SoK: All or Nothing - A Postmortem of Solutions to the Third-Party Script Inclusion Permission Model and a Path Forward

Steven Sprecher
Northeastern University
swsprec@ccs.neu.edu

Christoph Kerschbaumer
Mozilla Corporation
ckerschb@mozilla.com

Engin Kirda
Northeastern University
ek@ccs.neu.edu

***Abstract*—The web execution model allows third-party JavaScript to be leveraged in a single execution context. Access control for these scripts is currently all or nothing. It has been this way for over a decade despite the knowledge that this model allows for privacy violations and even user data exfiltration. Consequently, users have little to no control over which third-parties operate on their Personally Identifying Information (PII) when interacting with a web application.**

In this work we aim to explain the lack of solutions to this problem, and to suggest more promising future directions. We first survey past proposed solutions and their trade-offs. We then create a monitoring system in the Firefox browser which captures third-party script access to user supplied PII in HTML Form Elements. We proceed to inspect 100,000 websites with our Monitor and custom web crawler to highlight the complexity of use cases of third-party scripts operating on user PII.

Our findings inform the creation of a grading rubric and systematization for solutions in this space, which we then apply to many previous works. The complexity exposed through this effort allows us to start a discussion around why current technological and policy solutions fail adoption. Ultimately we propose a research direction that allows web applications to take advantage of the interoperability of the web execution model while also respecting an end user's privacy and security.

***Index Terms*—Web security, Privacy, Browsers, Third-Party Scripts, JavaScript**

1. Introduction

JavaScript has evolved to become the most pervasive and dominant programming language of the web, and is omnipresent on websites today. Among the reasons for its proliferation is its ability to pull and integrate together JavaScript code from multiple origins on the Internet. While this execution scheme is extremely powerful, pulling code from different origins into the same execution context also grants scripts full access to application internals, and in turn poses genuine threats to the security and privacy of end users [1]–[6]. The technical ability of third-party scripts to access or even exfiltrate PII such as email addresses, usernames, and passwords is particularly worrisome.

Many successful web applications aim to take advantage of the interoperability of the modern web. At the same time, they worry about the aforementioned security and privacy implications. Thus, they do not wish to grant a script access to application internals, and rather choose to execute the script in a different security context. Isolating, and ultimately sandboxing scripts by loading them into an `<iframe>` element is effective from a security point of view because it causes the browser to create a new global object, and therefore completely separates the execution context of the third-party script from the main application. The downside, however, is that the pulled in third-party script has limited options to interact with the main application.

In summary, the current web security model permits a third-party script either (a) full access to the main application, or (b) no access by generating a separate execution environment – effectively separating the two. Naturally, the majority of use cases on the web require a mix of the two. Over the course of a decade, the web community has proposed and introduced a variety of approaches to address this "all or nothing" trust paradigm. Classes of solutions include applying fine-grained application-specific controls, incorporating access control ideas into web elements, and tracking information flow to allow decisions on the script behavior. Yet, neither browser vendors nor web application developers seemingly have adopted any of these solutions. The consensus in industry is that the trade-offs between overhead, developer and user requirements, security flaws, backwards compatibility, and site breakage seem almost impossible to navigate successfully.

In this work, we conduct a deep dive into the techniques and trade-offs of previous academic and industry solutions. We find a complicated network of costs and benefits that has prevented the mainstream adoption of any solutions. This prompted us to investigate the problem in the wild to gain a deeper understanding of the lack of real world implementation. To do this we develop an in-browser *Script Access Monitor* that allows us to investigate all Document Object Model (DOM) [7] element access, and a custom Selenium [8]-based inspector to demonstrate and visualise how complicated the security and privacy of third-party scripts are in their uses. We inspect and measure 100,000 websites, and highlight the puzzling situation of third-party script inclusion and its security and privacy implications resulting from the many middle ground usages of this elevated privilege. Lever-

aging our insights from our investigations, we develop a grading rubric for techniques in this space and systematize them by applying our rubric. Completing our postmortem, we distill our findings and recommend a viable and promising path forward.

In summary, this work contributes the following:

- We conduct a deep dive into current and past approaches (Section 2) which aim to close the gap between full access and no access of third-party script inclusion within web applications.
- We implement a *Script Access Monitor* in Firefox (v.88 - v.92) which allows deep inspection of any interaction between JavaScript and the DOM, and illustrate the problematic trust situation by monitoring third-party scripts accessing PII in HTML Form Elements on 100,000 websites (Section 3).
- We present a systematized grading rubric for technical solutions to this problem and apply it to past work (Section 4).
- We provide a discussion (Section 5) around why current technological and policy solutions fail adoption, and propose a research direction that allows web applications to take advantage of the interoperability of the web execution model while also respecting an end user's privacy and security.

Availability. All code and data is publicly available on the authors' websites, or directly accessible here: https://gitlab.com/swsprec/sok-allornothing.

2. Background and Related Work

This section begins by detailing our survey methodology (Section 2.1) for the selection of related work. We then take a step back to present measurement work (Section 2.2) summarizing research projects in the field of third-party JavaScript inclusion, and its effects on an end user's security and privacy. We then present relevant background information on the current JavaScript Execution and Security Model enforced within web browsers (Section 2.3). Next, we survey approaches of the most promising techniques to introduce an additional level between full and no trust, and evaluate trade-offs (Section 2.4 and Section 2.5). To provide a wide spectrum of related work, we finally review miscellaneous solutions (Section 2.6) which sometimes only partially address the problem, but still provide relevant insights and details when trying to tackle third-party script containment.

2.1. Survey Methodology

In order to be transparent and repeatable about our research selection process, we provide a detailed survey methodology below. In order to provide a thorough background, we additionally cite supporting work that does not meet all our selection criteria to discuss. All solution techniques found in Sections 2.4 and 2.5 and any that factor into our systematization and rubric were found using the following methodology.

1) **Paper collection methodology** In order to capture as much relevant work as possible, we compiled our paper list in the following way:
   - First, using our domain expertise we collect the most relevant and impactful work into a *Seed List*.
   - Second, we include any vaguely relevant work (VRW) that is either cited by or cites any of the papers in our *Seed List* and combine into a new list, *List 2*. We define VRW as work that includes some form of discussion of third party scripts, permission models or access control on the web, information flow, confinement or isolation, JavaScript control or the like in the title and/or abstract.
   - Third, we search through the top security conferences, Google Scholar, and ACM Digital Library for the key terms that define VRW, and combine these with our last step to form *List 3*.
   - Fourth, we do a breadth-first-search of all papers in *List 3* for VRW as detailed in our second step to refine our list into *List 4*.
   - Finally, we repeat the previous step with all the new work that was added until we are left with *List 5* containing over 100 research papers.
2) **Exclusion Criteria** To further refine our collected works into a distilled list with directed relevance to our studied problem, we set the following exclusion criteria:
   - The work is not applicable to access control, permission models, or is too tangential to be useful.
   - The work is an exploration and does not include a defense attempt.
3) **Inclusion Criteria** To ensure only the most relevant research is included in our analysis, we set the following criteria for a paper to be included:
   - The research must be directly applicable with minimal changes to JavaScript 3rd party permissions and use therein.
4) **Application of Criteria** We apply our stated criteria to our collected paper list, *List 5*, in four rounds as detailed below:
   - *Round 1*: We read the title and abstract of each paper, and apply the exclusion criteria to remove papers from the list.
   - *Round 2*: We read the title, abstract, introduction and conclusion of all papers that survived *Round 1* and apply the exclusion criteria again.
   - *Round 3*: We do a full read of papers from the previous round and reapply the exclusion criteria.
   - *Round 4*: We perform a full re-read of each paper from *Round 3* and apply the inclusion criteria.

2.2. Studies on Third-Party Scripts

The JavaScript language in and of itself can be dangerous [9]–[11]. Unfortunately, JavaScript is not the only danger on the web for users, and prior work highlights this history and its development [12]–[14], as well as measuring the dangers of browser access control incoherences [15]. To make matters worse, it has been shown both historically and recently that the web is a tangled mess of included scripts. This opens issues in outdated

dependencies, and creates a web of implicit trust and a complicated ecosystem of inclusions [1], [16]–[20]. These scripts pose additional danger in the leakage of PII, and work has quantified this by looking at information flows, security issues on webpages where PII is present, and even the diffusion of PII across the web [21]–[24]. Worse yet, web tracking and data exfiltration by third-parties has been shown to be pervasive and a critical privacy issue [2], [25]. All of the above highlights the problematic situation of third-party script inclusion and calls for remedy.

2.3. The Execution and Security Context

The browser JavaScript execution context consists of three parts: the Document Object Model (DOM), the Browser Object Model (BOM), and the JavaScript core. JavaScript that is run in a first-party context, i.e. included in the main page, is considered from the same origin and executes in a global context, usually with unfettered access to the DOM and BOM.

