

COOKIEGUARD: Characterizing and Isolating the First-Party Cookie Jar

Pouneh Nikkhah Bahrami
pnikkhah@ucdavis.edu
University of California
Davis, USA

Aurore Fass
fass@cispa.de
CISPA Helmholtz Center for
Information Security
Saarbrücken, Germany

Zubair Shafiq
zubair@ucdavis.edu
University of California
Davis, USA

Abstract

As third-party cookies are being phased out or restricted by major browsers, first-party cookies are increasingly being used for web tracking. Prior work has shown that third-party scripts embedded in the main frame can access and exfiltrate first-party cookies—including those set by other third-party scripts. However, existing browser security mechanisms such as the Same-Origin Policy (SOP), Content Security Policy (CSP), and third-party storage partitioning do not prevent cross-domain access to first-party cookies in the main frame. While recent studies have begun to highlight this issue, there remains a lack of comprehensive measurement and practical defenses.

In this work, we conduct the first large-scale measurement and analysis of cross-domain access to first-party cookies in the main frame for 20,000 websites. We find that 56% of the websites include third-party scripts that exfiltrate first-party cookies that they did not originally set, and 32% where such scripts overwrite or delete these first-party cookies. To mitigate potential confidentiality and integrity risks due to this lack of isolation, we propose COOKIEGUARD, a browser-based runtime mechanism to isolate first-party cookies on a per-script-origin basis. COOKIEGUARD blocks unauthorized cross-domain cookie operations while preserving site functionality, with only 3% of the tested websites being affected by Single Sign-On (SSO) breakage. Our work highlights the risks posed by the lack of first-party cookie isolation in the current browser security model and offers a deployable path toward stronger protection.

CCS Concepts

• **Security and privacy** → *Web application security*.

Keywords

First-party Cookies; Isolation; Online Tracking; Third-party Scripts; Web Browsers

1 Introduction

As third-party cookies are being phased out or restricted by major browsers [1, 66–68], trackers are increasingly relying on first-party cookies for tracking [44]. One technique, known as *cookie ghost-writing*, allows third-party scripts embedded in the main frame to set cookies that browsers treat as first-party [11, 60]. Crucially, browsers do not distinguish between genuine first-party cookies

that are actually set by the first-party and those set by third-party scripts.

Scripts in the main frame (regardless of their origin) have full access to the first-party cookies in the cookie jar. This allows any third-party script in the main frame to read, modify, delete, or exfiltrate any first-party cookie, including those set by the first-party or other third-party scripts [11, 44]. Existing browser security mechanisms such as Same-Origin Policy (SOP), Content Security Policy (CSP), and partitioned storage (e.g., Safari’s ITP [58]) do not prevent such cross-domain access to first-party cookies (henceforth, *cross-domain cookie interaction*) in the main frame.¹ While this issue has recently gained attention [11, 60], prior work lacks both large-scale measurement and effective mitigation.

We address the following research questions in this work:

- (1) How prevalent are cross-domain cookie interactions?
- (2) Can we enforce domain-level isolation to mitigate this issue without causing website breakage?

To answer these research questions, we conduct a large-scale measurement and analysis of first-party cookies on 20,000 websites. We find that 56% of the websites include third-party scripts that exfiltrate cookies they did not set, and 32% where such scripts overwrite or delete these first-party cookies. To mitigate this issue, we present COOKIEGUARD, a prototype browser-based runtime mechanism to isolate first-party cookies on a per-script-domain basis. Implemented as a browser extension, COOKIEGUARD intercepts calls to `document.cookie` and `cookieStore`, allowing each script to access only the cookies it originally set. By enforcing policies based on execution context rather than script content, COOKIEGUARD is robust against code obfuscation since a script’s domain can still be identified regardless of code complexity. Moreover, unlike blocklist-based defenses [23, 24] that struggle against domain or URL manipulation [65], COOKIEGUARD does not rely on enumerating tracker domains; it enforces isolation across *all* domains by design, providing stronger and more durable protection. Our evaluation shows that COOKIEGUARD blocks all cross-domain cookie interactions while preserving functionality on most websites. It introduces a modest average page load overhead of 0.3 seconds and breaks Single Sign-On on 11% of the websites. Domain-level policies and organizational groupings help reduce this breakage to only 3%.

Our key contributions are as follows.

- (1) **Comprehensive measurement.** We conduct the first large-scale quantification of cross-domain cookie interactions



This work is licensed under a Creative Commons Attribution 4.0 International License.

¹We define a *cross-domain cookie interaction* as any read, write, delete, or exfiltration attempt by a script targeting a cookie it did not originally set.

(specifically overwrites and deletions) across 20,000 websites, going beyond prior work focused primarily on exfiltration [8, 11, 21, 22, 29, 44, 56, 60]. We further analyze the potential intent behind manipulations (e.g., competition vs. collusion) and treat exfiltration as a complementary dimension of our analysis.

- (2) **Attribution and use of first-party cookies.** We introduce domain-level attribution of cookie operations, identifying which scripts set, access, or manipulate cookies. Using this framework, we show how first-party cookies are increasingly used for tracking in the post-third-party cookie era, covering both `document.cookie` and the newer `CookieStore` API.
- (3) **Prototype enforcement and evaluation.** We design and implement `COOKIEGUARD`, a prototype browser extension that enforces per-script-domain isolation of first-party cookies. We evaluate `COOKIEGUARD`'s effectiveness in mitigating cross-domain exfiltration and manipulation, as well as its performance overhead and website breakage.

While `COOKIEGUARD` mitigates cross-domain cookie interactions, it does not address all security and privacy risks posed by third-party scripts in the main frame. More broadly, our findings underscore the need for finer-grained isolation in current browser security models. To support reproducibility and encourage further research, we make the source code of `COOKIEGUARD`'s browser extension implementation publicly available at <https://github.com/pouneh-nb/cookieGuard>.

2 Background

This section provides an overview of browser security mechanisms, script inclusion practices, and cookie management, with a focus on how first-party cookies can be accessed and manipulated by scripts in the main frame. An interested reader is referred to [71] for further discussion and background of cookies and online tracking.

2.1 Security Mechanisms

Modern browsers enforce several security models to protect users and enforce origin boundaries. We summarize those most relevant to script behavior and cookie access.

