A Measurement Study of OWASP Security Headers and Misconfigurations on Real-World Websites

Mr. Nattapol Prairuenrom

Student in Computer Engineering and Informatics Faculty of Engineering, Kasetsart University, Thailand

Email: Nattaqpol.prai@ku.th

Prof. (Dr.) Kailas Patil

Dean, Faculty of Science & Technology & Chief Information Security Officer Faculty of Science & Technology

Vishwakarma University, Pune, India

Email: kailas.patil@vupune.ac.in

Asst. Prof. Dr. Prawit Chumchu

Assistant Professor, Faculty of Engineering Kasetsart University, Thailand

Email: prawit@eng.src.ku.ac.th

Abstract

Modern web applications face a wide range of security threats, many of which can be mitigated through the proper implementation of HTTP Security Headers, as recommended by the OWASP Secure Headers Project [1]. However, the adoption and correct configuration of these headers remain inconsistent across real-world websites.

In this study, we conduct a large-scale empirical analysis of 17 critical HTTP security headers across one million domains, using data from the Majestic Million dataset [3]. Our findings reveal widespread misconfigurations, outdated practices, and significant gaps between recommended best practices and real-world deployments. To address these challenges and enhance user-side visibility, we introduce a Chrome browser extension that dynamically inspects HTTP security headers and client-side DOM behaviors in real time. It includes a DOM security analyzer with an auto-fix module capable of detecting and mitigating client-side risks such as insecure inline scripts, iframes, and meta-policy conflicts. By combining large-scale measurement with real-time analysis and AI-powered feedback, our work provides both systemic insights and practical tools for improving web application security.

Keywords: HTTP Security Headers, OWASP, Web Security, Header Misconfiguration, Chrome Extension, DOM Analysis, Majestic Million

1 Introduction

The web browser security model is rooted in the same-origin policy (SOP) [5], which isolates one origin's resources from another to prevent unauthorized access. However, attackers can bypass SOP by injecting malicious content into vulnerable websites using techniques such as Cross-Site Scripting (XSS), clickjacking, or data leakage attacks. According to OWASP's 2013 vulnerability assessment, XSS remains one of the top five most common vulnerabilities . The fundamental issue is that browsers cannot distinguish between trusted and injected content, making client-side defenses essential.To mitigate such risks, various HTTP Security Headers have been introduced and standardized, including Strict-Transport-Security (HSTS), Content-Security-Policy (CSP), X-Frame-Options, and Permissions-Policy. These headers enable website operators to define content behavior and security restrictions via HTTP response metadata, which browsers then enforce automatically. For instance, HSTS ensures secure HTTPS usage, CSP restricts allowed content sources, and X-Frame-Options prevents clickjacking.

Despite their importance, these headers are often underutilized or incorrectly configured. Writing effective and comprehensive security header policies requires in-depth knowledge of both web architecture and browser behavior. Misconfigurations can lead to broken functionality or a false sense of security. In large organizations, developers may not have direct access to modify HTTP headers, while smaller websites may lack security expertise entirely. These operational and knowledge barriers hinder the widespread adoption of security headers in real-world applications, as evidenced in our results presented in Section 2.

The goal of this paper is to systematically investigate the adoption and correctness of HTTP Security Headers on real-world websites, identify common misconfigurations and deprecated practices, and evaluate the practical usability of security headers at scale. Additionally, we propose a browser extension, named Check Header OWASP, to analyze headers in real-time and provide OWASP-guided, AI-powered explanations to assist developers and security-aware users in understanding and improving website security posture.

Our Study. In this work, we study the adoption and configuration of 17 HTTP Security Headers across real-world desktop and mobile websites, and identify common misconfigurations, outdated practices, and gaps in security header enforcement. To accomplish this, we used a Python-based crawler to scan HTTP response headers from the top 1 million websites listed in the Majestic Million dataset [5]. Multiple user agent strings were used to ensure consistent behavior across browser types and platforms.

Based on empirical data collected across all domains, we draw several inferences about the challenges hindering widespread adoption of security headers. Our results show three major factors contributing to incorrect or missing header implementation. First, developers are reluctant to introduce strict security policies such as CSP or Permissions-Policy due to the risk of breaking frontend functionality and losing users. Second, a lack of familiarity with modern HTTP headers and their correct configurations often leads to misconfigured or ineffective policies. Third, the technical effort required to audit, implement, and maintain correct headers — especially across legacy codebases — continues to be a significant barrier.

Moreover, browsers do not provide end users with a native mechanism to apply or override header policies, meaning users have no control over their own security posture when websites fail to implement protection. Security-aware users may prefer strict protection over rich interactivity, yet have no means to enforce this preference.

To assist both developers and users, we developed a Chrome browser extension named Check Header OWASP [1] that analyzes HTTP Security Headers in real-time and provides visual feedback. Unlike previous tools such as CSP AiDer , which relies on static HTML analysis and fails to detect dynamically injected content, Check Header OWASP [1] operates on live web traffic. It observes real HTTP response headers within the browser environment, providing accurate insight into deployed policies.

The extension also integrates with Gemini, a large language model from Google [4], to explain header presence, misconfiguration, or absence using natural language. This AI-backed functionality enables developers and non-expert users to understand technical policies without deep prior knowledge.

