The Web Needs a Logout Button: Moving Beyond Document Semantics on the Web

James McCauley
Mount Holyoke College
South Hadley, MA, USA

Abstract

As a platform, the web has evolved largely through the acquisition of capabilities represented by APIs with low-level, mechanistic semantics. This trajectory—codified by the 2013 *Extensible Web Manifesto (EWM)*—has enabled a world-changing ecosystem, but it has also created significant architectural debt by lacking native representation for pervasive, user-meaningful concepts. This has forced developers to endlessly reinvent concepts like authentication, e-commerce, and cookie consent within the "semantic black hole" of insecure, highly fragmented, and performance-degrading site-specific JavaScript. We argue that the web must reverse this course and elevate common, high-level, user-meaningful semantics to trustworthy, universal platform primitives. However, forcing browsers to natively implement such complex semantics risks recreating the exact design tension that the EWM sought to avoid. To sidestep this, we further propose an entirely new axis for semantic expression within the web platform. We argue that this resolves the impasse while simultaneously giving stakeholders from privacy advocates to commercial platforms new capabilities for shaping the web, and ultimately granting new agency to users.

CCS Concepts

• **Information systems** → **Browsers**; • **Security and privacy** → *Web application security*.

Keywords

Pluggable Semantics, web software architecture, web platform, web session management, web privacy

ACM Reference Format:
James McCauley. 2026. The Web Needs a Logout Button: Moving Beyond Document Semantics on the Web. In *The 25th ACM Workshop on Hot Topics in Networks (HotNets '26), November 16-17, 2026, Salt Lake City, UT, USA*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3847290.3859971



This work is licensed under a Creative Commons Attribution 4.0 International License.
HotNets '26, Salt Lake City, UT, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-3029-0/2026/11
https://doi.org/10.1145/3847290.3859971

1 Introduction

The World Wide Web began as a system to support hypermedia documentation [2]. It has, obviously, grown far past this initial goal: it has become a crucial and cross-cutting element of daily life for many, touching on aspects as diverse as education, entertainment, finance, personal communication, and virtually anything else imaginable. One enabler for this has been the continuous expansion of the *web platform* (HTTP, HTML, etc.) with powerful new mechanisms that extend the linked-text capabilities at its heart—mechanisms like WebSockets, WebGL, IndexedDB, and many others [12, 15, 16].

However, the web as a platform has been remarkably slow to introduce new abstractions or meanings (*semantics*) that natively support pervasive human-level concepts. HTML provides semantics for formatting and interlinking text, WebGL brought semantics for shading 3D surfaces, IndexedDB brought semantics for ranged queries on indexed data, and so on, but these are low-level, limited, mechanistic concerns. The web platform offers scant higher-level representations for the ways in which the web is *actually meaningful in our lives*. The web's dominant use cases today aren't just about linked documents, triangles, or key/value pairs; they're about shopping, social interaction, productivity, and otherwise living our digital lives. The persistent focus of the platform on low-level capabilities without higher-level, human-meaningful semantic representation or support has resulted in tremendous architectural debt as web developers endlessly reinvent these ubiquitous concepts within the vulnerable, fragmented layer of site-specific JavaScript instead.

Some might argue that the web platform's strength lies in the generality of its mechanistic semantics: they provide the building blocks from which virtually anything can be constructed without forcing site developers into specific patterns or preconceived high-level concepts. Indeed, the argument that the browser itself should be *restricted* to low-level semantics—with higher-level semantics the domain of site-level JavaScript—was articulated in the 2013 *Extensible Web Manifesto (EWM)* in response to frustrations with the slow pace and inflexibility of prior efforts to platformize higher-level semantics within browsers. The EWM was co-signed by individuals at Google, Mozilla, the W3C, and elsewhere [27], and has served as a core guiding principle for the web's evolution ever since.

For novel, evolving, and strictly presentational use cases, we fully agree that the site-level flexibility afforded by a wealth of low-level platform mechanisms is a truly invaluable keystone of the web. However, for features that are *sufficiently mature, common, pervasive, and endlessly replicated* across millions of websites—like logging in or accepting privacy policies—it creates immense architectural debt while providing no value. Just as graphics-capable computing systems eventually incorporated ubiquitous elements like menus, buttons, and scrollbars directly into platforms, it is time for high-level semantic capabilities ubiquitous on the web to be directly integrated into its platform.