In order to prevent scripts from one origin accessing another origin, the Same-Origin Policy (SOP) restricts communication to and access of externally loaded resources. The most common example of this is loading a different origin in an `<iframe>` [26]. This element is ideally locked down by applying additionally-provided `sandbox` attributes that causes the browser to generate a new global object, and therefore, completely separate it from the context of the main application. Scripts cannot access context between the barrier since they are not of the same origin.

SOP cannot prevent JavaScript from accessing or exfiltrating information on a page when developers purposefully pull and combine code from multiple origins in the same execution context. Including a third-party script in the context of the main application causes the script to execute using the same global object, and thus grants the third-party script full access to the DOM [7], and in turn access to potentially confidential user information.

In order to add more flexibility to SOP and to discourage the aforementioned combining of code, Cross-Origin Resource Sharing (CORS) [27] was developed. CORS uses HTTP headers to define trusted origins and other properties that scripts can leverage to bypass some of the strictness of SOP.

Scripts that are included in the main page of a document are considered as the same origin under SOP, thus third party scripts that are loaded often get unfettered access to the DOM and BOM. In order to mitigate this risk, primarily to prevent Cross-Site Scripting (XSS), Content Security Policy (CSP) [28] was added to the security model. CSP allows server administrators the ability to specify an allowlist of domains of which the browser will consider scripts as trusted. When a website owner specifies this list in the CSP header, browsers will not execute scripts loaded from any origin outside of this list. This protects against unintended inlined script inclusions.

2.4. Script Isolation and Confinement

As discussed before, the World Wide Web Consortium (W3C) introduced CSP, which allows web applications to define an allowlist of trusted scripts. For example, a CSP of "`script-src example.com`" allows scripts from `example.com` to load, but would prevent scripts from `example.com` to load further scripts from a different origin. Relying on a CSP or not, the fundamental security problem remains: the third-party script has full access to application internals, and hence, access to confidential user information [29], [30].

Techniques: The ultimate goal with isolation is to give foreign JavaScript code its own context, but at the same time, provide trusted channels to allow the third-party script to interact with the main application. Hence, most approaches build upon the `window.postmessage` API [31], and either aim to improve the communication between frames [32]–[34] or try to isolate foreign code [35]. Put differently, most of the work surrounding isolation and confinement can be boiled down to frame-based confinement and language-based confinement. Other work has changed where the isolation techniques are housed in the browser stack, moving it to the JavaScript interpreter [36], to web workers [37], and even trying to integrate control mechanisms in the client-side code to avoid browser modification [38]. Confinement has also moved towards more fine-grained controls by using data-confined sandboxes [39], or object-level confinement [40]. Industry has attmepted to isolate untrusted scripts by statically rewriting code. However, all of these confinement projects have been discontinued [41]–[44].

For an in-depth look at isolation and confinement methods, we refer the reader to Van Acker and Sabelfeld's survey [45].

Trade-offs: Isolation and confinement errs on the side of coarse-grained control. It forces code that is isolated to not have access to data, and thus is not well-suited for scripts that offer a service based on data in a web application. The aforementioned techniques harden this landscape and help add communication between isolated components, but it is clear that it cannot successfully work with scripts operating on data in the main context. Building intercommunication pathways for different scripts provides relief for privacy and security concerns since they can remain isolated. However this technique lacks usability, and hence, has not found widespread adoption in industry.

2.5. Information Flow Control

Information Flow Control (IFC) is a technique that allows tracking the flow of information throughout a system.

This technique has been around in one form or another in secure systems for quite some time. We see a form of IFC in standard access control with a permission structure and control over read and write operations. Prior to the seminal work of Denning et al. in 1977 [46], IFC was done at run time. Denning et al. pioneered the ability to do IFC at program compile time, which reduces the load on running programs, and allows for established security before execution. The next iteration of IFC took hold in language specific mechanisms for Java, and Objective Caml [47], [48].

Most relevantly, tracking the flow of information through the JavaScript engine and further throughout the entire browsing context generally allows researchers and developers to address the privacy and security concerns highlighted within this work. Some even argue that tracking the flow of information is better suited to overcome the problematic situation of granting a script full or no access to application internals than SOP [49]. Obviously, there is overlap between isolation and confinement and IFC [45]. A way to enact IFC is to confine code, and define rules to allow data to be passed between security contexts.

Techniques: There are a multitude of different technical approaches and flavors of IFC in the browser context, all with their own benefits and costs. Here we see the evolution of the space in addition to these trade-offs.

One of the initial uses of IFC on the web was to secure users against XSS attacks [4] using dynamic data tainting to follow sensitive information though execution.

SOP was argued to be too coarse-grained to allow developers to have both security and flexibility in mashups, where communication between components was not allowed by Crites et al. [50]. The authors proposed looking at pages like objects and applying IFC access control policies for fine-grained control of communications between frames.

The application of these techniques to allow scripts to operate on confidential information, yet disallowing the leaking of this data, was proposed in BFlow by Yip et al. [51]. BFlow was not without limitations, with protection zones being fairly coarse-grained, significant overhead both in terms of speed and effort, and difficulty in deployment as it required many separate components. The authors were also very clear that the browser extension space was still unmitigated. This was followed up by Dhawan et al. with Sabre [52]. The rest of the work in this space took to addressing the main problems that came about with solutions of this kind, mainly overhead, cost, developer effort, *etc.*, as well as introduce new variants.

ConScript by Meyerovich et al. moves script confinement and IFC to the client side by implementing IFC into the JavaScript engine in the browser. To address the developer effort issue, they also show is it possible to automatically generate security policies either by static code analysis or runtime analysis [6].

Guarnieri et al. took a static approach to secure information flow in their tool Actarus. They use static analysis to find vulnerabilities [53] and rewrite code, however they ignore model reflective calls such as eval.

Introducing the concept of secure multi-execution (SME) to IFC and the browser with FlowFox [54], De Groef et al. utilized a two level security model and implemented a full browser. Their overhead was notable, but proved SME had life in this space.

Giffin et al. introduced Hails, which tackles private data sharing further up the supply chain by adding mandatory access control to the Model-View-Controller (MVC) architecture [55]. Part of their solution involved users installing an extension to provide confinement in browser.

Richards et al. approached IFC from a different technical angle by using delimited histories with revocation as a mechanism for IFC policy enforcement [56].

Kerschbaumer et al. implemented IFC in the JavaScript engine of WebKit for the explict purpose of preventing sensitive data exfiltration, as it monitored network requests [57]. To address runtime issues with IFC in general, Kerschbaumer et al. switched the information tracking load from JavaScript interpreters to just-in-time compilers, reducing the overhead [58].

To address the difficulty in getting changes into browsers and their engines, Magazinius et al. proposed multiple different supporting architectures to inline security monitors for JavaScript including browser extensions, web proxies, suffix proxies, and integrators [59].

Hedin et al. created JSFlow, which enhanced the JavaScript interpreter, written in JavaScript as to be deployed as a browser extension, which tracks information flow even in the presence of libraries [60].

Stefan et al. created COWL, a label based mandatory access control enforcement mechanism at context boundaries. The DOM level API is implemented in the layout engine of browsers [3].

To address overhead issues, Kerschbaumer et al. introduced CrowdFlow. CrowdFlow probabilistically switches between two JavaScript interpreters, one for partial taint tracking and one for full information flow tracking. The switching is influenced by other users to increase accuracy [61].

Rajani et al. helped to bridge a gap in IFC by accounting for event handling and all of the DOM APIs in their instrumentation on WebKit [62].

Another inline tool, JEST, was created by Chudnov and Naumann utilising the no-sensitive-upgrade technique [63] .

To side-step the problems with both static and dynamic approaches to IFC, Hedin et al. developed a value-sensitive hybrid approach [64].

Trade-offs: Throughout the works listed above, there are many different limitations present. Often, these solutions are trading off between runtime overhead, developer effort, ease of deployment, backwards compatibility, and manual versus automatic labeling. Some works make progress on a few issues, but no single solution is able to combat all these compromises. Even an industry IFC project, named *FlowSafe* [65], has been discontinued over a decade ago.

2.6. Miscellaneous Approaches

Other solutions have also been proposed, yet they are often more restrictive or only partially address the problem.

Techniques: Solutions strip the ability of JavaScript to perform unsafe functions [66], ensure browser security with separation of contexts allowing for a multi-principal platform [67], apply CSP to browser extensions [68], fix and inject CSP into pages [69], verify the integrity of scripts with signatures [70], block malicous scripts all together [71], or secure the data pipeline surrounding user input using trusted hardware [72].