Same-Origin Policy (SOP). SOP [59] restricts interactions between resources from different origins, where an origin is defined by the scheme, host, and port. For example, `https://www.example.com:443` represents one origin. SOP isolates frames with different origins and prevents scripts embedded in frames of different origins from accessing cross-origin resources, helping mitigate attacks like cross-site request forgery. However, all scripts (from any domain) embedded in the main frame inherit the first-party origin and can access its resources.

In this paper, we explicitly distinguish between cross-origin and cross-domain interactions. The domain (eTLD+1) is part of the origin, which includes the full domain name, scheme, and port. For example, `https://example.com:8080` and `https://subdomain.example.com:8080` have different origins due to varied host names, despite the same scheme, port, and domain. We reserve the term cross-origin for SOP's strict definition, while we use cross-domain to

denote interactions between scripts from different eTLD+1 (or domains) but executing within the same main-frame origin. This distinction is important because our measurements and defenses in this paper focus on cross-domain interactions that deviate from cross-origin interactions that are targeted by SOP.

Content Security Policy (CSP). CSP [19] allows site owners to restrict the sources of executable scripts, images, and styles via HTTP headers or meta tags. While CSP allows some control over script inclusion, it does not regulate cookie access or define which scripts may read or modify cookies.

Storage partitioning. Some browsers have introduced storage partitioning to mitigate cross-site tracking. Safari implements partitioned cookies through Intelligent Tracking Prevention (ITP) [58], while Firefox enforces Total Cookie Protection, isolating all storage (including cookies) on a per-site basis [43]. Chrome is experimenting with Cookies Having Independent Partitioned State (CHIPS), an opt-in model for cookie partitioning under the Privacy Sandbox project [13]. Although these mechanisms limit third-party tracking across sites, they do not isolate scripts within the same top-level context—allowing third-party scripts in the main frame to access and manipulate first-party cookies.

2.2 Inclusion of Scripts

Scripts can be included either directly—through a `<script>` tag or inline JavaScript—or indirectly, when a loaded script dynamically inserts other scripts using `eval`, `import()`, or DOM APIs.

Scripts executing in the main frame, regardless of their original source, are treated as part of the first-party origin and granted full access to shared resources such as cookies and the DOM. SOP restricts cross-origin iframe content, but scripts in the main frame can freely interact with all main-frame resources unless explicitly restricted (e.g., via CSP or sandboxing).

2.3 Cookie Management in Browsers

Cookies are created and accessed either via HTTP headers or client-side JavaScript.

HTTP cookies. HTTP response header are classified as first-party if they originate from the same domain as the visited site. Otherwise, they are considered third-party. These cookies are automatically attached to future HTTP requests matching the domain (also depending on path and other cookie attributes). Cookies marked as `HttpOnly` are inaccessible to JavaScript, mitigating risks such as Cross-Site Scripting (XSS).

Script cookies. Scripts in the main frame can also create and access cookies using the `document.cookie` or `cookieStore` API, which provides a string interface to the browser's cookie jar. Although SOP governs access, the executing context is considered the visited site's origin. Thus, third-party scripts included in the main frame can create and read first-party cookies, regardless of their actual domain. These cookies are effectively indistinguishable from those set by actual first-party scripts.

document.cookie: The `document.cookie` provides an interface for reading and writing cookies. Any script in the main frame can access all first-party cookies for that origin, except those marked as `HttpOnly`. Although SOP restricts access to cookies based on origin, scripts inherit the origin of the main frame. For example,

if `tracker.com/myscript.js` runs on `example.com`, any cookies it sets are associated with `example.com`, granting it full access to all first-party cookies on that site.

CookieStore API: The CookieStore API is a modern interface that provides structured access to cookies via promises. Unlike `document.cookie`, it returns cookies as objects, making them easier to query and manipulate. Available in secure contexts (HTTPS), it is currently supported in Chrome, Safari, and Firefox Nightly [16]. To retrieve a specific cookie, a script can call `cookieStore.get("cookieName")`. To access all available cookies, the `cookieStore.getAll()` function can be used, which behaves similarly to `document.cookie` but returns a structured array of cookie objects.

Cookie categorization. Browsers traditionally distinguish between first-party and third-party cookies based on the domain attribute. However, this model misses a critical category: *ghost-written cookies*—cookies created by third-party scripts executing in the main frame. The cookies set by third-party scripts executing in the main frame are treated as first-party cookies. As prior work has shown [60], this leads to subtle security and privacy risks. In this paper, we focus on all first-party cookies created in the main frame, encompassing both genuine first-party and ghost-written first-party cookies.

3 Threat Model

Our threat model focuses on third-party scripts embedded in the main frame of a website. We consider the security and privacy risks posed by such scripts, which execute with the full privileges of the embedding origin.

As discussed in Section 2.1, modern browsers provide baseline defenses against untrusted content. CSP mitigates injection by constraining which scripts and other resources may be loaded or executed but once loaded, CSP does not govern what resources a script can access within a site. While SOP serves as the browser’s primary defense mechanism for isolating content across origins. Under SOP, scripts running in an `<iframe>` sourced from a different origin than the main frame are prohibited from accessing the main frame’s resources such as DOM and cookies. As illustrated in Figure 1, this boundary protects the main frame from untrusted cross-origin iframes, but it does not apply to scripts that execute within the main frame itself.

As a result, third-party scripts embedded in the main frame inherit all the privileges of the first-party origin, giving them access to the DOM, cookie jar, other included scripts, and additional first-party resources. Including third-party scripts in the main frame is common in practice, either intentionally (e.g., for analytics or advertising) [41, 69] or unintentionally [57] via transitive inclusion chains [35, 36]. With the phase-out of third-party cookies, trackers have increasingly turned to *first-party cookies*, allowing embedded third-party scripts to use the first-party cookie jar to store and exfiltrate user and device identifiers. To gain this level of access, trackers’ scripts must necessarily execute in the main frame, ensuring it inherits the full privileges of the embedding origin.

