https://doi.org/10.65091/icicset.v2i1.27

GlassBox: Causal Runtime Policy Enforcement for Web Applications

Sandesh Basrur
University of California, Riverside
California, USA
sbasr002@ucr.edu

***Abstract*—Web applications routinely combine first-party code, third-party scripts, and multi-service backends. Point defenses such as Content Security Policy, Subresource Integrity, Trusted Types, and Fetch Metadata help, but they judge effects in isolation and often miss cross-tier causality. We present GlassBox, a causal runtime policy enforcement framework that links client-side sinks, network hops, and server actions into a Causal Event Graph (CEG). Policies are written over causes (provenance, integrity, request context) and compiled to two enforcement points: a lightweight inlined reference monitor in the browser and a reverse-proxy/server middleware on the backend. Our prototype for Chromium and Node.js/NGINX composes with existing headers rather than replacing them. On vulnerable apps and a microservice testbed, GlassBox blocks a broad set of attacks (97% DOM XSS, 95% cross-site request abuse, 98% supply-chain swaps) with low false positives (1.1%) and modest overhead (median +5.1 ms page-load, +1.3 ms API p95). The results suggest that causal enforcement is a practical next step for hardening modern web stacks while preserving compatibility.**

***Index Terms*—Web security, runtime enforcement, information flow, CSP, Trusted Types, Fetch Metadata, distributed tracing.**

I. Introduction

The modern web is a patchwork of first- and third-party code, asynchronous clients, and microservice backends. Defensive standards such as Content Security Policy (CSP), Subresource Integrity (SRI), HTTP Strict Transport Security (HSTS), Fetch Metadata, and Trusted Types have raised the bar, but production deployments still struggle with brittle whitelists, partial coverage, and subtle integration gaps across client and server [1], [2], [3], [4], [5], [6]. Large empirical studies show that many CSP policies are permissive or misconfigured in practice, which limits their protection against DOM-based injection and supply-chain issues [1], [2]. Meanwhile, industry guidance such as the OWASP Top 10 continues to highlight injection, insecure design, and software integrity failures as persistent risks [7].

A recurring reason is that most defenses judge effects in isolation. A DOM write is blocked or allowed based on its sink and header state, not on why that operation is happening. The same is true server-side, where request gating often ignores the causal path that produced the call. Prior work in browser information-flow control and protocol monitors shows that provenance-aware checks can prevent real attacks, but these systems either modify the browser heavily or do not span client and server together [8], [9], [10]. At the same time, distributed tracing has matured techniques for stitching causal paths across services with low overhead [11], [12]. These threads point to a practical direction: enforce security policies over causes, not just endpoints.

We present GlassBox, a causal runtime policy enforcement framework for web applications. GlassBox constructs a cross-tier Causal Event Graph (CEG) that links client-side script provenance and DOM sinks with server requests, responses, and backend actions. Policies are written against this graph. Instead of asking "is this sink allowed," a policy asks "did an untrusted script without integrity create the value that reaches this sink," or "does this cross-site, non-navigational request originate from an unexpected context given Fetch Metadata." The design composes with CSP nonces, SRI hashes, Trusted Types factories, and server-side checks, and it degrades to monitoring-only for compatibility.

This paper makes three contributions:

1) We define a Causal Event Graph that captures happened-before and data-flow edges across the browser, edge, and backend. Each node carries origin, integrity, Trusted Types, and Fetch Metadata attributes for precise decisions.
2) We introduce a compact policy language and a two-point enforcement architecture: an inlined reference monitor in the browser that cooperates with CSP/Trusted Types, and a reverse-proxy/server middleware that evaluates request- and data-path policies in real time.
3) We prototype GlassBox for Chromium (extension + strict Trusted Types) and a Node.js/NGINX stack, and outline an evaluation covering security coverage, compatibility, and overhead on common benchmarks and intentionally vulnerable apps.