Trade-offs: Often, these solutions aid the issue, but suffer from similar trade-offs to IFC solutions, and have

less coverage, meaning they address a subset of the problem, or a tangential problem.

3. Monitoring HTML Form Element Access on 100,000 Websites

With all the proposed countermeasures and remedies discussed in the previous section, the question remains: why has nothing stuck? Seemingly, all browser-based projects have been abandoned, and research has shifted its focus towards other fields over the course of a decade. Third-party data exfiltration and abuse is still happening on the web today. In this section, we focus on third-party scripts and their interaction with user supplied PII to empirically shed light on the complicated nature of third-party scripts operating on user-supplied data in the wild.

3.1. Monitor

We implement a monitoring system within Firefox (v.88, maintained through v.92) that allows the inspection of all JavaScript to DOM communication within a web application. In more detail, this Monitor enables investigation of every `get()` and `set()` operation on DOM attributes which allows for deep inspection of every third-party script accessing DOM elements. Our generic implementation can capture all synchronous third-party JavaScript to DOM communication, regardless of how the attribute value is set. In more detail, our Monitor captures and monitors every form of setting an attribute, ranging from `element.setAttribute()` all the way to assigning a value using `attribue.value=`.

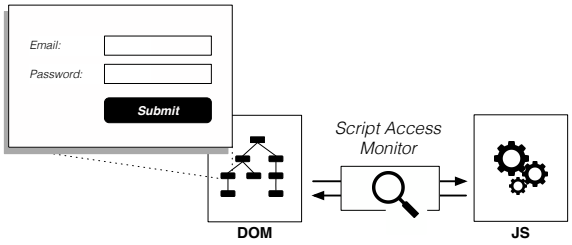

Figure 1: JavaScript to DOM Monitor to inspect, intercept and monitor all `get()` and `set()` methods on DOM Elements within a web application.

Loading a top-level page (i.e., the top-level HTML document [73]) and including third-party JavaScript into the same execution context grants the script the same capabilities as if the script is loaded from the same origin as the main application. As illustrated in Figure 1, our Monitor, which is implemented in the autogenerated DOM-Bindings, sits right between the JavaScript engine and the DOM. Our Monitor inspects the actual origin of the third-party script. Applying semantics of the same-origin-policy [74] precisely identifies what information the third-party script operates on. For this work, we focus on third-party JavaScript accessing HTML Form Elements because those commonly contain highly-sensitive user PII which highlights the problematic access situation.

3.2. Inspector

To utilise the aforementioned Monitor, we implement a web crawler to expose script behavior on sites. Our crawler (code available here: https://gitlab.com/swsprec/sok-allornothing) is implemented using Selenium [8] to drive user-like form interactions on top of our Monitor. This enables the capture of any element access and any kind of form access before and after forms are submitted. On each site, the crawler first looks for a form; if none is found, it will go visit five internal pages found from links presented on the landing page. These links are sorted and selected using a manually curated list of keywords identifying pages most likely containing forms. Once a form is found, we employ a similar method to Acar et al.'s bait technique [2] to induce script behavior. This technique supplies the appropriate input to HTML forms and input fields which are the main pathway for websites to query information from users. Generally, users supply PII through common form elements such as `<textarea>`, and `<input>` [75]. The more flexible `<input>` is of particular interest because of the vast number of input types it supports, such as `email`, `password`, `tel`, and `text` [76]. Our Inspector handles all these possible input forms, and supplies the relevant data for each. We determine what each form input field is asking for, either by the input type [76] or by keyword hints in the HTML element attributes, and check that the field is visible to the user. Finally, our Inspector simulates a user typing the response from our user data global configuration and it attempts to submit the form.

3.3. Measurement Ethics

Since our measurement involves actively interacting with webpages, it is important that we do so as ethical Internet citizens. We closely follow all the guidance set out in prior research and the Menlo Report [77]. We rate limit, scan infrequently, maintain a "no-scan" list and the ability for sites to opt-out, and host an informative study splash page on the scan server.

3.4. Infrastructure

Our measurement was carried out on AWS in *us-central1-a* on a *e2-standard-32* with 32 vCPUs and 128 GB of memory. The scan server was running Ubuntu 18.04 LTS with 4 TB of persistent disk, as well as hosting our scan splash page with an Apache web server. The measurement began in late 2020 and lasted five days.

3.5. Input List

We crawled 100,000 websites from the Tranco ranking list [78] created in late 2020. To observe both high traffic and low traffic sites, we sampled similarly to previous work [2] as follows: all of the top 30,000 sites; a 30,000 random sample from ranks [30,000 - 100,000); a random sample of 40,000 sites from ranks [100,000 - 1,000,000).

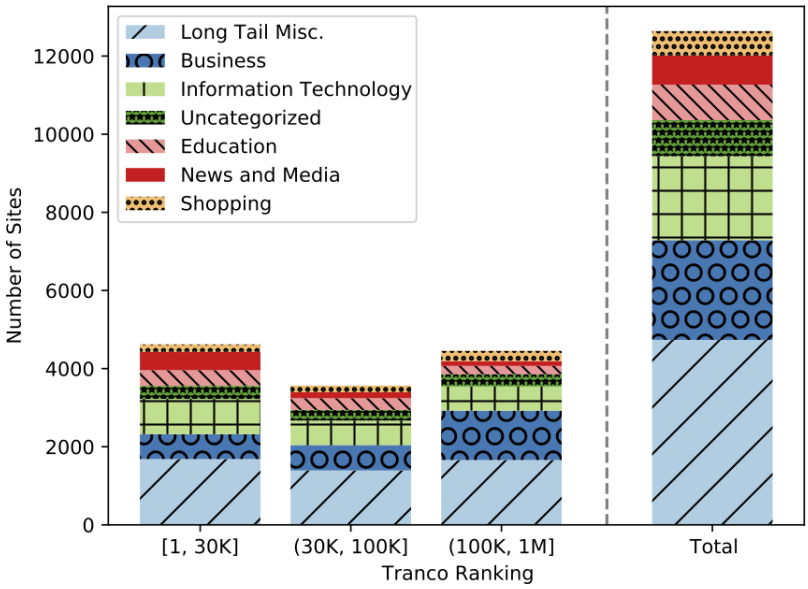

Figure 2: Tranco ranking and website categories of sites positively attributed to have a third-party script operating on bait PII.

3.6. Sites Interacted with and Baited

Of the 100,000 websites crawled, only 76,238 sites had a form our crawler identified. Sites which had forms but no injected bait PII account for 39,939 sites, 11,311 of which had an error, and the remaining had hidden input fields (our crawler ignores these by default). Of the sites with errors, 6,333 were timeouts, 3,650 were webdriver exceptions and the remaining 1,328 errors were from exceptions including stale element references, click interceptions, elements not interactable, unexpected alerts, and other unexpected behavior.

Our Inspector interacted and input PII on 36,299 sites. After combining our Inspector logs with our filtered Monitor data, we were left with 12,638 sites on which we positively attributed third-party scripts operating on our bait PII. This filtering is further explained in Section 3.7. These 12,638 sites are distributed almost identically over the input list buckets and the initial input list as seen in Figure 2.

Figure 2 reports on the category of these sites and their respective breakdowns over the input buckets described above. We categorized websites using Fortiguard [79], and report the top five categories of *Business, Information Technology, Education, News and Media, and Shopping* plus *Uncategorized* and *Long Tail Miscellaneous* sites.

Long Tail Miscellaneous contains 86 different categories of web applications, with their prevalance decaying exponentially. It is important to note that this distribution is not meant to be extrapolated as representative of the wider Internet, but rather it shows two important facts.

First, third-party scripts that access user PII can be found on many different categories of sites indicating this issue is spread out over the Internet.

And second, there exists a long tail of categories reinforcing the idea that this issue is pervasive regardless of web application function.

3.7. Data Filtering from the Monitor

We process our data in multiple stages in order to reveal what is happening on sites.

Stage 0 - The Monitor filters in accordance with the SOP, thus it logs any script access not from the same origin performing a `get()` or `set()` method on the main application.

Stage 1- Logs are verified for valid JSON, and we prune extraneous Selenium log statements from the Inspector.

Stage 2 - We utilise the public suffix list [80] to extract hostnames from third-party URLs and map them to their parent entity using DuckDuckGo's Tracker Radar [81]. For example, if a script originating from *doubleclick.com* accessed the login field on *firebase.com*, both domains are Google owned properties, so we treat them as first party scripts.

Stage 3 - We filter logs for requests that contain the PII we injected as bait. The Monitor logs the value of the element being accessed, so we search for matches with supplied PII in the scan's global configuration.