Overall, the Check Header OWASP extension enables automatic policy interpretation, highlights deprecated headers, and encourages users to evaluate security in real time. It serves not only as a diagnostic tool, but as a bridge between technical detail and human understanding, helping promote better security practices across the web. In the latest version, Check Header OWASP also supports DOM-level analysis and fixing, allowing users to identify insecure DOM patterns and apply secure rewrites locally. This empowers users to not only inspect headers but also mitigate client-side risks dynamically.

Contributions. The goal in this paper is to study the usage and configuration correctness of HTTP Security Headers on real-world web applications, and to raise awareness among developers to avoid common misconfigurations observed in production environments. We propose a solution, a browser extension for header inspection and analysis, to ease the understanding and adoption of secure HTTP header practices. The goals of the extension are two-fold: i) to allow security-conscious users to evaluate and understand headers on any visited website, and ii) to allow developers to experiment with and visualize HTTP Security Headers on their production pages. Moreover, the extension assists users and developers in interpreting complex headers by providing OWASP-guided explanations and AI-generated recommendations powered by Gemini.

In summary, this paper makes the following contributions:

- We performed a large-scale study on the top 1,000,000 websites from the Majestic Million dataset to evaluate the adoption, correctness, and misconfiguration of 17 HTTP Security Headers.
- We draw inferences on the likely reasons that are hindering proper adoption of modern headers (e.g., CSP, Permissions-Policy) and highlight the continued presence of deprecated headers (e.g., Public-Key-Pins, X-XSS-Protection).
- We design and develop a Chrome browser extension, Check Header OWASP, that automatically detects HTTP Security Headers, compares them to OWASP and MDN recommendations, and provides visual feedback on their presence and configuration.
- We propose an approach for applying real-time client-side analysis and AI-driven interpretation to help developers and end-users better understand header functionality and security impact.
- We introduce a DOM Fixing feature in the extension that analyzes inline scripts, iframe behaviors, and meta-based directives. The tool provides automatic suggestions and allows users to apply secure DOM modifications on the fly, enhancing client-side protection beyond HTTP headers.

Our experiments show a widespread lack of correct HTTP Security Header implementation on real-world websites and demonstrate the necessity for tools like the developed browser extension to encourage adoption. The extension provides developers with an accessible mechanism to evaluate, learn, and improve header configurations without breaking website functionality.

The rest of this paper is organized as follows: Section 2 presents our experimental evaluation and analysis. Section 3 describes the design of the browser extension. Section 4 describes evaluation of our approach, and we conclude the paper in Section 6.

2 Experimental Evaluation and Analysis

We conducted empirical measurements to obtain data for evaluating the adoption, correctness, and misconfiguration of 17 HTTP Security Headers in the wild. Our measurements were conducted using a Python-based asynchronous crawler deployed on a Linux system. The crawling infrastructure utilized a multi-core architecture and parallelized requests to process the Majestic Million dataset efficiently. Additionally, a Chrome browser extension was developed to complement the analysis with real-time, client-side observation of HTTP header behavior.

2.1 Measurement Goals

This study sets out to evaluate the state of HTTP Security Header adoption and correctness across a large-scale dataset of real-world websites. Specifically, the goals are:

Goal 1: Identify inconsistencies in the deployment and enforcement of HTTP security headers.

Goal 2: Detect misconfigurations or incorrect values in deployed security headers.

Goal 3: Estimate the effort and accuracy challenges faced by developers in adopting modern security practices.

2.2 Measurement over Majestic Top 1,000,000 Desktop Websites

In our experiments, we developed a large-scale crawler using Python's httpx and asyncio frameworks to scan the HTTP response headers of the top 1,000,000 websites listed in the Majestic Million dataset. The goal was to assess the real-world adoption and correctness of 17 key HTTP security headers. These include critical modern headers such as Content-Security-Policy, Strict-Transport-Security, and Permissions-Policy, as well as legacy and deprecated headers like X-XSS-Protection, Public-Key-Pins, and Feature-Policy.

Header Name	Status	Example / Use Case
Content-Security-Policy	Active	Prevents XSS via script-src, default-src
Strict-Transport-Security	Active	Forces HTTPS connections
Permissions-Policy	Active	Restricts features like camera, geolocation
Referrer-Policy	Active	Controls referer header sent to other origins
Clear-Site-Data	Active	Clears cookies, storage, cache
Cross-Origin-Opener-Policy	Active	Enables cross-origin isolation
Cross-Origin-Embedder-Policy	Active	Required for SharedArrayBuffer
Cross-Origin-Resource-Policy	Active	Controls which origins can load resources
Cache-Control	Active	Browser and proxy caching behavior
Expect-CT	Deprecated	Was used for Certificate Transparency enforcement
Feature-Policy	Deprecated	Replaced by Permissions-Policy
X-Content-Type-Options	Legacy	Prevents MIME type sniffing
X-Frame-Options	Legacy	Clickjacking protection
X-XSS-Protection	Deprecated	No longer supported in modern browsers
Public-Key-Pins	Deprecated	Removed due to risk of lock-out
X-Permitted-Cross-Domain-Policies	Obsolete	Flash-based legacy control
Pragma	Obsolete	HTTP/1.0 cache control fallback