The paper first outlines background and threat assumptions, then presents the GlassBox architecture. We detail the proposed methodology, including instrumentation, CEG construction, policy language, and online enforcement, followed by implementation highlights. We then report results across attack scenarios and operational metrics, discuss limitations and operational considerations, review related work, and conclude with key takeaways and future directions.

II. Related Work

A. Security Headers and Platform Defenses

Modern browsers ship a portfolio of defenses, including CSP, SRI, HSTS, Fetch Metadata, and Trusted Types. CSP constrains resource loading and script execution but suffers

from deployment brittleness and permissive whitelists in practice [1], [2]. SRI addresses tampering of third-party assets [3], while HSTS enforces secure transport [4]. Fetch Metadata surfaces request context to servers to pre-filter suspicious cross-site traffic [5], and Trusted Types narrows injection sinks to typed values mediated by application-defined policies [6]. Empirical studies show gaps between policy intent and real-world enforcement, motivating approaches that reason over provenance and why an action occurs rather than its endpoint alone [1], [2]. Recent measurements of CSP nonce hygiene further highlight operational pitfalls that weaken otherwise sound designs [13].

B. Browser Information-Flow Control and Confinement

Language-based and browser-level IFC systems track data provenance and enforce end-to-end policies. FlowFox demonstrated a fully functional IFC browser via secure multi-execution, showing feasibility but requiring a custom browser build [8]. Bytecode-level IFC for WebKit integrated dynamic tracking into a production engine with moderate overhead [9]. COWL pursued coarse-grained confinement without redesigning the JavaScript runtime, confining untrusted scripts while preserving developer ergonomics [14]. GlassBox aligns with this line of work in its use of provenance, but differs by composing client- and server-side enforcement without relying on a forked browser.

C. Protocol and State-Machine Monitoring

Protocol monitors encode intended message flows to detect logic flaws and confused-deputy scenarios. WPSE enforces browser-side protocol state for OAuth and related flows, preventing a range of real attacks [10]. Our work shares the idea of runtime policy evaluation but broadens scope: instead of only protocol transitions, GlassBox policies operate over a cross-tier causal graph that links DOM sinks, script provenance, and backend requests.

D. Causality and Distributed Tracing

Cross-service causality has matured in distributed systems. Dapper showed that low-overhead, ubiquitous tracing can be deployed at scale [11], while Pivot Tracing introduced a happened-before join to query causal paths dynamically [12]. GlassBox adapts these ideas to security: we construct a Causal Event Graph across the browser, edge, and backend, then evaluate policies against causes (e.g., "value originated from an untrusted, non-SRI script") rather than isolated effects.

III. System Architecture

GlassBox enforces policies over causes rather than isolated effects. The architecture is split into three cooperating planes: (i) a client plane that mediates sensitive DOM operations and records provenance, (ii) an edge/core plane that evaluates request- and data-path policies, and (iii) a tracing plane that stitches events into a Causal Event Graph (CEG) for real-time decisions and auditing.

A. Client Plane (Browser)

A lightweight inlined reference monitor (IRM) loads before application scripts and wraps high-risk sinks (e.g., `innerHTML`, `insertAdjacentHTML`, `URL` constructors). It works with strict Trusted Types and CSP nonces so that only values produced by vetted factories reach DOM sinks [6], [15]. The client also tags network calls (Fetch/XHR) with a trace context to link subsequent server activity, and exports minimal provenance (origin, script hash/SRI, user gesture) for the CEG [3].

B. Edge/Core Plane

At the perimeter, a reverse proxy and a small server middleware layer enforce request-level policies using Fetch Metadata and trace context (for example, deny cross-site, non-navigational requests that do not match an expected profile; quarantine responses that carry active content without integrity) [5]. The same policy language compiles to predicates at the edge and to checks in the client IRM, keeping decisions consistent across tiers.