As illustrative examples, this paper argues that the web should move significantly beyond simply providing mechanisms to secure a single login click or transmit a single purchase authorization (the boundary of recent platform additions). Rather, the platform should incorporate more holistic ideas about what "user sessions" and "online shopping" *mean*—and should *directly* facilitate users working within these frameworks of meaning (e.g., logging out or tracking a shipment). More broadly, we argue that the web platform must undergo a fundamental shift in development philosophy: we must strategically augment the web not just with additional powerful mechanisms, but with a deeper integration of meaning. By introducing semantics that reflect common concepts and interactions that are directly meaningful to users at the *platform level* rather than within site-specific JavaScript, the web can address its mounting architectural debt, establish stronger security baselines, and dramatically reduce developer toil.

A critical question, however, immediately arises. If not in the site nor—following the rationale of the EWM—in the browser… where should these semantics be housed?

To resolve this tension, we propose extending the web platform with an entirely new dimension for semantic expression that decouples complex, platform-level semantic logic from both the site and the browser. Indeed, we argue that this new degree of freedom promises to go beyond paying back debt by providing a more direct route for platform innovation to more stakeholders and unlocking new and cross-silo user experiences that are currently architecturally impossible but would bring significant "quality-of-life" improvements to all.

2 Architectural Debt and its Consequences

The web platform's architectural debt stems from its foundational design as a distributed document system. Its root semantics (as embedded in *HTTP* and *HTML*) were targeted expressly toward this purpose. The subsequent addition of *JavaScript* (and, later, WebAssembly) introduced generic programming semantics, giving developers the ability to implement *their own* complex semantics. When viewing the web purely as a technology stack, one might view this as simply adding a new layer or extending an existing one. However, the layered perspective—while invaluable for understanding dependencies and actually building systems—fails to capture how significantly this development altered the web. It is more illuminating to consider that in the early web, the space of expressible semantics was confined to the single axis of the platform, defined by web standards and embodied largely by the browser; the arrival of JavaScript brought an independent and orthogonal axis with semantics defined by site developers themselves.

The flexibility of this new two-dimensional *Sites*×*Browser* space of semantic expression enabled explosive growth in site complexity and the web's overall utility, but sparked debate over which axis should be responsible for which semantics and how they should intersect.[1] As noted above, the EWM formalized the answer: leave necessary mechanical semantics to the *Browser* axis and push all high-level semantics into the *Sites* axis. While we stress that this philosophy successfully resolved growing pains of the web and facilitated an age of incredible innovation, it is also the root of persistent issues that stem from the fact that JavaScript became a sort of "semantic black hole": the rich semantics implemented within this *site logic* lack meaning to the platform. Moreover, they are generally hidden from the site logic of other sites, resulting in substantially the same semantics being widely replicated: a million different sets of user authentication semantics, a million different sets of social media sharing semantics, and so on. One might argue that widespread use of JavaScript libraries and SaaS offerings (e.g., Auth0, Shopify) mitigates this by normalizing site-held semantics. However, libraries are numerous, services are built of their own site logic, and both require per-site integrations: they ultimately rely on site logic, which is fundamentally unsuited for high-stakes semantics such as those involving privacy, identity, or money for reasons that manifest in several critical ways:

Security Vulnerabilities: UIs implemented in site logic for critical actions create vectors for attacks like *clickjacking*, where users are tricked into unintended actions. Moreover, sensitive data like session IDs, passwords, and credit card details are frequently handled by client-side site logic and thus vulnerable to Cross-Site Scripting (XSS) attacks. Complex site logic based on mechanical semantics is also at the root of attacks like *client-side Cross-Site Request Forgery* which can bypass traditional anti-CSRF defenses [13], and the usage of code libraries in service of site logic opens the door to dramatic software supply-chain compromises (the polyfill.io

[1]We capture the entire platform as *Browser* in this formulation as it is the most significant embodiment of the protocols, languages, and APIs that define it. For simplicity, we do not directly consider the server side of the platform, and we omit separate but historical axes like Java Applets.

incident alone impacted over 100,000 sites [24]). The list of vulnerabilities goes on, and atop it all, site logic is essentially unauditable. Freezing dependencies to ease auditing is dangerous because it halts vital security updates; allowing updates throws the door open to supply-chain attacks. Even if a developer perfectly audits their own site, there is no possibility for meaningful *external* auditing, because a site can be entirely different from one reload to the next.

Fragmented and Inconsistent User Experience (UX): Fundamental interactions like logging in, sharing content, or accepting privacy policies are implemented uniquely by almost every website. This leads to user frustration due to inconsistent or unintuitive UIs, accessibility issues, and a fundamental lack of a secure baseline as there is little reference for "normal" behavior.

Bloat & Developer Toil: Web developers continually reinvent or re-deploy everyday functionality that should ideally be provided by the platform for the reasons discussed above. This redundant effort not only consumes vast engineering resources but also requires shipping and executing massive amounts of duplicate site logic, contributing significantly to page bloat and execution overhead.