The adversary in our threat model is a third-party script embedded in the main frame, either directly by the developer or indirectly through a transitive inclusion chain. We explicitly exclude scripts

isolated in iframes or sandboxed contexts, since SOP already constrains them from accessing first-party resources. From the set of main-frame resources, our analysis focuses specifically on cookies accessible via JavaScript; namely, non-HttpOnly HTTP cookies and those set directly through JavaScript. We refer to the interactions of this third-party script with the main frame’s resources as *cross-domain* interactions. As we stated in Section 2.1, we use cross-domain to denote interactions between scripts from different eTLD+1 domains executing in the same main-frame origin, and reserve cross-origin for the stricter SOP definition. Within this model, an adversarial script may: (1) **Read cookies** set by other cross-domain scripts, exposing identifiers such as session tokens, location data, or browsing history. If session cookies are exposed, the adversary can potentially hijack authenticated sessions and impersonate the user [3, 30]. (2) **Overwrite or delete cookies** set by other cross-domain scripts, altering application logic, user state, or enabling persistence cookie-based tracking; (3) **exfiltrate cookie contents** by embedding identifiers or values in outbound requests to third-party domains. Together, these capabilities demonstrate how cross-domain inclusions in the main frame undermine cookie integrity and pose privacy risks.

Some third-party scripts, such as analytics libraries or personalization services, are intentionally included by developers because functionality such as fine-grained user interaction tracking cannot be achieved from within an iframe. However, the fact that certain inclusions are benign does not mean that all included scripts should be implicitly trusted with first-party privileges. This uniform level of trust is risky even when individual scripts appear benign: an analytics library may inadvertently expose authentication cookies, and a tag manager can transitively load additional malicious scripts or tracking scripts that repurpose first-party identifiers for cross-site tracking. By treating all third-party inclusions as fully trusted, current browser security models overlook these risks.

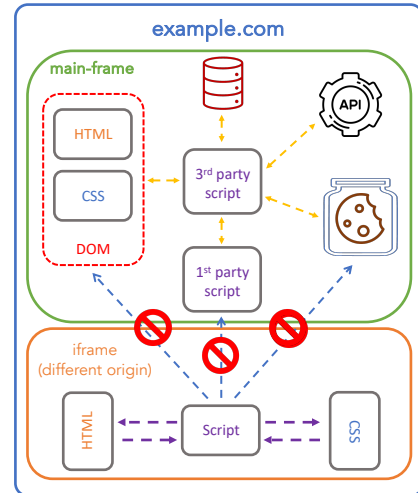


Figure 1: Access of a script included in the main frame of a website to shared resources.

4 Measurement Methodology

To quantify the prevalence of cross-domain cookie interactions in the wild, we design and deploy a custom Chrome extension that performs fine-grained instrumentation of both script- and HTTP-level cookie interactions, as well as outbound network requests. Our methodology consists of three key components: (1) a browser extension that performs dynamic instrumentation, (2) a data collection pipeline that crawls popular websites and simulates user interaction, and (3) an analysis framework that identifies cross-domain cookie manipulations and exfiltrations, and the responsible third-party scripts.

4.1 Extension Instrumentation

Our Chrome extension enables visibility into both synchronous and asynchronous cookie APIs, server-set cookies, and network requests—all within the main frame of visited pages.

Intercepting document.cookie API We override the native `document.cookie` property using `Object.defineProperty`, wrapping its *getter* and *setter* to intercept all read and write operations. For each access, we log the cookie string, the calling script’s URL (derived from the stack trace), and the base domain of the visited site. These logs are sent to the extension’s background service via `postMessage`.

Intercepting CookieStore API We override the *get*, *getAll*, *set*, and *delete* methods of the `CookieStore` API to monitor asynchronous cookie operations. Each access is logged with the cookie name, value, method, and initiating script URL.

Capturing HTTP Set-Cookie Headers To capture server-set cookies, we use Chrome’s `webRequest.onHeadersReceived` API to inspect HTTP response headers. We extract non-`HttpOnly` *Set-Cookie* values and log them along with the response URL and the initiator of the request. By comparing the initiator and response domains, we determine whether the cookie was set in a first- or third-party context.

Network Request Attribution To attribute outgoing data transmissions to third-party resources, we attach to each tab using the Chrome Debugger Protocol and listen for `Network.requestWillBeSent` events. When a request is initiated, we analyze the stack trace to identify the exact script responsible. This allows us to connect network activity (e.g., exfiltration) to prior cookie accesses, enabling end-to-end attribution.

4.2 Data Collection

We crawl the landing pages of the top 20,000 websites from the Universal Tranco list [39]. All crawls were conducted from a university network located in California, USA. Page visits are automated using Selenium, which launches a Chrome instance preloaded with our instrumentation extension.

As the page loads, our extension records cookie reads, writes, deletions, *Set-Cookie* headers, and network requests. To simulate real user interactions on a website, we perform light interaction by automatically scrolling through the page and clicking on a randomly selected link up to three times, pausing two seconds between each step. We retain only those sites for which both cookie access logs and network request data are successfully collected. Applying this

criterion, we obtained complete data for 14,917 websites used in our analysis.

Our analysis focuses exclusively on cross-domain interactions that occur in the *main frame*. According to the Same-Origin Policy (SOP), scripts executing inside iframes are restricted from accessing the main frame’s DOM or cookie jar unless they share the same origin. As such, we restrict our analysis to scripts running in the main frame, where these protections do not apply.

By combining cookie access monitoring with network attribution, our methodology enables a detailed analysis of how third-party scripts manipulate or exfiltrate cookies in real-world websites.

4.3 Identifying Advertising/Tracking Scripts

Throughout this paper, one of our primary objectives is to identify tracking and advertising scripts that are included in the main frame. To this end, we leverage the `adblockparser` [2] tool that is designed for matching URLs against filter rules. We combine nine popular crowd-sourced filter lists [5, 9, 23–26, 54, 64, 72], including EasyList and EasyPrivacy, that are specifically curated to target URLs associated with advertising and tracking. Note that we classify each occurrence of a third-party script URL within a particular website context as advertising/tracking or not.

4.4 Analysis Framework

We analyze the collected logs to detect and attribute cross-domain cookie access, manipulation, and exfiltration by third-party scripts. The results of this analysis are presented in Section 5. Before presenting our findings, we describe the methodology used to detect cross-domain access and exfiltration events.

Cross-domain Access Detection. We label a cookie access as *cross-domain* if the domain of the accessing script differs from the domain of the script that originally set the cookie. To identify such cases, we perform the following steps:

- (1) Identify all cookies set via script, along with the domain of the responsible script.
- (2) Track subsequent reads or modifications of these cookies by other scripts.
- (3) Determine whether those accesses resulted in manipulation (i.e., overwriting or deletion) or exfiltration.
- (4) Annotate the inclusion path of each accessing script—whether it was directly embedded by the first-party site or loaded indirectly via third-party chains.