On Our Data: We do not aim to present ground truth about the distribution of third-party scripts, but rather to investigate the complexity added to the problem through their vast differences. To this effect, our system has false-negatives, and our methods of identifying similar scripts are not exhaustive. We have no false positives, as our Monitor has a perfect view of all operations on fields, though we leave out scripts that utilise innerHTML to access data for parsing simplicity and fail to attribute some operations due to domain mismatching between the Monitor and Inspector. We only report on scripts that operated on our bait user PII, and that we can attribute through the combination of our Monitor and Inspector.

Script Consolidation: We compute a SHA1 hash of each third-party script encountered in order to combine

TABLE 1: Password field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://mc.yandex.ru/metrika/watch.js	UBA / Session Replay
https://mc.yandex.ru/metrika/tag.js	UBA / Session Replay
https://s.pinimg.com/ct/lib/main.2424edb5.js	Action Tracking

(b) Top Script Hosting Domains

Domain	Category
facebook.net	UBA / Data Aggregator
gstatic.com	Static Files
ajax.googleapis.com	Helper Library
yandex.ru	UBA / Session Replay
cloudflare.com	CDN

TABLE 2: User field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://mc.yandex.ru/metrika/watch.js	UBA / Session Replay
https://cdn.permutive.com/...-web.js	Modeled Ad Targeting
https://mc.yandex.ru/metrika/tag.js	UBA / Session Replay

(b) Top Script Hosting Domains

Domain	Category
gstatic.com	Static Files
facebook.net	UBA / Data Aggregator
ajax.googleapis.com	Helper Library
yandex.ru	UBA / Session Replay
jquery.com	Helper Library

TABLE 3: Email field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://mc.yandex.ru/metrika/tag.js	UBA / Session Replay
https://s.pinimg.com/ct/lib/main.2424edb5.js	Action Tracking
https://mc.yandex.ru/metrika/watch.js	UBA / Session Replay

(b) Top Script Hosting Domains

Domain	Category
facebook.net	UBA / Data Aggregator
gstatic.com	Static Files
ajax.googleapis.com	Helper Library
yandex.ru	UBA / Session Replay
jquery.com	Helper Library

scripts that are identical but hosted by different sources. We download scripts at a different time than our scan, therfore our hashing method is not exhaustive. We modified third-party script URLs for presentation (e.g., "..." in URLs).

3.8. Script Categorization

We manually categorize the top third-party scripts and their hosting domains by domain expertise, respective technical documentations, and marketing materials. Scripts and Domains fit the categories detailed in Table 5. Categories range from the security benefits of *Abuse Mitigation* scripts, to privacy violating *Session Replay* scripts, to seemingly neutral *Helper Library* scripts. We see the full range of these categories operating on the password, username, and email fields as shown in Tables 1, 2, 3. Another issue that solutions must successfully overcome is the varied use cases of scripts. While some scripts in privacy violating categories could be outright blocked, as most are by privacy preserving browser extensions, a blanket solution of removing third-party access would remove the benefits users gain through collaterally disabling those scripts as well. Complicating the matter further, scripts in categories similar to *Helper Library* could be utilized by other scripts for both good and bad uses. This demonstrates the need for solutions to be able to discriminate on these actions on the per-action level and not on the per-script level.

3.9. Script Discrimination On User Input

Table 4 reports which third-party scripts operate the most on our supplied user PII, and if they either indiscriminately grab all available user PII or discriminate over what fields they operate on. Scripts listed as *Never* discriminates are those who operate on all available PII, while those listed as *Sometimes* discriminates have been seen to operate on some subset of available PII, not all.

Scripts that *Sometimes* discriminate on PII could be thought of as less privacy violating, since they show restraint, yet we see an immediate counter example in `yandex.ru/.../tag.js`. This script *Sometimes* discriminates because it is

TABLE 4: Top scripts discrimination, category, and unique input sets operated over

Script	Discriminates	Category	Unique PII
https://www.gstatic.com/.../recaptcha__en.js	Sometimes	Abuse Mitigation	346
JQuery	Sometimes	Helper Library	182
https://mc.yandex.ru/metrika/tag.js	Never	UBA / Session Replay	59
https://mc.yandex.ru/metrika/watch.js	Sometimes	UBA / Session Replay	50
https://s.pinimg.com/ct/lib/main.2424edb5.js	Never	Action Tracking	38
https://s.adroll.com/j/sendrolling.js	Never	CRM / Email Marketing	62
https://js.hscollectedforms.net/collectedforms.js	Never	CRM	50
https://js.hsforms.net/forms/current.js Also v2.js	Never	CRM	32
https://s3.amazonaws.com/downloads.mailchimp.com/js/mc-validate.js	Never	CRM / Email Marketing	14
Marketo forms2.min.js	Sometimes	Form Builder / CRM	23
...	...	...	...
https://assets.squarespace.com/.../common-...-min.en-US.js	Never	Website Builder	5
...	...	...	...

TABLE 5: Script categories

Categories	Description
Abuse Mitigation	Identify and secure against bots and site abuse
Action Tracking	Capture user interactions and event tracking like "Checkout" and "AddToCart"
Content Delivery Network (CDN)	Provide availability and performance through distributed servers
Customer Relationship Management (CRM)	Collect data for managing interactions with customers such as follow-up emails
Data Aggregator	Collect data on users of many different types
Email Marketing	Focus on CRM through email management
Form Builder	Allow to build custom forms
Helper Library	Provide functionality to website developers
Session Replay	Record all user interactions for the purpose of enabling website owners to replay a live session to the same end of UBA
Static Files	Serve static, generally unchanging files
User Behavioral Analysis (UBA)	Capture user interaction with a site to illuminate user behavior for website owners
Website Builder	Development aid and display of a website

meant to be included on any site with little to no integration, and thus will not have full access unlike its counterpart `yandex.ru/.../watch.js`. Conversely, scripts that *Never* discriminate on PII could be thought of as primarily privacy violating, yet we see yet another counter example in `assets.squarespace.com/.../common-....js`. This script *Never* discriminates on PII, but is mainly used for assisting website developers with additional functionality, yet it is third-party to the main page.

Seeing scripts behaving this way presents yet another hurdle for solutions in this space to content with. We see that some forms of behavioral analysis do not lend clear cut answers to the intentions of these scripts. Solutions need to be more sophisticated and judicious than solely relying on behavioral indicators such as input discrimination, though they can certainly benefit from utilizing things like this to determine the reputation of a particular script indicating the need for further in-depth investigation.

Table 4 also shows the number of unique PII combinations that scripts are seen in the presence of. For example, a script could be seen on a page where *email* and *password* were baited, and also on a page where *email*, *password*, and *telephone number* were baited. These two pages have a unique set of PII, which shows the flexibility and variability of environments scripts are operating in. This makes the discrimination classification of *Never* worse, since these scripts operate on all the PII they have access to with great flexibility.

3.10. Script Diversity and Complexity

For a full listing of top scripts and top script sources that operate on specific user PII fields, refer to Appendix A. It is clear there are many different types of use cases for scripts to operate on user PII. We see subtle differences between the types of scripts operating on different PII fields, but script uses across Tables 1, 2, 3 remain fairly consistent.

When thinking in a security mindset, we would expect a field as sensitive as the password field to be more protected than others. Despite our security hopefullness, Table 1a shows use cases including `gstatic.com/.../recaptcha__en.js` providing

security features, `JQuery` making element access simpler to an unknown end, and `yandex.ru/.../tag.js` which is used for UBA and Session Replay tracking. The top source domains for these password operating third-party scripts show a similar picture despite the more coarse-grained view it provides. Table 1b combines all URLs from `gstatic.com` into one line item, which is categorized as a domain that serves static files. A majority of these files are `recaptcha__en.js`, but the domain itself serves other purposes as well. It is evident that for a solution to be successful, it must contend with this extra complexity.

3.11. Summary and Key Takeaways

All of the above use cases confirm the convoluted nature for granting third-party scripts access to the main application, and endorse that there is no straightforward solution to overcome such a complex problem. We have seen that solutions need to be able to decide on a per-action basis if something should be allowed in order to have a perfect defense, and that simple heuristics cannot address these issues successfully alone. Our measurement also shows that these scripts are extremely prevalent on many different types of websites, indicating their widespread use, and the need for deployable soltuions for users.

4. Solution Rubric and Applications

We have gained many insights into what makes solutions viable or not through our related work deep dive and measurement thus far. In this section, we leverage these insights to propose a way to systematize and grade the potential success and efficacy of proposed solutions to secure users against third-party PII access and exfiltration, completing our postmortem.

Table 6 applies the grading rubric to most relevant solutions covered in Section 2.

4.1. Rubric Items

In this section we detail each rubric item, how to score a technique, and how our insights have lead to these items.