Lost Opportunity: The need to interface with elements outside the web is one of the cases where semantics *have* been added to the web platform itself, as this cannot be done within site logic alone. In some cases, such as the introduction of WebGPU to allow access to GPU hardware (which lives outside the web), this is entirely appropriate. But in other cases, this pattern of reactive adaptation only when the platform's hand is forced represents lost opportunity. Current platform semantics involving identity and e-commerce are examples (touched on in Section 3.1 and Section 3.2), but a tragically illustrative case is social media sharing. Initially, social sharing (e.g., Facebook "likes" on third-party sites) was implemented without platform-level semantics via site logic (and consequently was famously vulnerable to, e.g., "likejacking" attacks). It was only the rise of external native mobile apps that finally forced platformization via the Web Share API [10], a thin mechanism that essentially delegates to the OS and apps. Thus, ironically, the web's first platform-level social sharing semantics—and the only ones not subject to all of the problems above—led to a better experience when sharing web content via external apps (most of which have roots on the web) than when sharing web content via the web itself.

Much of the above is simply exacerbated in the era of the Large Language Model, which simultaneously lowers the bar for creating *more* new variations of slightly broken, distinct, insecure site logic while giving threat actors dramatic new capacity for ingesting, analyzing, and finding weaknesses at scale. A new way forward must be found.

3 Platformizing High-Level Semantics

To address this architectural debt and leverage semantic understanding, we propose new platform primitives. These go beyond mechanical APIs, aiming to fundamentally augment the web platform's "understanding" of ubiquitous concepts. While myriad opportunities exist, we focus on three critical areas—identity, e-commerce, and cookie consent—that demonstrate the impact of this shift.

In the most obvious execution of this plan, these semantics would find their implementation natively in the browser. For relatively simple semantics, this is feasible. However, forcing browser vendors to natively absorb the design burden and liability of complex semantics (e.g., identity and e-commerce) risks recreating the exact problems the EWM sought to avoid, not least because doing so is often likely to be against browser vendors' rational self-interest. When new platform semantics do not provide commercial advantage, they are simply a sink for developer time and a potential source of liability (if not legal, then at least to brand perception).

To sidestep this deadlock, we propose extending the *Sites*× *Browser* space of semantic expression with an entirely new dimension: *Sites* × *Browser* × *Pluggables*. We envision *Pluggable Semantics* as first-class, standardized platform primitives that define specific semantic capabilities. Crucially, rather than requiring browser vendors to implement complex domain logic directly within browsers, pluggables could be constructed from sandboxed bundles of standard web technologies (e.g., HTML, JavaScript, WebAssembly). While a standards body would define the strict interfaces and semantic boundaries, the implementation logic would be decoupled from the browser itself and executed in a context distinct from the site logic invoking the semantic functionality. In greater detail, the architecture has the following properties:

First-Class Universality & The Ecosystem: As part of the platform, every pluggable standard should include a reference implementation, ensuring universal baseline availability. Where it is in their interests, browser vendors will likely forgo the reference and ship their own default implementations (e.g., Safari bundling an iCloud-backed identity pluggable), but should also support substitution, allowing users to swap defaults for third-party alternatives (e.g., an enterprise-managed identity pluggable through Okta or a privacy-preserving implementation backed by the EFF).

Versioned Auditability: By shifting complex functionality out of dynamically loaded site scripts and into self-contained, versioned, digitally signed bundles, pluggables become highly and externally auditable. Unlike the chaotic dependency trees of modern site logic, which pull unverified code from dozens of third-party domains on every page load, pluggables neutralize a major vector of software supply-chain debt.

Defense-in-Depth Sandboxing: Pluggables operate behind strict barriers, completely isolated from potentially hostile site logic, interacting exclusively through narrow, typed intent APIs, cleanly severing vectors for lateral movement. Isolation extends to state and interactions with the network, rendering them structurally invisible to site logic and preventing leakage or tampering. For highly sensitive semantics, a pluggable's manifest could specify that performance should be traded for extreme hardware isolation unsuitable for site logic, resulting in pinning to an isolated physical core, aggressive memory fencing, or constant-time padding to mitigate side-channel vulnerabilities like Spectre [14].

Trusted UI: Pluggables can be granted the ability to render UI elements entirely outside the document and within browser-managed trusted chrome. Because site logic cannot overlay or capture events within this space, entire classes of attacks, such as DOM-based clickjacking, are eliminated.