Detecting Exfiltration. To detect exfiltration of cookie identifiers, we analyze the actual outbound network requests initiated by third-party scripts executing in the main frame, rather than relying on static URL strings.

Our identifier detection pipeline begins by extracting substrings from cookie values using the format `(name1 =)value1|...|(nameN =)valueN`, where `|` represents a delimiter (non-alphanumeric characters). We split each value on such delimiters and keep strings that are at least eight characters long, which we treat as candidate identifiers. For each candidate identifier, we compute three encoded forms: (1) Base64 encoding, (2) MD5 hash, and (3) SHA1 hash. We apply a similar algorithm to extract potential identifiers from the query strings of all outbound URLs initiated by third-party scripts in

the main frame. Exfiltration is confirmed when an identifier derived from a cookie value appears in a request to a different domain.

5 Cross-domain First-Party Cookie Access by Third-Party Scripts

In this section, we present the results of our measurement study on cross-domain cookie manipulation and exfiltration, by third-party scripts embedded in the main frame. We also investigate the predominant scripts responsible for cross-domain actions. Before that, we report general statistics about script-based cookie access across our dataset to provide context for the cross-domain behaviors analyzed in the following subsections.

5.1 Prevalence of Third-Party Scripts

Third-party scripts are deeply embedded in the web ecosystem. We find that 93.3% of the 14,917 successfully crawled websites in our dataset include at least one third-party script in their main frame, with an average of 19 distinct third-party scripts per site. Notably, 70% of these scripts are affiliated with advertising or tracking services.

This level of integration has significant implications: on average, third-party scripts set 15 cookies per site, compared to just four set by first-party scripts.

5.2 Usage of Cookie APIs in the Wild

We measure the usage of both available approaches to set and get script cookies: 1) `document.cookie`, and 2) `cookieStore`.

document.cookie: We observe that `document.cookie` is invoked on 96.3% of sites. Across our dataset, we identified 81,918 unique cookie pairs, each defined as a tuple of (*cookie_name*, *domain of the script that set the cookie*)². These cookies were set by 92,235 scripts originating from 11,035 distinct domains.

cookieStore API: In contrast, the newer `cookieStore` API [16] is used far less frequently, appearing on only 2.8% of sites. We identify 411 unique cookie pairs created via `cookieStore.set()` by 428 scripts across 361 domains. Despite this diversity, only 13 unique cookie names were used. Nearly 90% of these are limited to just two cookie names: `_awl` and `keep_alive`. All instances of `keep_alive` are attributed to the Shopify performance SDK (`shopify-perf-kit-1.6.X.min.js`) hosted on `cdn.shopifycloud.com`, suggesting consistent deployment across Shopify-powered sites [61]. Similarly, `_awl` cookies follow a structured format encoding a counter, timestamp, and session ID (e.g., `_awl=count.timestamp.session_id`). Although undocumented, manual inspection of setting scripts reveals shared logic: the use of a JavaScript object `u` with fields like `u.u = "admiral"`, and references to endpoints on `getadmiral.com`, indicating the SDK is part of Admiral’s tracking infrastructure.

5.3 Prevalence of Cross-domain Cookie Interactions

While prior work has extensively examined cookie exfiltration, less attention has been paid to other forms of manipulation such as overwriting and deletion. We now systematically quantify three

²For example, a cookie named `_ga` set by `google-analytics.com/analytics.js` is represented as (`_ga`, `google-analytics.com`) and is treated as a distinct cookie throughout our analysis.

categories of cross-domain actions: **exfiltration**, **overwriting**, and **deletion**. We begin by quantifying the prevalence of *cross-domain* cookie interactions for cookies set via the `document.cookie` property and `cookieStore` API.

As shown in Table 1, cross-domain interactions to cookies set through `document.cookie` is both widespread and varied in nature. In contrast, such access to cookies set via the `cookieStore` API is significantly less common.

cookie Type	Action	% of Websites	% of Cookies (No.)
document.cookie	exfiltration	55.7	5.9 (4,825)
	overwriting	31.5	2.7 (2,212)
	deleting	6.3	1.8 (1,475)
cookieStore	exfiltration	0.7	16.3 (62)
	overwriting	0	0
	deleting	0	0

Table 1: Prevalence of cross-domain cookie actions across websites and affected cookies.

Cross-domain exfiltration. Cross-domain cookie exfiltration is the most prevalent forms of cookie misuse. We find that about 55.7% of websites include at least one script that exfiltrates a cookie it did not set originally. These behaviors impact 5.9% (4,825) of the 82,000 unique cookie pairs in our dataset, highlighting the scale at which confidentiality is compromised in the main frame. For cookies set via the `cookieStore` API, cross-domain exfiltration is significantly less common. Only 0.7% of websites include a script that exfiltrates a `cookieStore`-set cookie it did not set originally. These scripts affect 16.3% (62) of the 481 unique `cookieStore` cookie pairs observed in our dataset.

Cross-domain manipulation. Cross-domain manipulation actions (including cookie overwriting or deleting) are less common but still pose meaningful integrity risks. We find that approximately 32% of websites include at least one script that overwrites a cookie it did not originally set, while 6.3% include a script that deletes a cookie it did not create. Overall, 2.72% of all unique cookie pairs are overwritten by non-owning scripts, and 1.8% are explicitly deleted. For cookies set via the `cookieStore` API, we did not observe any cross-domain overwriting or deletion.

Together, these findings underscore the fact that cookie access in the main frame lacks effective isolation and is routinely violated. In the following subsections, we examine notable instances of these cross-domain actions and attribute them to the responsible script origins. Due to its negligible usage, the `cookieStore` API is excluded from further analysis.

5.4 Cross-domain Cookie Exfiltration

While authorized exfiltration by scripts from the same origin is a widespread and expected practice—for example, for analytics, session management, or personalization—cross-domain exfiltration lacks visibility and control, raising serious privacy concerns.

There are currently no built-in browser controls to restrict what data a script exfiltrates once it has access to the cookie jar. As a result, cookies containing user identifiers, authentication tokens,

and user browsing data are routinely extracted and sent to unrelated third-party domains [22, 27, 33].