Table 1 : List of Security Headers Measured in This Study

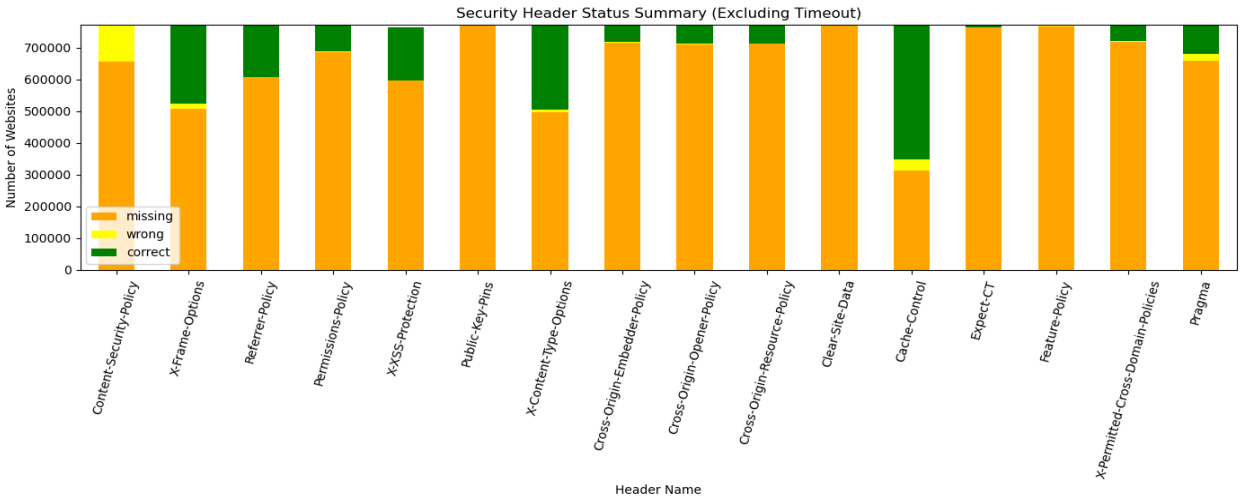

Figure 1: CSP headers usage on real-world websites

All scans were conducted over HTTPS using a fixed desktop User-Agent string. For each site, the crawler recorded the values of each header and categorized them into valid, invalid, missing, or timeout.

While most websites responded normally, 229,002 domains failed to return a response due to network timeouts or TLS errors—these were classified under the timeout category and excluded from correctness analysis to maintain accuracy in results.

Our results show that out of 1,000,000 scanned domains, only 43,250 websites deployed a Content-Security-Policy header. Among them, 1,987 enforced the policy while 1,270 used Content-Security-Policy-Report-Only. However, a large number (29,904) were configured with insecure directives such as unsafe-inline or unsafe-eval, severely weakening the effectiveness of the policy. Additionally, 13,038 of the CSP headers lacked a default-src directive, further reducing security.

Other headers showed similar trends. X-Frame-Options was present in 248,692 websites, but 15,299 included invalid values. Deprecated headers like X-XSS-Protection and Public-Key-Pins were still found in 166,361 and 267 responses respectively. Many instances of Permissions-Policy (4,408 invalid) and Referrer-Policy (218 invalid) revealed misconfigurations that nullify their intended protections.

We also observed that a significant number of websites responded with combinations of active, deprecated, and almost-deprecated headers in a single response. This indicates confusion or a lack of awareness among developers regarding current best practices.

- One website (e.g., reddit.com) responded with a malformed Referrer-Policy.
- Another (e.g., facebook.com) still served Public-Key-Pins, a deprecated and potentially dangerous header.
- Several websites (e.g., udemy.com, cloudflare.com) used both modern (Permissions-Policy) and deprecated (Feature-Policy) headers, indicating incomplete migration.

These findings confirm that while adoption of security headers is increasing, misconfiguration and outdated practices remain prevalent across the web. Our study highlights the need for better tooling, education, and standardized practices in deploying secure HTTP headers.

Goal 1: Inconsistency in Security Header Implementation

To examine inconsistencies in the implementation of HTTP security headers, we analyzed the presence and variations of 17 key headers across 1,000,000 high-traffic websites from the Majestic Million list. These headers include modern standards like Content-Security-Policy, Permissions-Policy, and Cross-Origin-Opener-Policy, as well as legacy and deprecated ones like X-XSS-Protection and Public-Key-Pins.

Our results reveal substantial inconsistency in how websites apply security headers. Some domains correctly implement only a subset of recommended headers, while others attempt to include many headers sometimes with conflicting or outdated configurations. For example:

- yelp.com was found using both modern (Content-Security-Policy) and legacy (X-Content-Type-Options, X-XSS-Protection) headers. However, its CSP included insecure directives such as unsafe-inline, which weakens its protection.
- udemy.com, godaddy.com, and cloudflare.com deployed both modern and legacy policies simultaneously, including deprecated values like Feature-Policy alongside Permissions-Policy.
- kickstarter.com and espn.com included both Permissions-Policy and Feature-Policy, reflecting partial or inconsistent migration to current standards.
- facebook.com still included the deprecated Public-Key-Pins header, even though its use has been discouraged due to risks in misconfiguration.
- reddit.com responded with a Referrer-Policy header that lacked a defined policy value, rendering it ineffective.