Taken together, these properties demonstrate why pluggables constitute a truly orthogonal third axis. The axes are fundamentally decoupled across development, deployment, governance, and trust: the *Browser* axis is governed by vendor consensus and deployed through traditional software release cycles; the *Sites* axis is controlled by individual developers and deployment is dynamic and essentially unauditable; the *Pluggables* axis introduces independent third-party authorship and stable, auditable deployment at an independent cadence. This orthogonality also extends to user agency: choice of browser, choice of sites to visit, and choice of pluggable are uncoupled degrees of user freedom.

Finally, it is worth noting that pluggables do share some technical similarity with browser extensions and modern WebExtensions [21], but differ fundamentally here as well. First, extensions are optional, ad hoc bolt-ons; pluggables are universal platform primitives sites can safely rely upon. Second, extensions operate via unstructured (and often unsafe) DOM and JavaScript environment interaction; pluggables are strictly insulated. Third and most significant, the extension ecosystem—like site logic—is a bit of a "Wild West" in that virtually anyone can write and distribute code with users often having relatively little insight into authorship or behavior due to both the ecosystem design and the sheer number of extensions. In contrast, the fixed number of pluggable standards and the high stakes of the semantics involved mean we respectively *can* and *must* construct their ecosystem entirely around rigorous trust and accountability so that users can make informed choices—a paradigm that has never been tractable for the open-ended extensions community.

3.1 Semantics for Identity and Sessions

The architectural debt associated with identity and *session management* is arguably the most pervasive on the modern web, and the web's historical journey highlights its accumulation. HTTP's *Basic Access Authentication (Basic Auth)* was an early platform-level semantic capability for, in effect, authenticated sessions. By having the browser manage credentials, the platform natively understood that a user was "logged in." However, Basic Auth was indeed frustratingly basic, lacking features like branded login flows and—infamously—any platform mechanism corresponding to actually *logging out*. Rather than refining this platformized authentication, it was abandoned almost entirely. The subsequent introduction of the browser cookie provided a powerful, generic mechanism for maintaining state; developers used this to build a wide variety of session semantics in site logic, subject to all of the downsides discussed in Section 2.

Today, platform APIs like the *Federated Credential Management API* (FedCM) [18] and *WebAuthn* [1] make tremendous strides in securing the *transaction* of initial authentication (e.g., replacing hugely insecure password handling), but stumble at the boundary of ongoing identity. The core platform remains fundamentally blind to what a "session" actually is. Even if WebAuthn secures the initial login, the ongoing session still relies on opaque cookies managed by site logic, vulnerable to attacks like post-authentication session hijacking (a spiking attack vector; one recent investigation found over 18 billion *active* stolen cookies on the dark web [17]).

To address this, we propose the addition of a session management pluggable with the following implications:

Consistent Browser-Managed Identity UI: The browser would present a consistent, non-modal UI (e.g., on the toolbar) that indicates current session state and serves as the entry point for starting sessions (seamlessly leveraging FedCM and WebAuthn as appropriate) and stopping them (the logout button the web has always lacked). Sites provide semantic hints for minimal branding rather than building custom forms, reducing developer burden and ensuring users authenticate through a known, un-spoofable interface.

Multi-Account Identity Management: Dedicated UI coupled with semantic understanding of identity separate from cookies allows for straightforward *per-tab identity management*. Switching identity in one tab (e.g., from *work* to *personal* via an OS-like "fast user-switching" menu) leaves other tabs logged in under their original identity, enabling simultaneous multi-account use without clunky browser profiles.

Enhanced Security by Design: Because the authentication UI is rendered outside the document, it is inherently resistant to clickjacking. Furthermore, by moving ongoing session state into the pluggable, session tokens are sequestered from site logic, immune to session hijacking. Finally, this pluggable provides a secure integration point for password managers, structurally superior to traditional browser extensions whose reliance on unsafe DOM interactions recently left tens of

millions of user accounts vulnerable to credential leaks across virtually all password managers [26].

Enhanced Digital Footprint Management: Currently, desirable features like viewing active sessions across devices or forcing remote sessions to log out exist within the site logic of a minority of well-resourced sites. Because a platform-level pluggable is aware of login status across all participating sites, a third-party implementation could provide a unified "identity dashboard" where users can easily audit and manage their digital footprint globally.