Most Commonly Exfiltrated Cookies. Table 2 lists the top 20 unique cookies most frequently exfiltrated by cross-domain scripts. To highlight real-world privacy risks and consolidate domains belonging to the same entity, we map the URLs of cookie-exfiltrating scripts and their destination endpoints to their respective entities using DuckDuckGo’s Tracker Radar dataset [20]. We find that the `_ga` cookie—created by scripts embedded from `googletagmanager.com`—is the most frequently exfiltrated cookie. Scripts from 1,191 distinct entities (excluding Google) accessed and transmitted this cookie to 708 unique recipient entities. Other frequently targeted cookies include `_gid`, `_gcl_au`, and `OptanonConsent`, often associated with major analytics platforms. Microsoft, Yandex, and Pinterest appear as top exfiltrators, while HubSpot, Amazon, and Microsoft are the most frequent exfiltration destinations.

One notable exception in Table 2 is the `us_privacy` cookie, which encodes the IAB U.S. Privacy (CCPA) signal. Unlike tracking identifiers, this cookie is *intended* to be read and exfiltrated by third parties so that downstream ad tech can honor a user’s “Do Not Sell” preference.³ We therefore flag `us_privacy` as a *consent signal* rather than a tracking identifier. Although prior work finds that their implementation is not always compliant [7, 34].

Case study: Targeted cookie parsing and exfiltration. We analyze a concrete example of targeted exfiltration on `optimonk.com`, where a third-party script loaded from `licdn.com/li_lms-analytics/insight.min.js` (LinkedIn) accesses and processes the `_ga` cookie created by `googletagmanager.com`.

The `_ga` cookie on `optimonk.com` holds the value: `GA1.1.444332364.174683882`. This format is consistent with Google Analytics, where the middle segment (444332364) represents a pseudonymous identifier. The script extracts this segment, encodes it in Base64 (`NDQ0MzMyMzY=`), appends an additional encoded timestamp (`0LjE3NDY4Mzg4Mjc`), and transmits it to a LinkedIn endpoint via a GET request to:

```
https://px.ads.linkedin.com/attribution_trigger?pid=621340&
time=1746838846149&url=www.optimonk.com*_gcl_
au*MTg5MTM3M4xNzQ2ODM4ODI3*FPAU*...*_
ga*NDQ0MzMyMzY0LjE3NDY4Mzg4Mjc...
```

A closer inspection of the URL reveals the presence of several additional cookie-derived identifiers, including `_ga`, `_fplc`, `gcl_au`, and `fpau`, suggesting general tracking intent. Notably, although over 35 cookies were present during the visit, the script selectively parsed and exfiltrated only a subset—suggesting that cookie exfiltration here is purposeful. This undermines the assertion that identifier exfiltration here is merely a byproduct of bulk cookie access.

Case Study: Cross-company identifier exfiltration. We also identify a case of cross-company identifier exfiltration. We observe a case of identifier sharing between Facebook and Criteo facilitated through a third-party script on the website `goosecreekcandle.com`. A script hosted by Osano, Inc. [51], a consent management provider,

accesses a Facebook cookie (`_fbp`) and transmits it to Criteo SA’s infrastructure. Specifically, `osano.com/1vX3GkPazR/.../osano.js` accesses the `_fbp` cookie set by `facebook.net: fb.0.1746746266109.868308499`.

The script extracts two core components: (1) 1746746266109, a timestamp; and (2) 868308499845957651, a Facebook-assigned browser identifier. It then includes both values in a request to:

```
https://sslwidget.criteo.com/event?...&sc=%7B%22fbp%22:
%22fb.1.1746746266109.868308499845957651%22...
```

This identifier transfer occurs within the context of a partnership between Meta and Criteo [17, 18]. Given this partnership, we surmise that the data exchange is likely intentional. Criteo may receive the `_fbp` identifier to help align its own user profiles with Meta’s for improved cross-platform attribution and conversion tracking. While this identifier sharing is between two partners, its execution still raises important privacy considerations. Users are typically unaware that consent interfaces or third-party privacy scripts (like Osano) may enable identifier syncing between advertisers/trackers. Moreover, the fact that a consent management tool also facilitates data transmission underscores the blurred lines between privacy-enhancing and tracking technologies.

Top exfiltrator Domains Figure 2 presents the top 20 script domains engaged in cross-domain cookie exfiltration by the number of unique cookies they leaked.

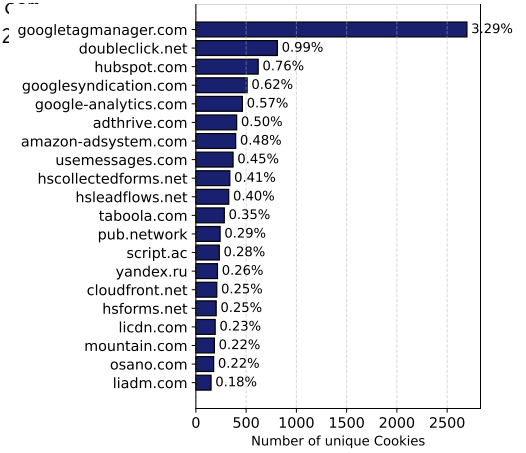


Figure 2: Top 20 script-hosting domains involved in cross-domain cookie exfiltration, ranked by the number of unique cookies exfiltrated to third-party destinations.

`google-analytics.com` is the top exfiltrator, accounting for exfiltrating 3.3% of the 82,000 cookies in our dataset. This domain is primarily associated with analytics services aiming to analyze user behavior by collecting and exfiltrating information. Several advertising exchanges platforms such as `doubleclick.net`, `amazon-adsystem.com`, and `pubmatic.com`, are also among the top exfiltrators. Their presence is largely explained by how Real-Time Bidding (RTB) systems operate[50]. During RTB auctions, bid requests are sent to multiple potential bidders and may include user identifiers and cookies—regardless of the script that originally created them.

³In practice, consent strings are commonly stored in a cookie (often named `us_privacy` or `usprivacy`) and read by third parties implementing the IAB signal.