Level of Control: This item serves to indicate the amount of visibility and control a solution has in the browser and or JavaScript engine. Typically, this is thought of as coarse-grained versus fine-grained control over a system. In our application, it is important to distinguish between control over script-level or action-level. A solution that has script-level controls can only allow or disallow an entire script from running, whereas a solution with action-level controls can allow scripts to run benign behavior and disallow only malicious actions.

Scores: [`action-level, in between, script-level`]

Application: Determining what grade to give a particular solution is based on the amount of control it provides.

Motivation: It was made evident by our measurement that defense techniques are highly subject to the level of control they are able to operate with. Our measurement showed a diverse landscape of script activity, indicating that the level of control solutions have factors heavily into the benefit they can provide on the modern Web.

Ease of Deployment: We have seen techniques fail to be adopted because of the difficulty deploying the solution in the real world. This complexity relates to the number of parties needed to make changes, and the complexity of these changes. Solutions that require browser developers, web application developers, and users to fully deploy are much more difficult than solutions that are solely in the browser.

Scores: [`single party, single complex, multiple easy, multiple parties`]

Application: The only subjective part of this grading lies in the complexity of deployment. This determination will be guided by the level of technical prowess required plus quantity of work.

Motivation: Our review of previously proposed solutions saw that the more complicated deployment schemes were, no matter how effective the solution, the less likely they were to be adopted in the long run indicating its importance in the success of a solution.

Maintainability: This rubric item is very similar to Ease of Deployment, but instead of deployment, it focuses on the potential effort in keeping the solution up to date over time and platform changes. If a technique is too difficult to successfully maintain while the underlying product evolves, it is never adopted.

Scores: [`stable, in between, volatile`]

Application: The location of the technique determines the score for this rubric item. For example, if a solution is spread out over multiple parts of the browser that change somewhat frequently, than it would be quite volatile. Whereas solutions that are fully self-contained in a single area of code that is generally stable, updating and maintaining the technique over time will be significantly easier.

Motivation: An important factor in any software system that is deployed is the ease of maintaining that system. Industry has shown, by abandoning projects discussed earlier, if they asses the costs of a solution higher than the benefits a solution is abandoned. Thus, a successful system will be one that weighs its costs carefully, maintainability being one key component of that.

Developer Overhead: When web application developers have to expend a lot of effort to apply solutions, we see almost no adoption. This problem is exacerbated if current web applications need to be rewritten in order to take advantage of the new benefits.

Scores: [`no effort, tool assisted, some manual effort, full rewriting`]

Application: Grading this rubric item is straightforward, if there is a tool to assist developers in enacting a technique or with policy expression, then they fall into that category, otherwise the other grades are applied respectively.

TABLE 6: Rubric for solutions to the third-party script inclusion permission model

Solutions		Level of Control	Ease of Deployment	Maintainability	Developer Overhead	Performance Overhead	Backwards Compatibility and Functionality	Level of Security	End User Usability
Isolation and Confinement	§2.4								
JaTE - JavaScript Confinement	[40]	◐	○	●	○	◐	◐	◐	◐
Isolation by Static Rewriting	[41]–[44]	◐	○	●	○	†	◐	○	●
Information Flow Control	§2.5								
Dynamic Data Tainting and Static Analysis	[4]	●	◐	○	N/A	●	●	●	○
OMash - Object Abstractions	[50]	◐	†	◐	†	†	◐	◐	●
BFlow	[51]	◐	†	◐	†	†	○	●	◐
ConScript	[6]	●	†	◐	◐	●	◐	●	●
FlowFox - Secure mulit-execution	[54]	◐	†	○	○	◐	◐	●	●
Flexible Access Control	[56]	●	○	◐	○	◐	●	●	●
WebKit Information Flow	[57]	●	◐	◐	●	○	●	●	●
IFC with Just-In-Time Compilation	[58]	●	◐	○	●	○	●	●	●
JSFlow	[60]	●	◐	●	○	†	◐	◐	◐
COWL	[3]	●	◐	◐	○	◐	◐	●	●
CrowdFlow	[61]	●	◐	◐	●	◐	●	◐	●
JEST	[63]	●	○	○	○	†	◐	●	◐
Other Approaches	§2.6								
ScriptProtect	[66]	◐	●	○	●	●	◐	◐	●
Injecting CSP	[69]	○	◐	◐	●	●	○	◐	◐

Level Of Control	Ease of Deployment	Maintainability	Developer Overhead	Performance Overhead	Backwards Compatibility and Functionality	Level of Security	End User Usability
●– action-level	●– single party	●– stable	●– no effort	●– no noticable - 15%	● protects backwards with some breakage or additional work	●– full	●– no skill
◐– in between	◐– single complex	◐– in between	◐– tool assisted	◐– 15% - 50%	◐ protects in place and ignores legacy applications	◐– partial	◐– low skill
○– script-level	○– multiple easy	○– volatile	○– some manual effort	○– 51% - 100%	○– breaks sites	○– introduces new flaws	○– medium skill
	† – multiple complex		† – full rewriting	† – not reported - 101%+			† – advanced skill

Motivation: The reason that XSS is still one of the most pervasive vulnerabilities on the Web today is that the onus of solution lies on each individual web developer. For a solution to be adopted in the modern Web, the amount of Developer Overhead incurred has a significant role in the cost benefit calculation industry must go through. We see many of the studied solutions in Section 2 incur developer effort, factoring into their failure to be adopted.

Performance Overhead: One of the biggest killers of solutions lies in the overhead they add to the browsing of users. It is clear that browsers and web application developers are sensitive to slowdowns of any nature, and thus the successful adoption of a technique is very closely tied to performance overhead.

Scores: [`no noticeable - 15%, 15% - 50%, 51% - 100%, not reported - 101%+`]

Application: Different techniques report their overhead differently, so this section is meant as a best effort determination. The base to measure overhead on top of is relative to the technique, but the effect is ultimately the same thing, where operations either in a JavaScript engine, or browser, or elsewhere are slowed down affecting upstream performance.

Motivation: Many of the proposed solutions in our survey have fantastic promises of perfect visibility into script action, yet suffer from tremendous performance overhead. Despite the field looking to lower the performance overhead over the years through innovated changes, many of the solutions still slow down performance significantly. Browser vendors and users are typically very sensitive to performance hits, making it an important part of the cost benefit analysis to a solution.

Backwards Compatibility and Functionality: Techniques have been seen to break functionality of websites, particularly those that outright block scripts from running, or disallow JavaScript entirely. The web community is particularly sensitive to maintaining backwards compatibility, and maintaining users ability to access sites without developers having to constantly update their codebases. For solutions to stand much

chance they need to take this into account. The best solutions are those that can provide a level of protection for sites that have not been updated as well.

Scores: [`protects backwards with some breakage or additional work, protects in place and ignores legacy applications, breaks sites`]

Application: The application of grades here is straightforward, for solutions that only apply to sites that deploy a change, they fall into the protects in place and ignores legacy application grade, whereas if it provides benefit to all sites, it can be said to protect backwards.

Motivation: As previously mentioned, the web community is very sensitive to maintaining backwards compatibility. The solutions in our survey generally acknowledge this and attempt to account for it when presenting their solutions, making it clearly an important factor when evaluating techniques.

Level of Security: This rubric item denotes how much protection is provided to a user under a specified technique. If a user is fully protected from third-party scripts misusing their data, that particular solution can gain traction for deployment. If a solution only provides protection in certain cases, there is less reason to cope with any other drawbacks.

Scores: [`full, partial, introduces new flaws`]

Application: If a technique itself introduces a new vulnerability, which we have heard of anecdotally, whole projects in industry tend to be abandoned when taking into account the other items of this rubric. The rest of the items are self explanatory.

Motivation: Ultimately, the level of security a solution in the space provides is the benefit in the cost benefit analysis done by industry. We observed in our survey that some industry attempted solutions had either mixed benefits to security, or even introduced flaws, leading to their abandonment. Our measurement also gave insight that with such a complex ecosystem, differing levels of security depending on the use case could be acceptable, making it clear as a valuable part in evaluating techniques.

End User Usability: The skill level required to deploy and use effectively on the users end ultimately makes or breaks the wide-spread adoption of any techniques.

Scores: [`no skill, low skill, medium skill, advanced skill`]

Application: An advanced user can complete complex tasks such as setting up proxies, or interfacing directly with code. Medium skill can be considered tasks like tweaking settings directly, and low skill can be actions similar to installing a browser extension.

Motivation: Many works in our survey purposefully target solutions without much end user interaction. Those that do, make a concerted effort to minimize the requirement on users, stressing its importance to the community, and in getting anything adopted at scale. Thus, end user usability is valuable to account for when looking at solution techniques.