3.2 Semantics for E-Commerce

Much like WebAuthn in the identity space, recent mechanistic semantics in the web platform such as the Payment Request API (PRA) [6] and the Secure Payment Confirmation (SPC) API [25] provide significant security wins, but only for momentary interactions: they authenticate the payment click and sequester raw credentials, yet leave the broader e-commerce lifecycle—encompassing ubiquitous concepts like shopping carts, recurring subscriptions, and delivery addresses—abandoned to site logic. Consequently, the total e-commerce experience remains highly fragmented and vulnerable; for example, clickjacking can still deceive users during cart management, and XSS and Magecart-style supply-chain attacks can still silently manipulate cart contents or redirect delivery addresses. We propose resolving this via a composable suite of e-commerce semantics fulfilled by pluggables:

Shopping Cart & Fulfillment: By pulling cart construction and shipping details entirely out of the DOM, the checkout intent becomes tamper-evident: what the user sees in the secure cart UI is guaranteed to be what ends up in the order, and fulfillment information (like home addresses) never enters vulnerable site logic, preventing e-skimming attacks. Even the basic reference implementations of these semantics would allow users to easily view cart contents or shipping status across all sites, eliminating the burden of chasing information down on individual sites or through emails. A third-party provider could augment this with features like cross-site comparison shopping that works at the semantic level rather than relying on brittle, difficult-to-secure browser extensions that scrape the DOM.

Subscription and Lifecycle Management: Standardized semantics for billing cycles enable a separate lifecycle pluggable to provide a unified user dashboard of subscriptions and other recurring charges across all sites, akin to app-store subscription management. Hooks for cancellation would allow users to easily cancel subscriptions, and a future version of regulations like the FTC's "click to cancel" effort [9] could simply mandate that sites implement those hooks correctly.

Payment and User Agency: The default implementation of a payment pluggable would likely be a thin wrapper around mechanical APIs like PRA and SPC. However, this modularity becomes an axis for user agency: the default could be replaced by a bank-provided pluggable to facilitate context-aware fraud prevention, or an employer-provided pluggable that enforces corporate purchasing policies and automates the filing of receipts.

While the proposed suite of e-commerce semantics would not address *every* pain point of e-commerce (e.g., server-side vulnerabilities), it could drastically raise the bar in terms of both security and user experience for this incredibly common web use case.

3.3 Semantics for Cookie Consent

The rise of privacy regulations like GDPR and CCPA [5, 7] has necessitated explicit user consent for data tracking, most visibly manifested as a need for *cookie consent*. However, the web platform lacks a standardized semantic primitive for managing this consent, essentially resulting in *legally mandated* replication of a ubiquitous semantic entirely within site logic. This has led to a proliferation of ad hoc, site-specific, and—frankly—annoying cookie consent dialogs. These dialogs disrupt browsing, induce *consent fatigue*, frequently employ UX *dark patterns* to manipulate users, inflate page load times with JavaScript and CSS, and transform regulatory compliance into an ongoing source of engineering toil.

To confront this, we propose augmenting the `Set-Cookie` header with a standardized `Purpose` field with values modeled after common cookie purposes seen in today's consent dialogs (e.g., `necessary`, `analytics`, `advertising`). This would empower the browser to enforce user-configured preferences, allowing it to present a consistent, non-intrusive UI or automate the choices entirely—for instance, by automatically allowing only `necessary` cookies when browsing in private/incognito mode, or providing a straightforward UI to change choices later (often difficult today). Web developers win through reduced compliance effort, as best practices become structurally built into the platform.

The semantics of cookie preferences are simple enough that we believe a native browser implementation would be practical and sufficient. That this has not been done already therefore serves as an example of how one barrier to a better web is not the lack of pluggables, but simply the bias towards only platformizing semantics when absolutely necessary.

4 Beyond Just Debt: New User Experiences

While we lack space to fully elaborate our vision, platformizing user-meaningful semantics goes beyond architectural debt reduction: it enables browsers that act as intelligent, context-aware agents which can orchestrate *across* isolated site boundaries.

For example, a *contacts* pluggable could provide address-book-type functionality across services like Gmail, Slack, and Instagram. Even a standalone reference implementation would allow users to go from an email address link to a webmail "To:" field without the present copy-and-paste tax. Moreover, a pluggable could serve as a bi-directional bridge between the web and mobile phone or cloud address books across any browser or OS. While past efforts like the W3C Contact Picker API [4] stalled under the burden placed on browser vendors to maintain integrations with native OS address books and mitigate privacy risks (e.g., user deception or accidental over-sharing), pluggables offload this burden to incentivized parties while simultaneously offering them new capabilities (e.g., in-browser UX to curb clickjacking).

Contact semantics could move beyond simple address book integration, instead creating a holistic understanding of people and relationships across the web that could provide context tags for history and bookmarks, all within the browser and its pluggables. Other contextual tags could include how a user arrived at a URL (e.g., an emailed link from a specific coworker, a search result for "best spinach dip recipe"), or item category (e.g., technical documentation, portrait photo). Such semantically tagged history and bookmarks would enable powerful tools: visualizing information-seeking journeys, automatically sorting bookmarks across multiple dimensions, or permitting complex history queries from the address bar like "Creative Commons photos shared by coworkers"—a query currently impossible because the concepts of "coworkers" and "photo licenses" live in disconnected site-logic silos. Such context could also inform browser policies, e.g., that "healthcare" sites should use the highest degree of site isolation possible.