Cookie Name	Owner Domain	# Exfiltrator Entities	# Destination Entities	Top 3 Exfiltrator Entities	Top 3 Destination Entities
_ga	googletagmanager.com	1191	664	Microsoft, Yandex, Pinterest	HubSpot, Microsoft, Amazon
_gid	google-analytics.com	718	576	Microsoft, Pinterest, Yandex	HubSpot, Microsoft, Yandex
_ga	google-analytics.com	482	456	Microsoft, Yandex, Pinterest	HubSpot, Microsoft, Yandex
_gcl_au	googletagmanager.com	586	416	Microsoft, Pinterest, Adobe	HubSpot, Microsoft, Yandex
_gcl_au	googletagmanager.com	367	285	Microsoft, Adobe, HubSpot	HubSpot, Microsoft, Amazon
i	openx.net	157	226	Google, Mediavine, AdThrive	Amazon, LiveIntent, Yandex
pd	openx.net	138	217	Google, Mediavine, AdThrive	Amazon, LiveIntent, Yandex
SPugT	pubmatic.com	87	181	Google, Taboola, AdThrive	Microsoft, Amazon, X
PugT	pubmatic.com	63	167	Google, script.ac, Amazon	Amazon, Yandex, LiveIntent
__utma	google-analytics.com	54	159	Yandex, Microsoft, Adobe	HubSpot, Microsoft, Yandex
_fbp	facebook.net	73	154	Criteo, Google, Salesforce.com	Criteo, whitesaas.com, c99.ai
__utmb	google-analytics.com	44	152	Yandex, Microsoft, envybox.io	HubSpot, Microsoft, Yandex
__utmm	google-analytics.com	44	152	Yandex, Microsoft, envybox.io	HubSpot, Microsoft, Yandex
_mkto_trk	marketo.net	44	127	c99.ai, ensembleiq.com, SSL	Adobe, c99.ai, insent.ai
_ym_d	yandex.ru	86	126	Google, mango-office.ru, envybox.io	Yandex, Criteo, whitesaas.com
lotame_domain_check	crwdcntrl.net	31	114	Amazon, hadronid.net, Google	Amazon, Criteo, LiveIntent
us_privacy	ketchjs.com	17	112	Google, tradehouse.media, ex.co	33Across, Anview, LiveIntent
_yjsu_yjad	yimg.jp	30	109	Google, Taboola, Microsoft	Microsoft, Criteo, Adobe
gaconnector_GA_Client_ID	gaconnector.com	4	106	Google, Microsoft, HubSpot	Microsoft, HubSpot, Airbnb
gaconnector_GA_Session_ID	gaconnector.com	4	106	Google, Microsoft, HubSpot	Microsoft, HubSpot, Airbnb
sc_is_visitor_unique	statcounter.com	18	103	Google, Mediavine, Intergr	LiveIntent, Magnite, ShareThis

Table 2: Top 20 cookie names and their creator domains that have been exfiltrated by cross-domain scripts across 20,000 websites. The table also reports the number of cross-domain exfiltrator entities responsible for the exfiltration and the number of destination entities that received these cookies. Entries are sorted by the number of destination entities.

With 14.3% of top Tranco sites supporting RTB [52], widespread cookie leakage to ad networks is expected which raises concerns about cross-domain access and misuse of user data by cross-domain scripts.

5.5 Cross-domain Cookie Overwriting and Deletion

In this section, we focus on cross-domain cookie manipulation including overwriting and deleting. Invoking the document.cookie API returns the entire cookie jar, regardless of whether the caller script requires all cookies or not. However, to overwrite or delete a specific cookie value, a script needs to know the corresponding cookie name (key). As shown in Table 1, 2,121 and 1,475 of unique cookies are overwritten and deleted by scripts that did not originally set them. To better understand the nature of cookie overwriting, we analyze the attributes modified during such events. Specifically, we examine changes to the cookie’s value, expires, domain, and path fields. Among all overwrite events involving cookies identified as victims of third-party modification, 85.3% involved a change in the value, 69.4% updated the expires timestamp, 6.0% altered the domain, and 1.2% changed the path. These results suggest that overwriting is primarily used to manipulate the content and lifespan of cookies, potentially to repurpose user identifiers or extend tracking durations beyond the original intent.

Most Frequently Modified Cookies. Table 5 in Appendix A.1 presents the top 10 unique cookie pairs that are manipulated—either overwritten or deleted—by cross-domain scripts. The _fbp cookie, set by facebook.net, is the most frequently overwritten, targeted by scripts from 132 entities, and also deleted by 18 distinct entities. Other commonly overwritten cookies include OptanonConsent, _ga, and cto_bundle, while frequently deleted ones include tracking-related cookies such as _uetvid, _ga, and _gid. Scripts from

Function Software, Google, and Gatehouse Media dominate cross-domain overwriting actions, whereas cdn-cookieyes.com, cookie-script.com, and Tealium are leading in cross-domain deletion activities.

Case Study: Intention behind manipulations Understanding the intent behind cross-domain cookie manipulations is inherently challenging due to the lack of documentation or explicit disclosures. Nevertheless, our analysis reveals several recurring patterns and concrete cases that help illuminate the purposes behind these actions.

Collision. One possible explanation for this behavior is the use of generic cookie names, such as *cookie_test* or *user_id*, which are more likely to be targeted. For instance, our analysis identifies at least eight distinct *cookie_test* cookies—each set by a different script—yet overwritten or deleted by more than 70 unique scripts. Therefore, these modifications or deletions may result from unintentional name collisions.

Collusion or Competition. We observe instances where cookies with non-trivial, hard-to-guess names are overwritten by cross-domain scripts. For instance, the value of *cto_bundle* cookie initially created by Criteo is over-written by a script associated with Pubmatic. It changes from a string in hash format with a length of 194 characters to an entirely different value in hash format with a length of 258 characters. This behavior suggests a deliberate overwrite rather than an accidental collision. One possible explanation is technical coordination between ad tech companies—such as ID synchronization or shared identifier infrastructure. However, such behavior may also signal competitive dynamics, where one entity seeks to override or disrupt the identifiers of another to gain control over user tracking or ad targeting capabilities.

Privacy Compliance. Site owners or tag managers may configure services to delete cookies from certain third parties (e.g., Bing) to enforce compliance with regional privacy laws (e.g., GDPR/CCPA) or to avoid unnecessary data retention [14].

In Table 5 we can see `cookie-script.com` and `cdn-cookieyes.com` among top entities engaged in cross-domain deleting. Both `cookie-script.com` and `cookieyes.com` are associated with cookie consent management platforms designed to help websites comply with data privacy regulations like GDPR and CCPA. Scripts from these services delete tracking cookies such as `_fbp` and `_uetvid` to enforce user privacy preferences and comply with regulations like GDPR and CCPA. If a user declines consent for marketing cookies, these scripts remove pre-existing tracking identifiers to prevent unauthorized data collection. In some cases, scripts may also delete and overwrite these cookies to reset tracking sessions or reassert control over user identifiers.