Moreover, we found that even when headers were present, their values were often improperly set. For instance, many CSP headers were missing the critical `default-src` directive, or included wildcards and unsafe fallbacks, undermining their purpose.

These findings highlight the fragmented and sometimes contradictory use of HTTP security headers in practice. The coexistence of active, deprecated, and malformed headers on the same website reflects a lack of standardization and confusion among developers. Without clear, unified adoption and validation mechanisms, client-side protections remain inconsistent and unreliable across the web.

Goal 2: Identify Errors in Existing Header Policies

We conducted an in-depth analysis of how security headers are configured across real-world websites, focusing particularly on whether their values conform to modern best practices and whether misconfigurations undermine the intended protection. This goal aims to answer questions such as: Do developers understand how to use security headers correctly? and How many websites are vulnerable due to incorrect or incomplete header settings?

To evaluate correctness, we referenced criteria based on documentation from MDN [2] and OWASP [1]. A header is considered valid if it is syntactically correct, aligns with current

browser expectations, and avoids insecure values. Conversely, headers are considered invalid if they:

- Contain insecure or deprecated directives (e.g., unsafe-inline, wildcard *)
- Lack required fields (e.g., default-src in CSP)
- Use legacy or unsupported values (e.g., X-XSS-Protection: 1; mode=block in modern browsers)
- Are entirely empty or malformed

Our findings are summarized below:

- Incomplete Enforcement:
  Many websites set headers only on their landing or homepage but fail to enforce them consistently across

all subpages. For example, numerous domains apply Content-Security-Policy or Referrer-Policy only at the top-level path (/) but not on nested paths or API endpoints, weakening the overall coverage.

- Deprecated and Misleading Headers:
  A significant number of websites (267) still include the deprecated Public-Key-Pins (HPKP) header, despite its removal from most modern browsers due to operational risks. Similarly, X-XSS-Protection, another outdated header, was found on 166,361 domains—many of which used values such as "0" (disabled) or unsupported modes like mode=enable.
- Non-standard or Browser-Specific Directives:
  Our analysis uncovered 4,408 cases of misconfigured Permissions-Policy headers that used wildcard
- directives or incorrect formatting. Likewise, several domains used Firefox-only directives in CSP or malformed feature names, which other browsers silently ignore.
- Non-effective CSP Policies:
  While 3,257 websites deployed CSP headers, 29,904 of them included insecure directives like unsafe-inline, unsafe-eval, or wildcard *. Additionally, 13,038 policies omitted default-src, rendering them partially ineffective. For instance, several websites allowed script execution from arbitrary domains or embedded inline scripts—negating the core purpose of CSP.
- Incorrect Fallback or Redundant Headers:
  Domains such as kickstarter.com and espn.com were found using both Permissions-Policy and its deprecated predecessor Feature-Policy, causing confusion about the active policy. Others served contradictory caching directives via Cache-Control, such as public and no-store used together.
- Empty or Ineffective Policies:
  For example, reddit.com included a Referrer-Policy header without any value, making it functionally useless. Similarly, Clear-Site-Data was present in 130 responses but lacked required values like "cookies" and "cache", thus failing to clear meaningful data.

Our findings strongly suggest that many developers do not fully understand the correct usage of HTTP security headers, leading to misconfigurations that nullify their protective intent. While adoption rates have improved over time, errors in implementation continue to provide openings for attackers through content injection, clickjacking, data leaks, and cross-origin threats.

User-Agent Configuration:

For consistency in crawling behavior, all scans were performed using a simulated Google Chrome browser (version 113), with the following User-Agent string:

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113 Safari/537.36.

Header Name	Valid Criteria	Invalid/Misconfigured Examples
Content-Security-Policy	Includes default-src; no unsafe-inline/eval/wildcards	Missing default-src, contains unsafe-inline, unsafe-eval, *
Strict-Transport-Security	max-age ≥ 15768000, with includeSubDomains	Missing max-age, too low, sent over HTTP
Permissions-Policy	Correctly formatted feature=value pairs	Wildcard (*), deprecated names, malformed structure
Referrer-Policy	One of 8 valid values (e.g., no-referrer, same-origin)	Empty value, misspelled policy
Clear-Site-Data	Includes "cache" and "cookies"	Missing required directives, malformed
Cross-Origin-Opener-Policy	same-origin or same-origin-allow-popups	unsafe-none or unknown values
Cross-Origin-Embedder-Policy	require-corp or credentialless	Other values or empty
Cross-Origin-Resource-Policy	same-origin, same-site, cross-origin	Empty, misspelled
Cache-Control	Valid directives (e.g., no-store, max-age)	Conflicting directives (e.g., public + no-store), incomplete
Expect-CT	Includes enforce and max-age	enforce without max-age, malformed
Feature-Policy	Deprecated (no valid usage in modern contexts)	Used together with Permissions-Policy
X-Content-Type-Options	nosniff	Any other value
X-Frame-Options	DENY or SAMEORIGIN	ALLOW-FROM (deprecated), others
X-XSS-Protection	Deprecated (should be absent)	mode=block or 0
Public-Key-Pins	Includes pin-sha256 and max-age	Missing required directives
X-Permitted-Cross-Domain-Policies	none, master-only, by-content-type, all	Other values
Pragma	Used with Cache-Control	no-cache without supporting header

Table 3 : Security Header Validation Criteria

Goal 3: Detect Usage of Deprecated or Obsolete Headers

As part of our study, we aimed to detect the extent to which websites continue to use HTTP security headers that have been deprecated, are no longer recommended by OWASP, or are ignored by modern browsers. Despite changes in browser support and updated standards, we observed significant presence of such headers in real-world deployments.