5 Incentives for Adoption

While our proposals are ambitious, the structural incentives of developers, browsers, and users give us hope for adoption.

For web developers and service providers, platformization provides a number of compelling "carrots" like reduced development burden, reduced page load times, improved security, and potentially simplified regulatory compliance.

Additionally, pluggables hold promise as a more incentive-aligned way of addressing the root issue that motivated the EWM. In past efforts to platformize complex semantics, the implementation was shouldered solely by browser vendors with their own strategic and (zero-sum) engineering priorities. In contrast, pluggables can be implemented by highly motivated third parties; an organization like the EFF or a commercial entity like Shopify could implement or update a pluggable entirely independently of browser interests and release cycles. Such third parties would have far more latitude for innovation than today, as their primary constraint in a pluggable future is just that they conform to a standardized pluggable interface. Naturally, these same parties (along with other domain experts) would also likely drive the standardization process for those interfaces.

Notably, browser vendors can almost entirely opt out of both implementation and standardization, only needing to hook the interfaces into the isolated pluggable runtime environment which uses technology largely similar to what they already utilize for per-site isolation. Should browser vendors find it advantageous, however, they could always prioritize their own pluggables, as they control the defaults they ship.

Users may also drive adoption by eschewing legacy sites after experiencing the improvements that sites with platformized semantics offer (e.g., unified UX across sites inside trusted chrome, better performance due to pluggables executing out of local bundles). For some users, the increased and finer-grained agency of pluggable selection (e.g., running a privacy-first session pluggable in a mainstream browser, smoother integration with their preferred payment ecosystem) may also be a strong motivator.

Finally, as touched on in Sections 3.2 and 3.3, platformization of high-level semantics may offer a path to *better realization of legal aims* like consumer protection than existing efforts, which attempt to enforce high-level goals by regulating low-level mechanisms or by litigation after the fact.

6 Related Work

Previously mentioned browser-level APIs [1, 6, 18] have introduced specific mechanistic semantics for establishing identity or facilitating payment. While invaluable for specific interactions, their focus (like many other web APIs) has primarily been on reactively exposing external services or underlying system capabilities (e.g., biometric authenticators) rather than embedding comprehensive, user-centric semantic understandings into the platform as we advocate.

OAuth DPoP [8] and TLS Token Binding [19, 20] share one aim of our proposed web session semantics: reducing hijacking. However, the former operates *atop* the web platform (in site logic) and the latter (mostly) *below* it.

The original *Semantic Web* vision and *Schema.org* aim to create a "web of data" [3, 11]. While sharing our goal of infusing the web with meaning, their primary emphasis has been on semantic tagging of content for machine consumption and inference. In contrast, we focus on folding user-meaningful interactions into the platform to directly improve the web: reducing site logic, enhancing security, and improving UX. These goals are complementary, and the rich Semantic Web vocabularies have already laid groundwork for the metadata required by the type of features touched on in Section 4; e.g., the Schema.org *Person* is an ideal basis for behaviors associated with our proposed *contact* semantics.

7 Discussion & Conclusion

The web's evolution has been fueled by the integration of remarkable mechanisms that enable increasingly complex sites. However, the ubiquitous concepts of today's web are not just text styling, video streams, or WebSocket messages. They are not even just things like credentials and payment transactions that commendable efforts have worked to platformize. They are complex ideas like user sessions, shopping carts, purchase histories, tracking preferences, colleagues, and much more. The web has been slow to reify such concepts into platform-level primitives, leaving developers to repeatedly reinvent them, typically hidden within the semantic black hole of site-specific JavaScript. This has created a brittle and ever-shifting landscape of vulnerabilities and user frustration. We argue that the way forward is to stop seeing the web as a collection of mechanisms and instead view it as the platform for humans to live the online part of their lives—and to embed semantics within it accordingly.

While much of our analysis in this paper employs an axial view of the web that focuses attention on how aspects of the web can be acted on independently, the actual implementation still exists as a layered technology stack featuring a platform layer and a sites layer. And while much of this paper is framed as a response to architectural debt, more fundamentally, it addresses an essential architectural question: at which layer should functionality be placed?