Top manipulator. Figure 8 in Appendix B presents the top 20 script-hosting domains involved in cross-domain cookie overwriting and deletion. The y-axis lists the top domains, while the x-axis indicates the number of unique first-party cookies manipulated by scripts from each domain. `googletagmanager.com` ranks highest in overwriting activity, modifying 0.5% of all cookies in our dataset (386 out of 82,000). On the other hand, `prettylittlething.com` leads in cookie deletion, removing 0.31% of cookies (252 out of 82,000). Most domains associated with overwriting are tied to advertising and tracking services, whereas those responsible for deletion are primarily linked to consent management platforms.

5.6 Where Do These Scripts Come From?

We classify cookie-accessing scripts by inclusion path: 93.3% of websites include third-party scripts in the main frame. Indirect inclusions outnumber direct inclusions by a factor of 2.5×. 33% of indirect third-party scripts are associated with advertising or tracking.

Websites often include tools like Google Tag Manager or advertising SDKs directly, which then dynamically inject additional scripts—including those that perform cross-domain cookie operations. This inclusion indirection makes it hard for website owners to audit or control access to the cookie jar.

5.7 Implications

Our analysis reveals that first-party cookie integrity is fundamentally broken in the presence of third-party scripts in the main frame: Scripts frequently access, modify, and exfiltrate cookies they did not create. Such actions occur on nearly half of all websites. Indirect inclusion chains obscure the origin of cross-domain behavior. Even potentially sensitive domains (e.g., healthcare) are not immune. Emerging practices like server-side tracking bypass client-side defenses, including our own COOKIEGUARD (§6), by proxying exfiltration through seemingly first-party endpoints [4, 28, 71]. These findings call for stronger ownership semantics for browser cookies—where access control is tied to the script that created the cookie, not merely the domain from which it is served.

6 COOKIEGUARD: Design & Implementation

Our measurement and analysis in Section 5 reveals that first-party cookies are vulnerable to cross-domain manipulation and exfiltration by third-party scripts embedded in the main frame. To mitigate

this issue, we introduce COOKIEGUARD, a browser extension that enforces isolation of first-party cookies on a per-script-origin basis.

6.1 Design

COOKIEGUARD enforces access control over first-party cookies by maintaining a metadata store that logs each cookie’s name and the ETLD+1 of the script or server that created it. This metadata is updated during cookie creation events, both via JavaScript (`document.cookie` and `cookieStore.set()`) and HTTP Set-Cookie headers.

When a script attempts to read from `document.cookie`, COOKIEGUARD identifies the script’s ETLD+1 and filters the returned cookies, exposing only those that were originally created by scripts from the same domain. This policy effectively prevents third-party scripts embedded in the main frame from accessing or interfering with cookies set by other parties.

On some websites, essential functionality like user session management, preferences, or shopping carts are handled by third-party scripts embedded as first party. Blocking cookies set by these scripts could disrupt website functionality. To prevent such disruptions, in COOKIEGUARD’s implementation, we grant full access control to the website owner to get access to all first-party cookies. Consequently, if a script from the same domain as the visiting website invokes `document.cookie` or `cookieStore.getAll()`, it retrieves all first-party cookies from the cookie jar, regardless of the domain that set them.

Inline scripts present an attribution challenge, as their origin cannot be reliably determined. To address this, COOKIEGUARD supports two modes of handling inline scripts. In strict mode, COOKIEGUARD adopts a safe-by-default strategy by treating such scripts as untrusted and denying them access to any cookies. This conservative stance aligns with privacy defenses that restrict ambiguous behaviors by default to minimize the attack surface and increase the cost of circumvention. In relaxed mode, by contrast, COOKIEGUARD treats inline scripts as first-party scripts. We will not use the relaxed mode in our evaluations and it is included only to illustrate alternative design choices.

Figure 3 shows an overview of COOKIEGUARD. ① A server at `site.com` sets cookie “c0” via an HTTP Set-Cookie header. Since the response matches the visited domain, the cookie is marked first-party. The domain of the URL in the original cookie jar and creator domain of this cookie in COOKIEGUARD’s database are both set to `site.com`. ② A script from `site.com` sets cookie “c1” using `document.cookie`. Like “c0”, it is considered a first-party cookie with `site.com` as its creator in both the browser and COOKIEGUARD’s records. ③ A third-party script from `ad.com`, embedded in the main frame of `site.com`, sets cookie “c2”. While the browser treats it as first-party (domain: `site.com`), COOKIEGUARD records `ad.com` as its creator. ④ When the script from `ad.com` calls `document.cookie`, the browser returns all first-party cookies (“c0”, “c1”, and “c2”). COOKIEGUARD filters out “c0” and “c1” since their creators differ, and returns only “c2”. ⑤ A script from `site.com` calls `document.cookie`. Under the full-access policy (Section 6), COOKIEGUARD returns all first-party cookies, since the script’s domain matches the visited domain.

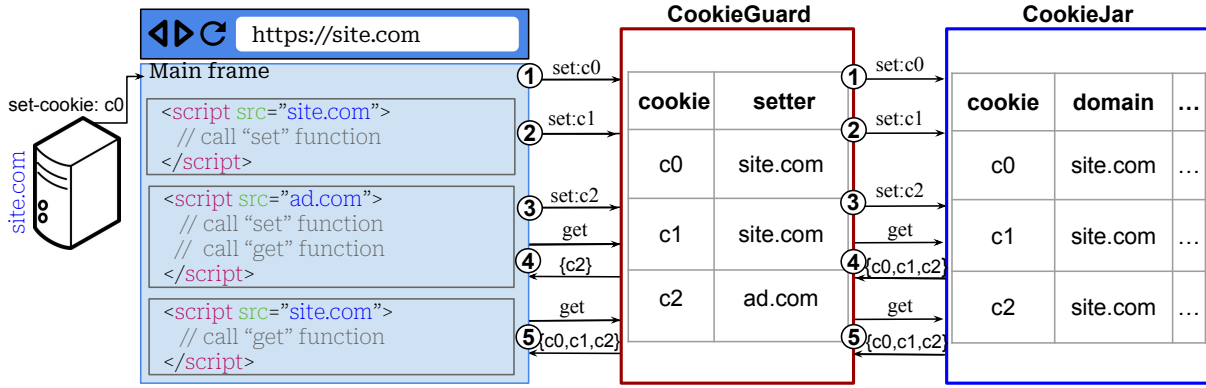


Figure 3: COOKIEGUARD overview

In the following section we explain the components of COOKIEGUARD, and we detail its implementation.