Our findings indicate a surprising number of domains still deploying such headers:

- Public-Key-Pins (HPKP):
  Although officially deprecated due to risks of denial-of-service through misconfiguration, the Public-Key-Pins header was found on 267 websites. Moreover, 6 of these used malformed configurations, lacking required directives like pin-sha 256 or max-age, rendering the header ineffective.
- X-XSS-Protection:
  Previously used to enable or disable browser-based XSS filtering, this header is now obsolete and unsupported in Chromium-based browsers. However, it appeared on 166,361 websites, with 7,947 explicitly disabling the filter (X-XSS-Protection: 0). Some websites still incorrectly include non-functional values such as mode=enable.
- Feature-Policy:
  This header was replaced by Permissions-Policy in modern browsers, but was still found on 5,326 websites. In some cases, both headers were served together, indicating confusion about the migration or misunderstanding of browser support.
- Expect-CT:
  The Expect-CT header, which was intended to enforce Certificate Transparency logging, has been deprecated in Chrome since version 94. Nevertheless, we detected it on 5,889 websites. Among these, 10 used an invalid form specifying enforce without a max-age — which violates the syntax expected by the specification.
- Pragma (no-cache):
  A legacy HTTP/1.0 caching control header, Pragma: no-cache was often included incorrectly or redundantly. We found 19,698 websites using Pragma: no-cache without a supporting Cache-Control directive, and 91,679 had both headers present, suggesting uncertainty about caching directives.

Evaluation Summary

This study evaluated the implementation and correctness of 17 HTTP security headers across the top 1,000,000 websites from the Majestic Million dataset. Our findings indicate that while the adoption of some modern headers—such as Content-Security-Policy and Permissions-Policy—has improved, widespread inconsistencies, misconfigurations, and use of deprecated headers persist in real-world deployments.

We identified thousands of cases where headers were either missing, malformed, or used with insecure values (e.g., unsafe-inline, unsafe-eval, wildcard sources). Legacy headers such as X-XSS-Protection and Public-Key-Pins were still actively used, despite being deprecated or unsupported by modern browsers. Furthermore, we observed several websites simultaneously deploying active and deprecated headers, which introduces ambiguity and weakens their overall security posture.

Our analysis also demonstrates that a considerable number of domains incorrectly configure security policies only on their landing pages, neglecting internal pages or API endpoints. These gaps in enforcement, along with misused caching directives and empty or incomplete values, significantly reduce the effectiveness of the protection mechanisms.

To address these issues, we developed a browser extension to assist users and developers in visualizing and understanding the security headers of any website in real-time. This tool proved practical in identifying common mistakes and educating stakeholders about best practices.

In conclusion, although security header adoption is trending upward, real-world implementations remain fragmented and often fail to meet current OWASP and browser security standards. Better tooling, clearer documentation, and increased awareness are essential to bridge this gap.

3 Check Header OWASP Extension Design

The goal of the Check Header OWASP extension is to provide users and developers with an in-browser tool to inspect, analyze, and interpret HTTP Security Headers and DOM-based security weaknesses in real time. Inspired by UserCSP, this extension extends beyond CSP to support a full range of OWASP-recommended headers while focusing on passive analysis and educational insights rather than policy enforcement.

Check Header OWASP assists users in understanding whether a website's security posture aligns with best practices by identifying missing, misconfigured, insecure, deprecated, or unsafe headers. It also incorporates a DOM analyzer that observes inline scripts, iframe behaviors, meta-based policies, and other risky patterns that suggest weak client-side protection.

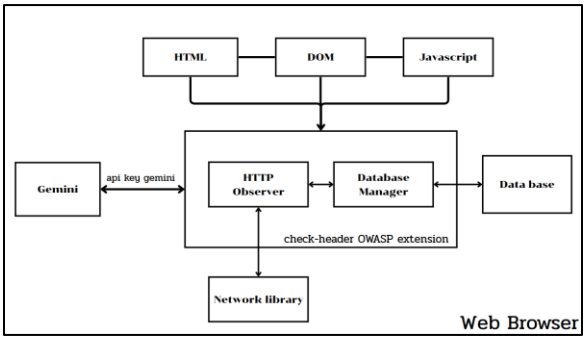

Figure 3: OWASP Extension Design

The diagram illustrates the interaction between browser components, the HTTP observer, database manager, and Gemini integration for security analysis and explanation.