Huge portions of code that characterize the modern web are presently placed in what amounts to the highest software layer, immediately beneath the human layer. This *site logic* layer, typified by JavaScript, implements most of the human-meaningful semantics of the web atop the generally thin, mechanical semantics of the underlying platform layer. One might argue that high-layer placement of high-level semantics is principled, invoking variations of Saltzer, Reed, and Clark's *end-to-end argument* [22] or Saltzer and Schroeder's principle of *least common mechanism* [23]. These suggest that functions are best placed at higher layers if they can be "completely and correctly" implemented there, or if sharing a lower-layer implementation could compromise security.

We contend that for many pervasive web functionalities, practical evidence in the form of decades of security issues demonstrates they *cannot* be correctly implemented within site logic. Moreover, the end-to-end argument, as the paper itself calls out, is most aptly applied as an Occam's razor against unnecessary lower-layer complexity in support of *hypothetical future applications*, not against the platformization of already ubiquitous (and problematic) functionalities. Furthermore, platforms enable holistic and synergistic effects that *cannot* be equivalently implemented at higher layers. Only the platform, for instance, could provide a tamper-proof and unforgeable UX "outside the document" for sensitive interactions, and only by *lowering* the layer of a piece of functionality can one simultaneously *elevate* that functionality to a first-class concept which can then be freely and reliably reused, orchestrated, and synthesized by layers above.

While platformizing functionality might seem to run counter to the spirit of *least common mechanism*, it is in keeping with other major principles articulated by the same authors. By moving sensitive functionalities into the platform, critical information can be cloistered away from attacks like XSS, consistent with the *principle of least privilege* as applied to JavaScript. A well-known, universal user experience for sensitive operations adheres to the *principle of psychological acceptability*. Perhaps most importantly, platformizing ubiquitous functionality is consistent with the *principle of economy of mechanism*. Rather than infinite libraries, frameworks, and one-off implementations of widespread functions, we should prefer a smaller number of implementations from a handful of trusted parties following a single community specification. Thus, while *the platform's* surface area might grow slightly, there is still a dramatic net reduction in *total* surface area. This numerical and spiritual alignment with economy of mechanism is vital: as Saltzer and Schroeder highlight, it enables code auditing. Even if it were not structurally impossible, there are simply not enough well-motivated auditing eyeballs to confront the site-logic JavaScript ecosystem. Pluggables, conversely, offer delineated and versioned codebases that can be rigorously evaluated.

While we propose embracing a fundamentally new architectural dimension and rich semantics at the platform layer, the underlying technical complexity is modest compared to the vast engineering investments already made in core technologies like performant JavaScript engines, QUIC, and robust sandboxes. The most fundamental challenge is not one of technical infeasibility, but of structural imagination. The introduction of new mechanisms to expand the web's capabilities is vital, but equally important is the platform's responsibility to house ubiquitous human meanings. By introducing a new axis of semantic expression through *Pluggable Semantics*, interested parties—like the broad networking community—gain a new dimension for extending the architecture of the web and, ultimately, better serving its users. And while it's not as flashy as the WebAssembly Component Model, WebTransport, or WebGPU Shading Language, we really do need that logout button.

Acknowledgments

The author would like to thank Hamed Haddadi for shepherding the paper, the anonymous reviewers for their constructive comments, and Scott Shenker for serving as a thoughtful sounding board during the development of these ideas. All errors remain solely the author's own.

References

[1] Dirk Balfanz, Alexei Czeskis, Jeff Hodges, J.C. Jones, Michael B. Jones, Akshay Kumar, Angelo Liao, Rolf Lindemann, Emil Lundberg, Vijay Bharadwaj, Arnar Birgisson, Hubert Le Van Gong, Christiaan Brand, Adam Langley, Giridhar Mandyam, Mike West, and Jeffrey Yasskin. 2019. *Web Authentication: An API for accessing Public Key Credentials - Level 1.* W3C Recommendation. https://www.w3.org/TR/2019/REC-webauthn-1-20190304/

[2] Tim Berners-Lee. 1990. *Information Management: A Proposal.* Technical Report. CERN. https://cds.cern.ch/record/369245

[3] Tim Berners-Lee. 1998. *Semantic Web Road map.* https://www.w3.org/DesignIssues/Semantic.html

[4] Peter Beverloo and Rayan Kanso. 2023. *Contact Picker API.* W3C Working Draft. https://www.w3.org/TR/2023/WD-contact-picker-1-20230120/

[5] California Privacy Protection Agency. 2025. California Consumer Privacy Act of 2018, Cal. Civ. Code § 1798.100 et seq. https://cppa.ca.gov/regulations/pdf/ccpa_statute.pdf.

[6] Marcos Cáceres, Rouslan Solomakhin, Ian Jacobs, Domenic Denicola, Adrian Bateman, Zach Koch, Roy McElmurry, and Danyao Wang. 2022. *Payment Request API.* W3C Recommendation. https://www.w3.org/TR/2022/REC-payment-request-20220908/

[7] European Parliament and Council of the European Union. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council. https://data.europa.eu/eli/reg/2016/679/oj.