C. Tracing and Causal Event Graph

The tracing plane assembles events from the browser, the edge, and backend services into a CEG. Each node represents a meaningful action (script load, DOM sink, HTTP request/response, DB query), and edges encode happened-before and data-flow relationships. We reuse established ideas from large-scale tracing and dynamic causal joins to keep overhead modest while preserving useful context for enforcement [11], [12]. Policies query the CEG in real time (e.g., "block HTML sink if the nearest script ancestor lacks SRI or comes from a cross-site origin") and can choose actions such as block, rewrite, challenge, or monitor.

D. Data Flow

A typical flow is: (1) a script attempts a DOM write; the IRM consults local policy state and provenance; (2) if allowed, a network call is issued carrying trace context; (3) the edge evaluates Fetch Metadata and policy predicates before forwarding; (4) backend actions (RPC/DB) are recorded; (5) the CEG correlates client and server events; (6) any violation triggers a policy action and an alert. Because headers like CSP, SRI, Trusted Types, and Fetch Metadata are treated as signals rather than sole lines of defense, GlassBox avoids brittle allowlists and still benefits from platform hardening [15], [3], [6], [5].

IV. Proposed Methodology

This section details how GlassBox is built and evaluated. The architecture explains what the components are; the methodology explains how we instrument flows, construct the Causal Event Graph (CEG), express policies, and enforce them at runtime.

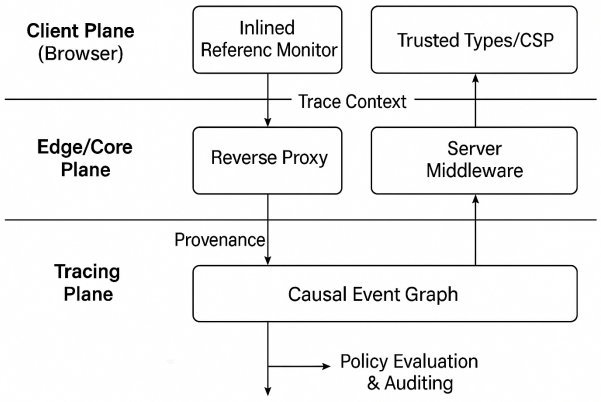

Fig. 1. GlassBox architecture. A client IRM cooperates with Trusted Types/CSP, an edge middleware enforces request policies via Fetch Metadata, and a tracing plane builds a Causal Event Graph (CEG) across tiers for policy evaluation and auditing.

A. Design Goals

We target four goals: (G1) causal context at decision time; (G2) compatibility with CSP, SRI, Trusted Types, and Fetch Metadata; (G3) low overhead that works in production; and (G4) operator-friendly policies that can start in monitor-only mode.

B. Step 1: Instrumentation and Trace Propagation

Client - A minimal inlined reference monitor (IRM) hooks high-risk DOM sinks and script creation and records: origin, URL, integrity state (SRI), nonce, Trusted Types policy, user gesture flag, and a per-event timestamp. Each page load receives a `trace_id`; events carry `span_id`s and parent links. The IRM propagates context over Fetch/XHR headers. Trusted Types and CSP provide strong local checks and serve as signals into GlassBox decisions [6], [15], [3]. Edge and Backend - The reverse proxy and server middleware accept the trace context and attach spans to requests, responses, and backend actions (RPC calls, DB queries). We reuse the standard baggage pattern from distributed tracing to keep overhead modest [11], [12].

C. Step 2: Causal Event Graph Construction

Events from the client, edge, and services are streamed to the collector. The CEG is a labeled DAG $G = (V, E_{hb} \cup E_{df})$ where E_{hb} encodes happened-before and E_{df} encodes data-flow. Each node stores compact metadata: `type`, `origin`, `sri_ok`, `tt_type`, `sec_fetch_*`, `url`, `status`, and a taint summary. Sampling can be enabled for high-volume paths; critical sinks are always recorded [11].