4.2. Key Takeaways

It is clear from Table 6 that no single proposed solution is without trade-offs. There are many reasons why each of these techniques has to make these trade-offs, yet the fact that none of them have been adopted for users to benefit from indicates that either they are not making the proper trade-offs, or they are not acceptable enough for deployment. The only class of solutions that were deployed commercially at any point were the `Isolation by Static Rewriting` solutions, and we see they sacrificed security for usability, ease of deployment, and low developer overhead. Other solutions in this space focus on providing a very high level of security but fall short in overhead, ease of deployment, and maintainability.

It appears that to be able to sell an idea, minimizing back end costs (development effort, overhead, deployment, *etc.*) helps, but if it cannot deliver enough security for users, ultimately the hassle is not deemed worth it.

An ideal solution would not have to make any trade-offs, which is generally an impossibility, but in order to afford users any semblance of protection, a solution should try and maximize back end simplicity in order to see adoption. Ultimately, some protection in this space is better than none.

While IFC provides the best protection for users, it is generally uncompromising when it comes to back end costs. We see some solutions making headway on reducing performance overhead, but the field has been dormant for the last few years with no commercial solutions available for users.

Looking at solutions from the angle of feasibility for adoption, it would be beneficial to move trade-off decision making to end users instead of browsers and web application developers. The value analysis of tools varies drastically between these parties and could be used to provide more benefit to those parties who desire it. Thus, a possible new avenue for solutions in this vein lies in leveraging deep introspection hooks similar to Virtual Machine Introspection (VMI) to provide an interface for the community to decide their own trade-offs. We discuss directions for future work further in Section 5.

5. A Path Forward

Currently, all web based applications have to find a compromise between granting third-party scripts full access or no access to application internals. We have shown that it is almost impossible for web application developers to find the right balance between taking advantage of the interoperability of the modern web while simultaneously not putting end users' security and privacy at risk.

In Section 2 we have shown that no proposed solution addresses all trade-offs sufficiently, and so far none of the proposed solutions have found adoption in any of the mainstream web browsers in industry. Solutions struggle to navigate between slowing down runtime, adding onerous developer effort, scaling poorly, and lacking backwards compatibility. Browser vendors are not going to compromise when it comes to many of these trade-offs, particularly when it comes to trading performance for "slightly" improved security and privacy. None of the proposed approaches provide bulletproof evidence that

TABLE 7: Rubric application for future direction solution

Solutions	Level of Control	Ease of Deployment	Maintainability	Developer Overhead	Performance Overhead	Backwards Compatibility and Functionality	Level of Security	End User Usability
Deep Hooks and addon	●	○	●	●	Choice	Choice	Choice	◐

they actually stop privacy violations or at least prevent user data exfiltration in an industry acceptable fashion.

Our measurements in Section 3 further highlight the complexity of third-party script behavior operating on user supplied PII. As we have shown, script access on user PII ranges from clearly positive security features, to privacy violating session recordings. Simple coarse-grained solutions, such as isolating the password field, will not work in a space this complex with such varied use cases.

We combine these insights into a new systematization and way to evaluate technical solutions in Section 4. It is clear that previous solutions have their own failings resulting in lack of commercialization, but a path forward is illuminated.

A clear solution lies in allowing tools to be built that distribute the power of making access decisions. By looking at browsers as complex software systems akin to operating systems, deep introspection hooks similar to VMI present an interesting model. We argue that the research and web community needs to be empowered to build prototypes, probably in the form of addons to inspect and monitor third-party JavaScript accessing and operating on PII.

We note that implementing and maintaining our Monitor over multiple Firefox versions has resulted in manageable effort. Browser vendors could theoretically expose similar APIs which would ease the creation of research prototypes. Such monitoring APIs would allow the community to conduct studies, implement their own solutions, and could even empower end users to make their own decisions about which third-party they trust or do not trust to operate on their PII.

For the solution of deploying deep introspection hooks and an addon, we apply our rubric as shown in Table 7. The Level of Control with this technique enables monitoring at the `action-level`, Ease of Deployment is rated as `multiple easy` since solutions need an addon, plus browser modification which we have done with our Monitor. Maintaining our Monitor has resulted in manageable effort, as we placed our hooks well, allowing for the grade of `stable`. As web application developers do not need to make any changes, and just a single addon needs to be developed, Developer Overhead is `no effort`. Performance Overhead, Backwards Compatibility and Functionality, and the Level of Security are all left up to choice. The user and addon get to choose what they are willing to give up in these three domains, and since end users need only to install an addon End User Usability is graded `low skill`. A solution as this minimizes back end effort and decisions to aid in adoption, while allowing costs to be decided and incurred by only those parties desiring solutions.

As an example of an analogous problem and success story we argue that the famous browser extension *HTTPS Everywhere* [82] was released in 2014, but it was only just recently, in late 2020 that Mozilla integrated an *HTTPS-Only Mode* [83] into Firefox. We argue that *HTTPS Everywhere* sufficiently highlighted the benefits which finally allowed browser vendors to incorporate such a solution. We further argue that it will take monitoring systems which similarly allow inspection of third-party script accessing end user PII to facilitate adoption.

Ultimately, end users have the right to transparency over how their PII is being handled. We think that providing hooks for deep inspection is a viable way forward to generate transparency without forcing end users to randomly trust any third-party scripts that a web application pulls together to create a service.

6. Conclusion

Currently, the web execution model allows sites to leverage third-party JavaScript in a single execution context. Despite the knowledge that this execution model allows privacy violations and user data exfiltration, the general model has been unchanged for over a decade.

We have presented a postmortem of solutions attempting to overcome the all or nothing permission model of current web and browser architecture. We evaluated proposed solutions, discussed their trade-offs, and explained the lack of industry adoption of all of the proposed solutions. We further created a Monitor in the Firefox web browser which allows for the capture of third-party script access of user supplied PII in HTML Form Elements. By inspecting 100,000 websites using our Monitor and a custom web crawler, we have emphasized the complexity of use cases of third-party scripts operating on user PII.

We then leverage our insights to present a novel systematization and grading rubric for solutions in this space, apply it to notable solutions, and draw informed conclusions from it.

In summary, we adhere that trade-offs between overhead, user requirements, security flaws, backwards compatibility, and site breakage are complicated to navigate. Our findings suggest that browsers could offer interfaces for deep inspection of third-party script access PII in the main application. We believe that such interfaces would allow the community to close the gap between the all or nothing inclusion model, and would enable informed decisions for web application developers and end users.

Acknowledgements

The authors are grateful to Giancarlo Pellegrino for shepherding the paper, to Emily Kubik, Mark Melendy, Bahruz Jabiyev, Dakota, and the anonymous reviewers for their constructive feedback. This work was partially funded by the National Science Foundation grant CNS-1703454, by Secure Business Austria, and by a travel grant from Mozilla Corporation.

References

[1] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "You are what you include: Large-scale evaluation of remote javascript inclusions," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2012.

[2] G. Acar, S. Englehardt, and A. Narayanan, "No boundaries: data exfiltration by third parties embedded on web pages," *Proceedings on Privacy Enhancing Technologies*, 2020.

[3] D. Stefan, E. Z. Yang, P. Marchenko, A. Russo, D. Herman, B. Karp, and D. Mazières, "Protecting users by confining javascript with cowl," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[4] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Cross site scripting prevention with dynamic data tainting and static analysis." in *Proceedings of the Network and Distributed System Security Symposium*, 2007.

[5] D. Y. Zhu, J. Jung, D. Song, T. Kohno, and D. Wetherall, "TaintEraser: protecting sensitive data leaks using application-level taint tracking," *ACM SIGOPS Operating Systems Review*, 2011.

[6] L. A. Meyerovich and B. Livshits, "Conscript: Specifying and enforcing fine-grained security policies for javascript in the browser," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2010.

[7] W3C, "Document Object Model (DOM)," http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/DOM3-Core.pdf, 2004.

[8] Selenium, "Selenium," https://www.selenium.dev, 2004.

[9] C. Yue and H. Wang, "Characterizing insecure javascript practices on the web," in *Proceedings of the World Wide Web*, 2009.

[10] F. Ocariza, K. Bajaj, K. Pattabiraman, and A. Mesbah, "An empirical study of client-side javascript bugs," in *Proceedings of the IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013.

[11] C. Kerschbaumer, T. Ritter, and F. Braun, "Hardening firefox against injection attacks," in *Proceedings of the IEEE European Symposium on Security and Privacy Workshops*. IEEE, 2020.

[12] B. Stock, M. Johns, M. Steffens, and M. Backes, "How the web tangled itself: Uncovering the history of client-side web (in)security," in *Proceedings of the USENIX Security Symposium*, 2017.

[13] T. van Goethem, P. Chen, N. Nikiforakis, L. Desmet, and W. Joosen, "Large-scale security analysis of the web: Challenges and findings," in *Proceedings of the International Conference on Trust and Trustworthy Computing*, 2014.

