future efforts must therefore focus not only on increasing HSTS adoption but also on developing new standards to solve the problem of local network encryption, which remains the most significant barrier to a fully encrypted web.

References

[1] Internet Engineering Task Force (IETF), "HTTP Over TLS," https://tools.ietf.org/html/rfc2818, 2000, [Online; accessed September, 2025].

[2] ——, "HTTP Strict Transport Security (HSTS)," https://tools.ietf.org/html/rfc6797, 2012, [Online; accessed September, 2025].

[3] C. Kerschbaumer, F. Braun, S. Friedberger, and M. Jürgens, "The State of https Adoption on the Web," 2025.

[4] 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.

[5] A. Alabduljabbar, R. Ma, S. Choi, R. Jang, S. Chen, and D. Mohaisen, "Understanding the Security of Free Content Websites by Analyzing their SSL Certificates: A Comparative Study," in *Proceedings of the 1st Workshop on Cybersecurity and Social Sciences (CySSS '22)*. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3494108.3522769

[6] 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.

[7] The Certification Authority Browser Forum (CA/Browser Forum), "Latest Baseline Requirements," https://cabforum.org/working-groups/server/baseline-requirements/requirements/, 2024, [Online; accessed September, 2025].

[8] C. Project, "Check HSTS preload status and eligibility," https://hstspreload.org/, 2012, [Online; accessed September, 2025].

[9] Mozilla, "Preloading HSTS," https://blog.mozilla.org/security/2012/11/01/preloading-hsts/, 2012, [Online; accessed September, 2025].

[10] 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.

[11] The World Wide Web Consortium (W3C), "Upgrade Insecure Requests," https://www.w3.org/TR/upgrade-insecure-requests/, 2015, [Online; accessed September, 2025].

[12] Internet Engineering Task Force (IETF), "HTTPS RR," https://datatracker.ietf.org/doc/draft-ietf-dnsop-svcb-https/00/, 2020, [Online; accessed September, 2025].

[13] KirstenS, "Cross site scripting (xss)," https://owasp.org/www-community/attacks/xss/, 2019, [Online; accessed October, 2025].

[14] 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.

[15] M. 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, [Online; accessed September, 2025].

[16] StatCounter, "Desktop Browser Market Share Worldwide," https://gs.statcounter.com/browser-market-share/desktop/worldwide, 2025, [Online; accessed September, 2025].

[17] C. Blog, “Increasing HTTPS adoption,” https://blog.chromium.org/2021/07/increasing-https-adoption.html, 2021, [Online; accessed September, 2025].

[18] The World Wide Web Consortium (W3C), “Mixed Content,” https://www.w3.org/TR/mixed-content/, 2023, [Online; accessed September, 2025].

[19] 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, [Online; accessed September, 2025].

[20] 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, [Online; accessed September, 2025].

 doi: 10.2313/NET-2026-05-1_05