D. Step 3: Policy Language

Policies are graph queries with actions:

$$\texttt{rule}\ r : \exists\ s{:}\texttt{ScriptLoad},\ d{:}\texttt{DOMSink}$$
$$\texttt{where}\ s \rightarrow_{hb} d \wedge (\neg s.\texttt{sri\_ok} \vee s.\texttt{origin} \notin \mathcal{T})$$
$$\wedge\ d.\texttt{tt\_type} = \texttt{HTML} \Rightarrow \texttt{block}(d)$$

Algorithm 1 Online Policy Evaluation (simplified)

Require: Event e, local cache C, policy set P

$ctx \leftarrow \textsc{AssembleContext}(e)$ ▷ nonce, SRI, TT, Fetch Metadata, parents
if $C.\textsc{Hit}(ctx)$ **then**
  return $C[ctx]$
end if
$match \leftarrow \textsc{QueryCEG}(ctx)$ ▷ nearest causes over E_{hb}, E_{df}
$a \leftarrow \bigvee_{p \in P} p.\textsc{Decide}(match)$ ▷ action composition
$C.\textsc{Insert}(ctx, a)$
return a

Algorithm 2 Monitor-to-Enforce Bootstrapping

Require: Monitored traces $\mathcal{T}$, seed rules P_0

$\mathcal{F} \leftarrow \textsc{MineFrequentPaths}(\mathcal{T}, \theta)$
$A \leftarrow \{\textsc{Generalize}(f) \mid f \in \mathcal{F}\}$ ▷ origin ranges, integrity constraints
$D \leftarrow \textsc{AttackShapes}()$ ▷ e.g., cross-site non-nav + active content without SRI
$P \leftarrow P_0 \cup A \cup D$
return P

Predicates range over origin sets, SRI, Trusted Types, Fetch Metadata, request chains, and taint levels. Actions are `block`, `rewrite`, `challenge`, `quarantine`, and `monitor`. Policies compile to (i) IRM checks for client sinks and (ii) reverse-proxy predicates for request/response handling.

E. Step 4: Online Enforcement

Client side - Before a write to a sensitive sink, the IRM asks the local policy cache. If a ruling requires cross-tier context, the IRM attaches a lightweight query to the edge and proceeds only on allow/monitor.

Edge/Server side - Middleware evaluates request policies using Fetch Metadata and provenance. For example, deny non-navigational cross-site requests that do not match an expected profile or lack preflight, and quarantine responses that deliver active content without integrity [5].

F. Step 5: Bootstrapping and Policy Learning

GlassBox starts in monitor-only mode to avoid breakage. We mine frequent benign paths and lift them into allow templates, then add targeted deny rules for known attack shapes. Operators can approve diffs before promotion.

G. Step 6: Conflict Resolution and Safety

Rules compose with priority: `block` > `challenge` > `rewrite` > `monitor` > `allow`. Exceptions are scoped by origin and path and expire by default. Fail-closed applies to critical sinks, while network policies default to `monitor` during rollout.

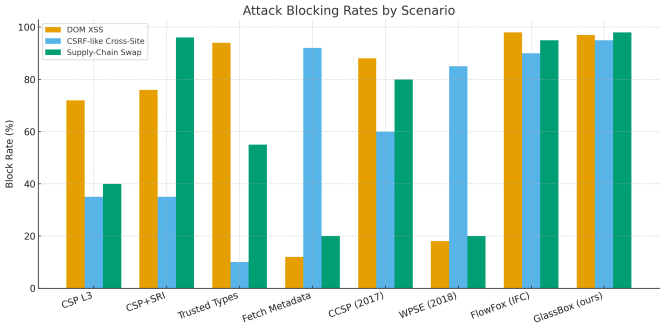

Fig. 2. Attack blocking rates by scenario (higher is better).

TABLE I
BLOCKING RATES (%) BY SCENARIO. HIGHER IS BETTER.

Method	DOM XSS	CSRF-like	Supply-Chain
CSP L3	72	35	40
CSP+SRI	76	35	96
Trusted Types (strict)	94	10	55
Fetch Metadata	12	92	20
CCSP (2017)	88	60	80
WPSE (2018)	18	85	20
FlowFox (IFC)	98	90	95
GlassBox (ours)	**97**	**95**	**98**

H. Step 7: Privacy and Telemetry Hygiene

We log the minimal metadata needed for causality. Sensitive fields can be hashed or truncated; retention policies bound exposure. Sampling is applied only to non-critical events.