[14] N. Bielova, "Survey on javascript security policies and their enforcement mechanisms in a web browser," *The Journal of Logic and Algebraic Programming*, 2013.

[15] K. Singh, A. Moshchuk, H. J. Wang, and W. Lee, "On the incoherencies in web browser access control policies," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2010.

[16] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda, "Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web," in *Proceedings of the Network and Distributed System Security Symposium*, 2017.

[17] Y. Zhou and D. Evans, "Understanding and monitoring embedded web scripts," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2015.

[18] J. Jueckstock and A. Kapravelos, "Visiblev8: In-browser monitoring of javascript in the wild," in *Proceedings of the Internet Measurement Conference*, 2019.

[19] D. Kumar, Z. Ma, Z. Durumeric, A. Mirian, J. Mason, J. A. Halderman, and M. Bailey, "Security challenges in an increasingly tangled web," in *Proceedings of the World Wide Web Conference*, 2017.

[20] M. Ikram, R. Masood, G. Tyson, M. A. Kaafar, N. Loizon, and R. Ensafi, "The chain of implicit trust: An analysis of the web third-party resources loading," in *Proceedings of the World Wide Web Conference*, 2019.

[21] D. Jang, R. Jhala, S. Lerner, and H. Shacham, "An empirical study of privacy-violating information flows in javascript web applications," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2010.

[22] O. Starov, P. Gill, and N. Nikiforakis, "Are you sure you want to contact us? quantifying the leakage of pii via website contact forms," *Proceedings on Privacy Enhancing Technologies*, 2016.

[23] S. Van Acker, D. Hausknecht, and A. Sabelfeld, "Measuring login webpage security," in *Proceedings of the Symposium on Applied Computing*, 2017.

[24] B. Krishnamurthy and C. Wills, "Privacy diffusion on the web: a longitudinal perspective," in *Proceedings of the World Wide Web Conference*, 2009.

[25] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[26] WHATWG, "HTML Living Standard - The iframe element," https://html.spec.whatwg.org/multipage/iframe-embed-object.html#the-iframe-element, 2021.

[27] W3C, "Cross-Origin Resource Sharing," http://www.w3.org/TR/cors, 2010.

[28] W3C, "Content Security Policy," http://www.w3.org/TR/CSP2/, 2014.

[29] S. Van Acker, D. Hausknecht, and A. Sabelfeld, "Data exfiltration in the face of csp," in *Proceedings of the ACM Asia Conference on Computer and Communications Security*, 2016.

[30] M. Weissbacher, T. Lauinger, and W. Robertson, "Why is csp failing? trends and challenges in csp adoption," in *Research in Attacks, Intrusions and Defenses*, 2014.

[31] WHATWG, "HTML, 9.4.3 Posting messages," https://html.spec.whatwg.org/multipage/web-messaging.html, 2020.

[32] C. Jackson and H. J. Wang, "Subspace: Secure cross-domain communication for web mashups," in *Proceedings of the World Wide Web Conference*, 2007.

[33] F. De Keukelaere, S. Bhola, M. Steiner, S. Chari, and S. Yoshihama, "Smash: Secure component model for cross-domain mashups on unmodified browsers," in *Proceedings of the World Wide Web Conference*, 2008.

[34] A. Barth, C. Jackson, and J. C. Mitchell, "Securing frame communication in browsers," *Proceedings of the USENIX Security Symposium*, 2008.

[35] M. T. Louw, K. T. Ganesh, and V. N. Venkatakrishnan, "Adjail: Practical enforcement of confidentiality and integrity policies on web advertisements," in *Proceedings of the USENIX Security Symposium*, 2010.

[36] J. Terrace, S. R. Beard, and N. P. K. Katta, "Javascript in javascript (js.js): Sandboxing third-party scripts," in *Proceedings of the USENIX Conference on Web Application Development*, 2012.

[37] L. ingram and M. Walfish, "Treehouse: Javascript sandboxes to help web developers help themselves," in *Proceedings of the USENIX Annual Technical Conference*, 2012.

[38] P. Agten, S. Van Acker, Y. Brondsema, P. H. Phung, L. Desmet, and F. Piessens, "Jsand: Complete client-side sandboxing of third-party javascript without browser modifications," in *Proceedings of the Computer Security Applications Conference*, 2012.

[39] D. Akhawe, F. Li, W. He, P. Saxena, and D. Song, "Data-confined html5 applications," in *European Symposium on Research in Computer Security*, 2013.

[40] T. Tran, R. Pelizzi, and R. Sekar, "Jate: Transparent and efficient javascript confinement," in *Proceedings of the Computer Security Applications Conference*, 2015.

[41] Google, "Caja: The Caja Compiler is a tool for making third party HTML, CSS and JavaScript safe to embed in your website." https://github.com/googlearchive/caja, 2020.

[42] Adsafe, "Adsafe: Making JavaScript Safe for Advertising," https://www.adsafe.org/, 2021.

[43] Microsoft, "Microsoft Web Sandbox," https://web.archive.org/web/20120625221655/www.websandbox.org/, 2012.

[44] Facebook, "Facebook JavaScript," https://web.archive.org/web/20120104194744/http://developers.facebook.com/docs/fbjs/, 2012.

[45] S. Van Acker and A. Sabelfeld, "Javascript sandboxing: Isolating and restricting client-side javascript," in *Foundations of Security Analysis and Design VIII*, 2016.

[46] D. E. Denning and P. J. Denning, "Certification of Programs for Secure Information Flow," *Communications of the ACM*, 1977.

[47] A. C. Myers, "JFlow: practical mostly-static information flow control," in *Proceedings of the ACM Symposium on Principles of Programming Languages*, 1999.

[48] V. Simonet, "Flow Caml in a Nutshell," in *Proceedings of the APPSEM-II workshop*, 2003.

[49] E. Yang, D. Stefan, J. Mitchell, D. Mazières, P. Marchenko, and B. Karp, "Toward principled browser security," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2013.

[50] S. Crites, F. Hsu, and H. Chen, "Omash: Enabling secure web mashups via object abstractions," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2008.

[51] A. Yip, N. Narula, M. Krohn, and R. Morris, "Privacy-preserving browser-side scripting with bflow," in *Proceedings of the ACM European Conference on Computer Systems*, 2009.

[52] M. Dhawan and V. Ganapathy, "Analyzing Information Flow in JavaScript-Based Browser Extensions," in *Proceedings of the Computer Security Applications Conference*, 2009.

[53] S. Guarnieri, M. Pistoia, O. Tripp, J. Dolby, S. Teilhet, and R. Berg, "Saving the World Wide Web from Vulnerable JavaScript," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2011.

[54] W. De Groef, D. Devriese, N. Nikiforakis, and F. Piessens, "Flowfox: A web browser with flexible and precise information flow control," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2012.

[55] D. B. Giffin, A. Levy, D. Stefan, D. Terei, D. Mazières, J. C. Mitchell, and A. Russo, "Hails: Protecting data privacy in untrusted web applications," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2012.

[56] G. Richards, C. Hammer, F. Zappa Nardelli, S. Jagannathan, and J. Vitek, "Flexible access control for javascript," in *Proceedings of the ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications*, 2013.

[57] C. Kerschbaumer, E. Hennigan, P. Larsen, S. Brunthaler, and M. Franz, "Towards precise and efficient information flow control in web browsers," in *Proceedings of the International Conference on Trust and Trustworthy Computing*, 2013.

[58] C. Kerschbaumer, E. Hennigan, P. Larsen, S. Brunthaler, and M. Franz, "Information flow tracking meets just-in-time compilation," in *Proceedings of the ACM Transactions on Architecture and Code Optimization*, 2013.

[59] J. Magazinius, D. Hedin, and A. Sabelfeld, "Architectures for inlining security monitors in web applications," in *Proceedings of the International Symposium on Engineering Secure Software and Systems*, 2014.

[60] D. Hedin, A. Birgisson, L. Bello, and A. Sabelfeld, "Jsflow: Tracking information flow in javascript and its apis," in *Proceedings of the ACM Symposium on Applied Computing*, 2014.

[61] C. Kerschbaumer, E. Hennigan, P. Larsen, S. Brunthaler, and M. Franz, "Crowdflow: Efficient information flow security," in *Proceedings of the Information Security Conference*, 2015.

[62] V. Rajani, A. Bichhawat, D. Garg, and C. Hammer, "Information flow control for event handling and the dom in web browsers," in *Proceedings of the IEEE Computer Security Foundations Symposium*, 2015.

[63] A. Chudnov and D. A. Naumann, "Inlined information flow monitoring for javascript," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[64] D. Hedin, L. Bello, and A. Sabelfeld, "Value-sensitive hybrid information flow control for a javascript-like language," in *Proceedings of the IEEE Computer Security Foundations Symposium*, 2015.