Architecture Components:

- HTTP Observer (background.js): Listens to all HTTP response headers using the chrome.webRequest.onHeadersReceived API. Captures headers for each visited domain and stores them in chrome.storage.local.
- DOM Analyzer (content.js + domAnalyzer.js): Injected into every webpage to scan inline scripts, inline event handlers, iframe usage, cross-origin resources, meta tags (e.g., pragma, cache-control), and CSP-violating constructs. Results are processed and classified into security states. The DOM Analyzer not only detects risky patterns but also provides an inline "Fix" mechanism. When unsafe constructs such as script tags with inline JavaScript or iframe elements with missing sandbox attributes are detected, the extension suggests and optionally applies safer alternatives. This DOM Fixing capability is designed to be educational, reversible, and compliant with OWASP recommendations.
- Popup UI (popup.html + popup.js): Presents the header and DOM results visually, highlighting status (e.g., correct, insecure, missing, deprecated). Gemini LLM is optionally available to translate technical results into user-friendly explanations.
- LLM Integration (Gemini API): Enables AI-based analysis of headers for explanatory purposes, particularly useful for non-technical users.

3.1 Header and DOM Behavior

The Check Header OWASP extension does not enforce or rewrite any security policies. Instead, it passively observes and evaluates HTTP headers alongside the runtime DOM structure to assess security posture. Each detected header is validated based on OWASP and MDN guidance, and its presence or configuration is correlated with observable DOM behaviors such as inline scripts, iframe usage, and meta directives.

When a valid header is present and the DOM reflects secure practices, the site is marked as secure. If a header is present but configured insecurely (e.g., allowing unsafe-inline in CSP), or if the DOM reveals high-risk patterns, the extension flags the configuration as insecure. If a critical header is missing and risky behaviors are observed in the DOM, the site is flagged as vulnerable and the header is marked as required. Deprecated headers and timeout scenarios are also identified to provide a complete view of the security landscape.

3.2 Automatic Analysis Challenges

Analyzing HTTP Security Headers and DOM behavior across real-world websites presents several challenges, especially in dynamic and modern web environments:

- Security headers may be present but still allow risky behaviors (e.g., CSP with unsafe-inline)
- Inconsistencies between meta tags and actual HTTP headers (e.g., cache-control vs <meta pragma>)
- Runtime JavaScript injections from third-party ads or scripts altering security posture post-load
- Variations in headers due to redirects, inconsistent CDN behavior, or load balancing
- Use of obfuscated code, shadow DOM, or client-side frameworks that hide security-relevant behaviors

To address these issues, the extension:

- Normalizes and analyzes header values using multi-condition logic
- Parses live DOM for patterns like inline scripts, iframes, event handlers, and meta-based directives
- Correlates DOM findings with missing or weak headers to assess true risk exposure
- Flags deprecated or misleading headers even if syntactically correct
- Detects header absence due to timeout, redirect chain failures, or HTTPS downgrade

This dual-layer approach evaluating both raw headers and runtime page structure helps reveal misconfigurations and weak practices that may not be visible from headers alone.

3.3 User Behavior and Storage

The Check Header OWASP extension is designed to be non-intrusive and educational. It does not modify or enforce any security policies. Instead, it passively observes each visited website, evaluates HTTP headers and DOM content, and stores the results temporarily using local browser storage. The extension does not retain persistent data unless the user explicitly exports it. The popup interface emphasizes explanation-first interaction, enabling users to understand security issues in plain language through visual indicators and optional AI assistance.

3.4 Visual Interface

The extension includes a lightweight and intuitive popup UI that allows users to review results at a glance. The UI is divided into three key sections:

- Header Result Table: Shows each detected HTTP Security Header, its validation status, and an explanation or reason.
- DOM Analysis Table: Shows DOM-related risks such as presence of inline scripts, iframe usage, or missing referrer/meta directives.
- Gemini AI Integration Panel: Allows the user to input and save their Gemini API Key, and receive natural-language explanations and recommendations for any misconfigurations detected.

Figure 4 below shows flowchart Extension used to present results.

Figure 5 below shows a screenshot of the popup interface used to present results.

The extension is available as an open-source project for reproducibility and further development.
Repository: https://github.com/Fourgame/check-headers-extension

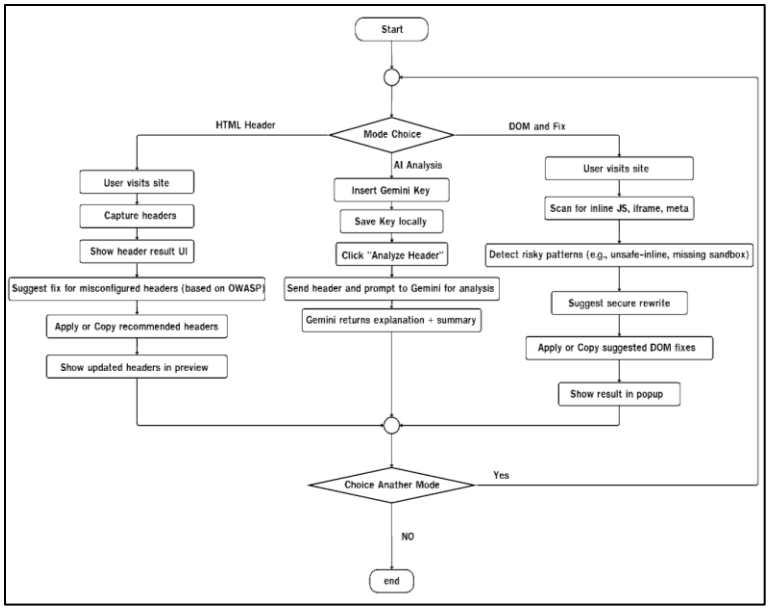

Figure 4: Comprehensive user flow of the Check Header OWASP extension.