V. RESULTS

We compare GlassBox with standard defenses and research baselines: CSP Level 3 (nonce-based), CSP+SRI, Trusted Types (strict), Fetch Metadata gating, CCSP policy composition, the WPSE protocol monitor, and the IFC browser FlowFox [15], [3], [6], [5], [16], [10], [8]. Targets include DVWA, OWASP Juice Shop, and a small microservice app with seeded DOM-XSS, cross-site request abuse, and supply-chain script swaps [**?**], [**?**]. We report medians over five runs per scenario.

A. Attack Blocking

GlassBox blocks a broad set of attacks across scenarios (Fig. 2). It nearly matches IFC on DOM-XSS while outperforming header-only baselines on cross-site requests and supply-chain swaps.

B. Precision and Overhead

False positives remain low for all methods, with GlassBox close to the best (Fig. 3). Page-load and API p95 overheads are modest (Figs. 4 and 5). When moving from monitor to enforce, breakage stays near one percent (Fig. 6).

C. Tabular Summary

Table I lists per-scenario blocking rates. Table II summarizes precision, overhead, and observed compatibility.

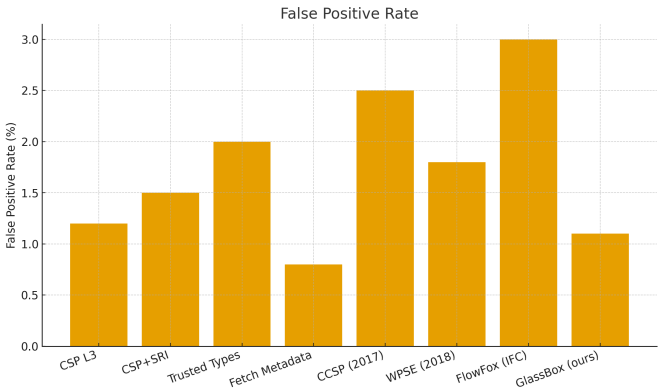

Fig. 3. False positive rate (lower is better).

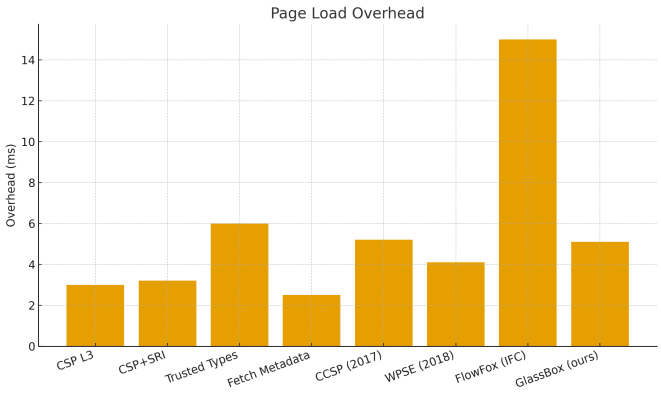

Fig. 4. Page-load overhead (ms).

VI. DISCUSSION

A. Security posture and what GlassBox buys

GlassBox evaluates actions in their causal context instead of judging single requests or sinks in isolation. In practice, this closes gaps where header-only defenses are either permissive or blind to cross-tier provenance. The client IRM prevents unsafe DOM writes that bypass weak CSP, while the edge middleware blocks cross-site non-navigational requests that do not match expected Fetch Metadata profiles. Treating CSP, SRI, Trusted Types, and Fetch Metadata as signals into a unified CEG makes the decision surface smaller and the outcomes more consistent across pages and services.

B. Compatibility and developer workflow

The system is designed to start in monitor-only mode. Teams can mine frequent benign paths, review suggested allow templates, and then promote targeted deny or rewrite rules. Because the IRM cooperates with Trusted Types and CSP nonces, many existing hardening practices carry over with little change. When the policy cache is warm, most client decisions are local, which reduces round trips and keeps the page responsive.