[8] Daniel Fett, Brian Campbell, John Bradley, Torsten Lodderstedt, Michael B. Jones, and David Waite. 2023. OAuth 2.0 Demonstrating Proof of Possession (DPoP). RFC 9449. https://www.rfc-editor.org/info/rfc9449/

[9] FTC Office of Public Affairs. 2024. Federal Trade Commission Announces Final "Click-to-Cancel" Rule Making It Easier for Consumers to End Recurring Subscriptions and Memberships. Press Release. https://www.ftc.gov/news-events/news/press-releases/2024/10/federal-trade-commission-announces-final-click-cancel-rule-making-it-easier-consumers-end-recurring

[10] Matt Giuca, Eric Willigers, and Marcos Cáceres. 2023. *Web Share API.* W3C Recommendation. https://www.w3.org/TR/2023/REC-web-share-20230530/

[11] R. V. Guha, Dan Brickley, and Steve Macbeth. 2016. Schema.org: Evolution of Structured Data on the Web. *Commun. ACM* (Jan. 2016). https://doi.org/10.1145/2844544

[12] Ian Hickson. 2009. *The Web Sockets API.* W3C Working Draft. https://www.w3.org/TR/2009/WD-websockets-20090423/

[13] Soheil Khodayari, Thomas Barber, and Giancarlo Pellegrino. 2024. The Great Request Robbery: An Empirical Study of Client-side Request Hijacking Vulnerabilities on the Web. In *2024 IEEE Symposium on Security and Privacy.* https://doi.org/10.1109/SP54263.2024.00098

[14] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2020. Spectre Attacks: Exploiting Speculative Execution. *Commun. ACM* (June 2020). https://doi.org/10.1145/3399742

[15] Chris Marrin. 2011. *WebGL Specification 1.0.* Khronos Group Specification. https://registry.khronos.org/webgl/specs/1.0.0/

[16] Nikunj Mehta, Jonas Sicking, Eliot Graff, Andrei Popescu, Jeremy Orlow, and Joshua Bell. 2015. *Indexed Database API.* W3C Recommendation. https://www.w3.org/TR/2015/REC-IndexedDB-20150108/

[17] Nord Security Research. 2025. From 54 billion to 94 billion: Cookie theft skyrockets as hackers exploit your browser. https://nordsecurity.com/press-area/from-54-billion-to-94-billion-cookie-theft-skyrockets-as-hackers-exploit-your-browser.

[18] Nicolás Peña Moreno and Sam Goto. 2024. *Federated Credential Management API.* W3C First Public Working Draft. https://www.w3.org/TR/2024/WD-fedcm-1-20240820/

[19] Andrei Popov, Magnus Nyström, Dirk Balfanz, Nick Harper, and Jeff Hodges. 2018. Token Binding over HTTP. RFC 8473. https://www.rfc-editor.org/info/rfc8473/

[20] Andrei Popov, Magnus Nyström, Dirk Balfanz, and Jeff Hodges. 2018. The Token Binding Protocol Version 1.0. RFC 8471. https://www.rfc-editor.org/info/rfc8471/

[21] Mukul Purohit, Tomislav Jovanovic, Oliver Dunk, and Kiara Rose. 2026. *Web Extensions.* Draft Community Group Report. https://w3c.github.io/webextensions/specification/

[22] Jerome H. Saltzer, David P. Reed, and David D. Clark. 1984. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems* (Nov. 1984). https://doi.org/10.1145/357401.357402

[23] Jerome H. Saltzer and Michael D. Schroeder. 1975. The Protection of Information in Computer Systems. *Proc. IEEE* (Sept. 1975). https://doi.org/10.1109/PROC.1975.9939

[24] Sansec Forensics Team. 2024. Polyfill supply chain attack hits 100K+ sites. https://sansec.io/research/polyfill-supply-chain-attack.

[25] Rouslan Solomakhin, Stephen McGruer, Ian Jacobs, and Nick Burris. 2023. *Secure Payment Confirmation.* W3C Candidate Recommendation Snapshot. https://www.w3.org/TR/2023/CR-secure-payment-confirmation-20230615/

[26] Marek Tóth. 2025. DOM-based Extension Clickjacking: Your Password Manager Data at Risk. https://marektoth.com/blog/dom-based-extension-clickjacking/. Presented at DEF CON 33.

[27] W3C Extensible Web Community Group. 2013. Extensible Web Manifesto. https://github.com/extensibleweb/manifesto.