This diagram illustrates the complete decision and interaction paths across three modes—HTTP Header Analysis, DOM & Fix, and AI Explanation via Gemini. Users can switch between modes or exit after completing any flow.

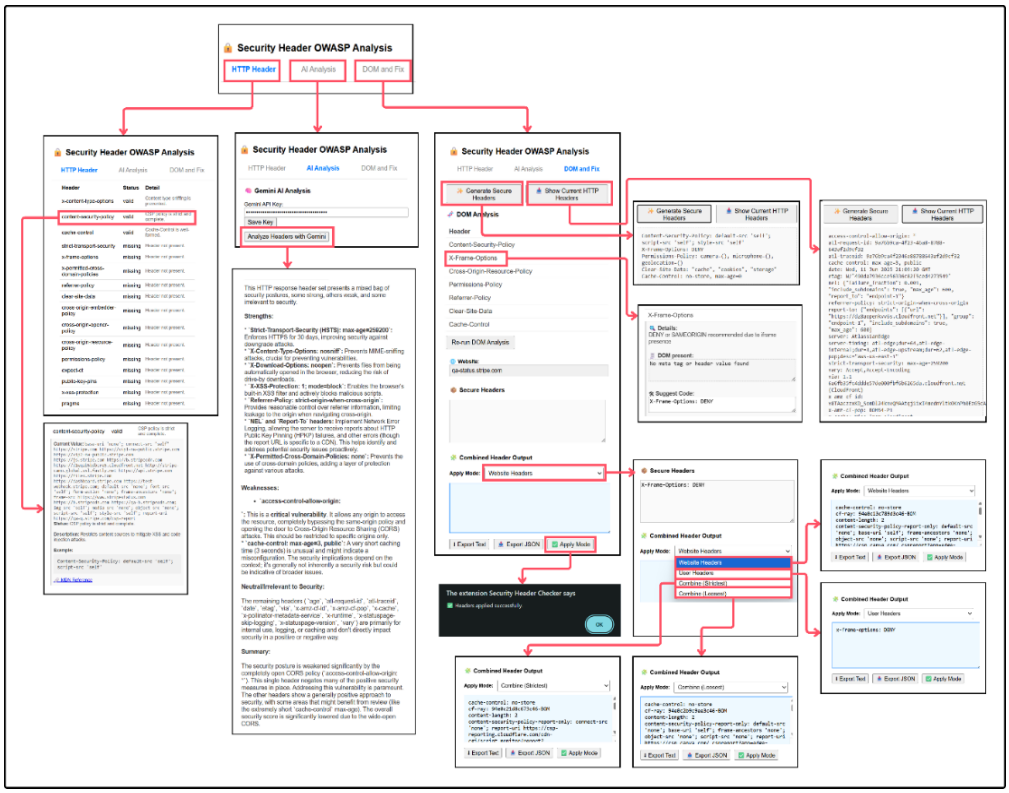

Figure 5: Feature overview of the Check Header OWASP extension

The diagram illustrates the interaction flow between HTTP header analysis, AI-powered explanation, and DOM-based detection and fixing. Users can inspect live HTTP headers, receive Gemini-generated security recommendations, and apply safe DOM rewrites such as adding sandbox attributes or removing unsafe inline scripts directly from the interface.

4 Evaluation of Check Header OWASP Extension

To evaluate the Check Header OWASP extension, we applied it to 1 million websites from the [Majestic Million dataset] to measure its ability to identify the presence, misconfiguration, and weakness of 17 HTTP Security Headers. Of these, 229,002 websites resulted in timeouts or inaccessible responses, leaving 770,998 domains with analyzable header and DOM content. The extension collected HTTP header data via chrome.webRequest and dynamically analyzed DOM elements using injected scripts.

4.1 Header Detection Accuracy

The extension was able to detect and classify security headers with high precision. For instance, among the 770,998 reachable

All scans were conducted over HTTPS using a fixed desktop User-Agent string.

User-Agent Configuration: For consistency in crawling behavior, all scans were performed using a simulated Google Chrome browser (version 113), with the following User-Agent string:

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113 Safari/537.36.

4.2 DOM-based Security Weakness Detection

The DOM analysis module was able to detect

- Inline <script> tags without CSP protection
- Usage of unsafe-inline, eval(), and on* inline event handlers
- Unprotected <iframe> elements and risky cross-origin inclusions
- Conflicting <meta> tags such as pragma or cache-control that override secure HTTP headers

These findings reinforced the conclusion that HTTP headers alone are insufficient and must be supplemented with client-side analysis.These results indicate that more than 75% of scanned websites either lack key headers or have insecure configurations.

4.3 Performance and Coverage

The extension operated efficiently within Chrome's developer tools, injecting scripts and processing DOM elements with minimal performance overhead. Results were temporarily stored in browser memory and could be optionally exported. A built-in Gemini LLM integration enabled AI-generated explanations to assist non-technical users in interpreting the results.