[65] B. Eich, "Flowsafe," https://wiki.mozilla.org/FlowSafe, 2009.

[66] M. Musch, M. Steffens, S. Roth, B. Stock, and M. Johns, "Scriptprotect: Mitigating unsafe third-party javascript practices," in *Proceedings of the ACM Asia Conference on Computer and Communications Security*, 2019.

[67] H. J. Wang, X. Fan, J. Howell, and C. Jackson, "Protection and Communication Abstractions for Web Browsers in MashupOS," *ACM SIGOPS Operating Systems Review*, 2007.

[68] D. Hausknecht, J. Magazinius, and A. Sabelfeld, "May i? - content security policy endorsement for browser extensions," in *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2015.

[69] C. Kerschbaumer, S. Stamm, and S. Brunthaler, "Injecting csp for fun and security:," in *Proceedings of the International Conference on Information Systems Security and Privacy*, 2016.

[70] K. Nakhaei, F. Ansari, and E. Ansari, "Jssignature: eliminating third-party-hosted javascript infection threats using digital signatures," *SN Applied Sciences*, 2020.

[71] S. Arshad, A. Kharraz, and W. Robertson, "Include me out: In-browser detection of malicious third-party content inclusions," in *Proceedings of the International Conference on Financial Cryptography and Data Security*, 2016.

[72] S. Eskandarian, J. Cogan, S. Birnbaum, P. C. W. Brandon, D. Franke, F. Fraser, G. Garcia, E. Gong, H. T. Nguyen, T. K. Sethi, V. Subbiah, M. Backes, G. Pellegrino, and D. Boneh, "Fidelius: Protecting user secrets from compromised browsers," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2019.

[73] WHATWG, "HTML," https://html.spec.whatwg.org/, 2020.

[74] IETF, "The Web Origin Concept," https://tools.ietf.org/html/rfc6454, 2011.

[75] M. D. Network, "Web forms – Working with user data," https://developer.mozilla.org/en-US/docs/Learn/Forms, 2021.

[76] M. D. Network, "The Input element," https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input, 2021.

[77] D. Dittrich and E. Kenneally, "The Menlo Report: Ethical principles guiding information and communication technology research," U.S. Department of Homeland Security, Tech. Rep., 2012.

[78] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," *Proceedings of the Network and Distributed System Security Symposium*, 2019.

[79] FortiNet, "Fortiguard labs web filter," https://fortiguard.com/webfilter.

[80] Mozilla, "Public suffix list," https://publicsuffix.org/, 2020.

[81] DuckDuckGo, "Duckduckgo tracker radar," https://github.com/duckduckgo/tracker-radar, 2021.

[82] EFF, "HTTPS:// Everywhere," https://www.eff.org/https-everywhere, 2014.

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

Appendix

TABLE 8: Age field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://mc.yandex.ru/metrika/watch.js	UBA / Session Replay
https://mc.yandex.ru/metrika/tag.js	UBA / Session Replay
https://www.google.com/.../cse_element__en.js	Custom Search Engine

(b) Top Script Hosting Domains

Domain	Category
facebook.net	UBA / Data Aggregator
gstatic.com	Static Files
ajax.googleapis.com	Helper Library
yandex.ru	UBA / Session Replay
jquery.com	Helper Library

TABLE 9: City field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://fw.cdn.technolutions.net/.../base.js	CRM (For Higher Ed)
https://s.adroll.com/j/sendrolling.js	CRM / Email Marketing
https://maps.google.com/.../places_impl.js	Embedded Map

(b) Top Script Hosting Domains

Domain	Category
gstatic.com	Static Files
facebook.net	UBA / Data Aggregator
technolutions.net	CRM (Higher Ed)
ajax.googleapis.com	Helper Library
maps.googleapis.com	Embedded Maps

TABLE 10: Company field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
Marketo forms2.min.js	Form Builder / CRM
https://js.hsforms.net/forms/current.js	CRM
https://cdn.bizible.com/scripts/bizible.js	Marketing Attribution

(b) Top Script Hosting Domains

Domain	Category
facebook.net	UBA / Data Aggregator
gstatic.com	Static Files
marketo.com	CRM
hsforms.net	CRM
ajax.googleapis.com	Helper Library

TABLE 11: First_Name field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://js.hsforms.net/forms/current.js	CRM
https://s.adroll.com/j/sendrolling.js	CRM / Email Marketing
Marketo forms2.min.js	Form Builder / CRM

(b) Top Script Hosting Domains

Domain	Category
facebook.net	UBA / Data Aggregator
gstatic.com	Static Files
ajax.googleapis.com	Helper Library
hsforms.net	CRM
googletagmanager.com	UBA

TABLE 12: Full_Name field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://mc.yandex.ru/metrika/tag.js	UBA / Session Replay
https://mc.yandex.ru/metrika/watch.js	UBA / Session Replay
https://js.hscollectedforms.net/collectedforms.js	CRM

(b) Top Script Hosting Domains

Domain	Category
gstatic.com	Static Files
facebook.net	UBA / Data Aggregator
ajax.googleapis.com	Helper Library
yandex.ru	UBA / Session Replay
jquery.com	Helper Library

TABLE 13: Last_Name field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://js.hsforms.net/forms/current.js	CRM
Marketo forms2.min.js	Form Builder / CRM
https://s.adroll.com/j/sendrolling.js	CRM / Email Marketing

(b) Top Script Hosting Domains

Domain	Category
facebook.net	UBA / Data Aggregator
gstatic.com	Static Files
ajax.googleapis.com	Helper Library
hsforms.net	CRM
googletagmanager.com	UBA

TABLE 14: Message field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://mc.yandex.ru/metrika/tag.js	UBA / Session Replay
https://mc.yandex.ru/metrika/watch.js	UBA / Session Replay
https://s.adroll.com/j/sendrolling.js	CRM / Email Marketing

(b) Top Script Hosting Domains

Domain	Category
gstatic.com	Static Files
facebook.net	UBA / Data Aggregator
ajax.googleapis.com	Helper Library
yandex.ru	UBA / Session Replay
jquery.com	Helper Library

TABLE 15: State field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://s3...mailchimp.../mc-validate.js	CRM / Email Marketing
https://s.adroll.com/j/sendrolling.js	CRM / Email Marketing
https://maps.google.com/.../places_impl.js	Embedded Map

(b) Top Script Hosting Domains

Domain	Category
gstatic.com	Static Files
facebook.net	UBA / Data Aggregator
ajax.googleapis.com	Helper Library
s3.amazonaws.com	Cloud Storage
adroll.com	CRM / Email Marketing

TABLE 16: Street1 field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://maps.google.com/.../places_impl.js	Embedded Map
https://s.adroll.com/j/sendrolling.js	CRM / Email Marketing
https://mc.yandex.ru/metrika/watch.js	UBA / Session Replay

(b) Top Script Hosting Domains

Domain	Category
facebook.net	UBA / Data Aggregator
gstatic.com	Static Files
ajax.googleapis.com	Helper Library
maps.googleapis.com	Embedded Maps
yandex.ru	UBA / Session Replay

TABLE 17: Tel field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://js.hsforms.net/forms/current.js	CRM
https://s.adroll.com/j/sendrolling.js	CRM / Email Marketing
Marketo forms2.min.js	Form Builder / CRM

(b) Top Script Hosting Domains

Domain	Category
facebook.net	UBA / Data Aggregator
gstatic.com	Static Files
ajax.googleapis.com	Helper Library
yandex.ru	UBA / Session Replay
jquery.com	Helper Library

TABLE 18: Title field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://mc.yandex.ru/metrika/watch.js	UBA / Session Replay
Marketo forms2.min.js	Form Builder / CRM
https://mc.yandex.ru/metrika/tag.js	UBA / Session Replay

(b) Top Script Hosting Domains

Domain	Category
gstatic.com	Static Files
facebook.net	UBA / Data Aggregator
yandex.ru	UBA / Session Replay
marketo.com	CRM
ajax.googleapis.com	Helper Library

TABLE 19: Zip field and top parties operating on it

(a) Top Scripts

Script	Category
https://www.gstatic.com/.../recaptcha__en.js	Abuse Mitigation
JQuery	Helper Library
https://fw.cdn.technolutions.net/.../base.js	CRM (For Higher Ed)
https://s.adroll.com/j/sendrolling.js	CRM / Email Marketing
https://maps.google.com/.../places_impl.js	Embedded Map

(b) Top Script Hosting Domains

Domain	Category
facebook.net	UBA / Data Aggregator
gstatic.com	Static Files
ajax.googleapis.com	Helper Library
technolutions.net	CRM (Higher Ed)
googletagmanager.com	UBA