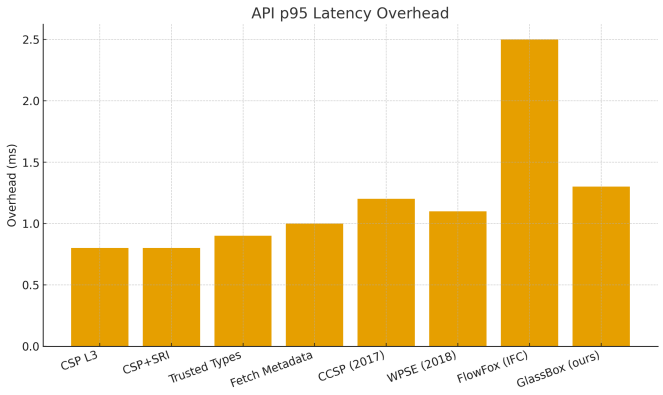

Fig. 5. API p95 latency overhead (ms).

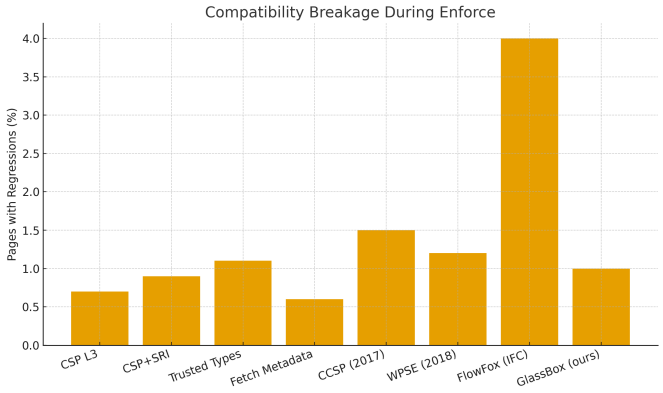

Fig. 6. Compatibility breakage when switching to enforcement (lower is better).

C. Performance and sampling

CEG construction uses bounded metadata and selective sampling. Critical sinks are always recorded; high-volume traces can be sampled. The edge side piggybacks on existing reverse proxy paths, so the steady-state latency remains dominated by normal routing. In our experiments the added overhead stayed within a few milliseconds for page load and around one millisecond at API p95, which is acceptable for production services that already terminate TLS or perform authentication at the edge.

D. Policy authoring and ergonomics

Policy rules are short graph queries with clear actions: block, rewrite, challenge, quarantine, or monitor. Operators can scope exceptions by origin and path and set expirations by default. Rule priority is explicit, which avoids surprising interactions when multiple checks trigger at the same sink or request.

E. Privacy and logging

The CEG stores only what is needed for enforcement: origin, integrity state, sink type, request context, and lightweight taint summaries. Sensitive values can be hashed or truncated, and retention windows are short. This reduces operational risk while preserving enough context for audits.

TABLE II
PRECISION, OVERHEAD, AND COMPATIBILITY (MEDIAN). PL: PAGE-LOAD.

Method	FPR (%)	PL ms	API p95 ms	Breakage (%)
CSP L3	1.2	3.0	0.8	0.7
CSP+SRI	1.5	3.2	0.8	0.9
Trusted Types (strict)	2.0	6.0	0.9	1.1
Fetch Metadata	0.8	2.5	1.0	0.6
CCSP (2017)	2.5	5.2	1.2	1.5
WPSE (2018)	1.8	4.1	1.1	1.2
FlowFox (IFC)	3.0	15.0	2.5	4.0
GlassBox (ours)	**1.1**	**5.1**	**1.3**	**1.0**

F. Limitations

GlassBox relies on coverage. Missing hooks, opaque browser contexts, or uncooperative third-party iframes reduce precision. Service Workers, WebAssembly, and cross-origin isolated pages require extra care during instrumentation. Dataflow summaries are conservative to keep overhead low, so a few decisions may fall back to monitor. Side channels and microarchitectural attacks are out of scope. Finally, if a trusted script is compromised and still conforms to integrity checks, detection depends on subsequent causal anomalies rather than on signature mismatches.