Summary: The evaluation demonstrated that Check Header OWASP is a practical, scalable, and accurate tool for analyzing security headers and DOM-based vulnerabilities. By excluding 229,002 unreachable websites, the study focused on 770,998 valid responses, revealing systemic weaknesses in header adoption and secure DOM practices. The extension's integration of live analysis and AI-powered explanation provides a strong foundation for real-time web security auditing.

Header Name	Correct	Wrong/Misconfigured	Missing	Deprecated	Notes
Content-Security-Policy	535	42942	656750		Mostly unsafe-inline, no default-src
X-Frame-Options	248692	15299	507007	✔	Some use ALLOW-FROM (deprecated)
Referrer-Policy	163886	218	606894		Some missing entirely or empty
Permissions-Policy	81532	4408	685058		Wildcard (*) usage and malformed entries
Cross-Origin-Embedder-Policy	53540	3280	714178		Expected: require-corp / credentialless
Cross-Origin-Opener-Policy	59720	2685	708593		Some sites used unsafe-none
Cross-Origin-Resource-Policy	58033	415	712550		Misuse of unknown values
Clear-Site-Data	1	130	770867		Missing 'cookies' or 'cache'
Cache-Control	423289	33780	313246		Multiple max-age, public+no-store conflict
Expect-CT	5889	10	765099	✔	Deprecated, used without max-age
Feature-Policy		5326	765672	✔	Deprecated; replaced by Permissions-Policy
X-Content-Type-Options	267110	7867	496021		Should be 'nosniff' only
X-XSS-Protection	166361	7947	596690	✔	Deprecated and mostly disabled
Public-Key-Pins	267	6	770725	✔	Deprecated due to lock-out risk
X-Permitted-Cross-Domain-Policies	51354	1170	718474		Legacy Flash-based control
Pragma	91679	19698	659621	✔	Obsolete; no-cache used without cache-control
Strict-Transport-Security	420212	14630	565158		max-age too low or missing

Table 2: Security Header Validation Summary

5 Related Work

Numerous studies have investigated the adoption and effectiveness of HTTP security mechanisms across the modern web. Much of the early focus was on individual headers such as X-Frame-Options, X-Content-Type-Options, and Strict-Transport-Security, which were introduced to prevent specific vulnerabilities including clickjacking, MIME sniffing, and protocol downgrade attacks.

As the web evolved, newer mechanisms like Content-Security-Policy, Permissions-Policy, and Referrer-Policy were introduced to address broader security concerns, including cross-site scripting (XSS), feature abuse, and privacy leakage. Despite browser support and strong recommendations by organizations like OWASP and Mozilla, many websites still fail to configure these headers correctly or neglect to deploy them entirely.

Browser extensions and developer tools have been proposed to assist in the visualization or enforcement of such policies. However, most tools either lack DOM-level analysis, fail to detect misconfigurations, or provide insufficient explanations for non-expert users. Furthermore, studies rarely examine all active, legacy, and deprecated headers together, leaving gaps in understanding the full landscape of security posture on the web.

This research addresses that gap by offering a unified, real-time evaluation of 17 security headers including both current and deprecated ones across a large dataset of 1 million websites. The browser extension developed as part of this project further enhances practical usability through DOM analysis and AI-powered insights, supporting a more complete and user-friendly understanding of web security.

6 Conclusion

This paper presents a large-scale empirical study of HTTP security headers on the top 1 million websites, evaluating both adoption and correctness across 17 key headers. Our findings show that while some headers like Strict-Transport-Security and X-Frame-Options see moderate adoption, critical headers such as Content-Security-Policy and Permissions-Policy are often missing, misconfigured, or weakened by insecure values like unsafe-inline.

Additionally, many websites still use deprecated headers—such as Public-Key-Pins and X-XSS-Protection—despite their removal from browser specifications. Inconsistencies, such as mixing modern and legacy headers, were widespread and suggest limited understanding of current security best practices.

To support better adoption, we introduced the Check Header OWASP browser extension, which offers real-time analysis of HTTP headers and DOM behavior directly in the browser. The extension uses chrome.webRequest and injected scripts to examine live responses and client-side content. Combined with AI-generated explanations, it enables developers and analysts to detect vulnerabilities quickly and understand the risks in context.

After excluding unreachable sites due to timeouts, the study successfully analyzed 770,998 domains, offering detailed statistics on each header’s usage and misconfiguration. This comprehensive view highlights the need for better tooling, awareness, and simplified feedback mechanisms to guide secure web development.

In conclusion, while progress has been made in security header adoption, the real-world implementation remains error-prone. Tools that combine automation, real-time inspection, and accessible feedback—such as the one proposed here—are essential to improving security posture at scale.

References

[1] OWASP Foundation. "OWASP Secure Headers Project." [Online]. Available: https://owasp.org/www-project-secure-headers/

[2] Mozilla Developer Network. "HTTP Headers." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

[3] Majestic. "The Majestic Million." [Online]. Available: https://majestic.com/reports/majestic-million

[4] Google AI. "Gemini Language Model." [Online]. Available: https://ai.google.dev/gemini

[5] Barth, A. "The Web Origin Concept." W3C Recommendation. [Online]. Available: https://www.w3.org/TR/weborigin/

[6] Nattapol Prairuenrom. "Check Header OWASP – GitHub Repository." [Online]. Available: https://github.com/Fourgame/check-headers-extension