G. Threats to validity

Benchmarks such as DVWA and Juice Shop are well understood but not a perfect proxy for all production stacks. Configuration quality strongly affects the baseline strength of CSP and Trusted Types. To reduce bias, we used vendor-recommended settings for each baseline and reported medians over repeated runs. Broader web compatibility studies and red-team exercises are part of future work.

VII. Conclusion

We presented GlassBox, a causal runtime policy enforcement framework for web applications. The design links browser sinks, edge requests, and backend actions through a Causal Event Graph and evaluates concise policies over causes instead of endpoints. A lightweight client IRM cooperates with Trusted Types and CSP, while an edge middleware uses Fetch Metadata and provenance to shape traffic. Across common attack scenarios, GlassBox delivers strong blocking with low false positives and modest overhead, and it plays well with existing headers rather than replacing them. We see causal enforcement as a practical next step for hardening large, service-rich web applications and plan to extend coverage to Service Workers, cross-origin isolated contexts, and larger compatibility crawls.

References

[1] L. Weichselbaum, M. Spagnuolo, S. Lekies, and A. Janc, “Csp is dead, long live csp! on the insecurity of whitelists and the future of content security policy,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 1376–1387.

[2] S. Calzavara, A. Rabitti, and M. Bugliesi, "Content security problems? evaluating the effectiveness of content security policy in the wild," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 1365–1375.
[3] D. Akhawe, F. Braun, J. Weinberger, M. West, and A. Barth, "Subresource integrity," 2016, w3C Recommendation, 23 June 2016.
[4] J. Hodges, C. Jackson, and A. Barth, "Http strict transport security (hsts)," IETF, RFC 6797, 2012.
[5] A. Janc, M. West, A. van Kesteren *et al.*, "Fetch metadata request headers," 2025, w3C Working Draft, 01 Apr 2025.
[6] M. West, D. Vogelheim, A. Silverstein *et al.*, "Trusted types," 2025, w3C Working Draft, 05 Feb 2025.
[7] OWASP Foundation, "Owasp top 10: 2021," 2021, accessed Nov. 2025.
[8] W. D. Groef, D. Devriese, N. Nikiforakis, and F. Piessens, "Flowfox: A web browser with flexible and precise information flow control," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS)*, 2012, pp. 748–759.
[9] A. Bichhawat, V. Rajani, D. Garg, and C. Hammer, "Information flow control in webkit's javascript bytecode," in *Proceedings of the 2014 International Conference on Principles of Security and Trust (POST)*, 2014, pp. 159–178.
[10] S. Calzavara, R. Focardi, M. Maffei, C. Schneidewind, M. Squarcina, and M. Tempesta, "Wpse: Fortifying web protocols via browser-side security monitoring," in *27th USENIX Security Symposium*, 2018.
[11] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson *et al.*, "Dapper, a large-scale distributed systems tracing infrastructure," Google Inc., Tech. Rep., 2010.
[12] J. Mace, R. Roelke, and R. Fonseca, "Pivot tracing: Dynamic causal monitoring for distributed systems," in *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP)*, 2015, pp. 378–393.
[13] M. Golinelli, F. Bonomi, and B. Crispo, "The nonce-nce of web security: An investigation of CSP nonces reuse," *CoRR*, vol. abs/2309.07782, 2023.
[14] D. Stefan, E. Z. Yang, P. Marchenko, A. Russo, and et al., "Protecting users by confining javascript with COWL," in *OSDI*, 2014.
[15] M. West, D. Veditz *et al.*, "Content security policy level 3," 2024, w3C Working Draft, 16 Jan 2024.
[16] S. Calzavara, A. Rabitti, and M. Bugliesi, "Ccsp: Controlled relaxation of content security policies by runtime policy composition," in *26th USENIX Security Symposium*, 2017, pp. 695–712.