6.2 Implementation of COOKIEGUARD

We implement COOKIEGUARD as a browser extension, as illustrated in Figure 4. The extension has three main components:

contentScript.js: Injects `cookieGuard.js` into the main frame and mediates communication between `cookieGuard.js` and `background.js`. It relays messages to ensure synchronization between DOM-level script activity and background-level cookie tracking.

cookieGuard.js: Intercepts cookie access by wrapping both the legacy `document.cookie` and the modern `cookieStore` API. It implements a wrapper around the `document.cookie` and `cookieStore`'s `get()` and `set` functions, enabling the interception of cookie jar interactions. For cookie writes, `cookieGuard.js` captures the cookie name and the domain of the calling script, inferred by analyzing the JavaScript stack trace to locate the last external script URL. This metadata is sent to `background.js` to update the extension's internal dataset. For reads, `cookieGuard.js` queries `background.js` for the latest dataset and filters the cookies based on the ETLD+1 of the caller. If the script's domain matches the top-level site, all first-party cookies are returned (preserving default browser behavior). Otherwise, only cookies set by the same third-party domain are exposed.

background.js: Monitors HTTP responses that contain Set-Cookie headers, logging details of non-HTTPOnly and first-party cookies by name and setter domain in the extension's storage. It handles all updates to the dataset initiated by 'set' requests from both HTTP headers and `cookieGuard.js`. Furthermore, when the 'get' function is invoked in `cookieGuard.js`, `background.js` receives a message through `contentScript.js` to provide a current copy of the dataset for accurate cookie filtering and response.

7 COOKIEGUARD Evaluation

We assess the efficacy of COOKIEGUARD, our cookie protection tool, through a multi-faceted evaluation framework, focusing on accuracy, website breakage, and runtime performance.

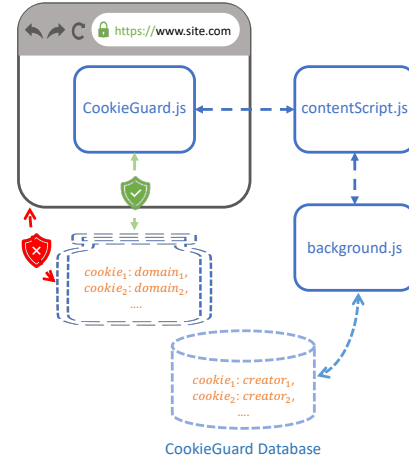


Figure 4: COOKIEGUARD's components

7.1 Access Control

We assess the effectiveness of COOKIEGUARD in preventing cross-domain actions such as the retrieval, overwriting, deletion, and exfiltration of first-party cookies. Figure 5 illustrates the comparison between the percentage of sites engaging in these actions on first-party cookies with the COOKIEGUARD extension enabled versus a regular browser without the extension. Notably, COOKIEGUARD significantly reduces cross-domain actions on first-party cookies: overwriting by 82.2%, deletion by 86.2%, and exfiltration by 83.2%. As discussed in Section 6.1, to minimize website breakage, COOKIEGUARD grants website owners full control over access to first-party cookies by their own scripts. As a result, cross-domain actions on first-party cookies may still occur under COOKIEGUARD, and the percentage of affected sites is not zero.

7.2 Website Breakage

We perform a manual assessment of COOKIEGUARD to determine its impact on website functionality when the COOKIEGUARD extension is enabled. A random sample of 100 websites from the Tranco top 10k list is selected and tested in two separate settings: with

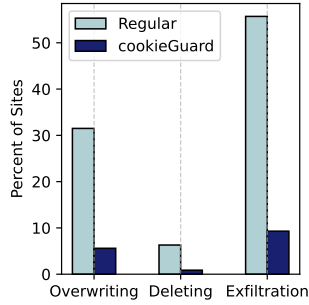


Figure 5: Evaluation of cross-domain cookie interactions: Comparing regular browser behavior with and without the COOKIEGUARD extension.

and without the COOKIEGUARD extension activated. One author examines all 100 websites, while two additional evaluators are recruited to independently assess website breakage. Each of these evaluators is responsible for assessing a unique set of 50 websites from our chosen sample. At least two distinct evaluators review each website to ensure a comprehensive evaluation.

We classify breakage into four categories: navigation (moving between pages), SSO (initiating and maintaining login state), appearance (visual consistency), and other functionality (such as chats, search, and shopping cart). Breakage is labeled as either major or minor for each category: Minor breakage occurs when it is difficult but not impossible to use the functionality. Major breakage occurs when it is impossible to use the functionality on a web page.

The results are shown in Table 3. While COOKIEGUARD has no impact on navigation and appearance (0% breakage), we observe 1–11% minor and major breakage on SSO and other functionality. More specifically, on 11% of the websites, we are not able to sign in using SSO, because the sign-in process is managed by third parties. When the user logs into SSO provider’s system such as Google account, the system authenticates the user and upon successful authentication, the SSO provider sets a session cookie in the user’s browser. This cookie is then used to remember the user’s authentication status. The SSO that leads to web page breakage stems from the dependency on third-party cookies for session management and login process. On some websites, multiple third-party scripts are in charge of session management. For example, on zoom.us two scripts, i.e. microsoft.com and live.com are in charge of SSO. Since COOKIEGUARD eliminates all cross-domain manipulations, in such cases breakage happens. An example of minor breakage occurs on cnn.com, where a user can sign in, but refreshing the page logs them out. This behavior results from a limitation in our browser extension implementation, which interferes with how certain session cookies are handled across page reloads. In terms of functionality, on 3% of the websites, there is at least one advertisement served by a third-party script, which is not shown due to our extension; and in another 3% of the websites, COOKIEGUARD leads to major breakage in functionality. For example, on facebook.com, Facebook Messenger, an instant messenger service owned by Facebook, is managed by cookies served from fbcdn.net. Although fbcdn.net is a CDN belonging to Facebook, since their domains are different, it is considered as third-party on facebook.com. Therefore, its access to

the cookie jar is limited by COOKIEGUARD which leads to not working properly. To address compatibility issues on facebook.com and other websites with similar behaviors, we implement a whitelist feature in COOKIEGUARD that groups all domains belonging to the same entity, using information from DuckDuckGo’s entities list [20]. This refinement reduces website breakage to 3%.

	Navigation	SSO	Appearance	Functionality
Minor	0%	1%	0%	3%
Major	0%	11%	0%	3%

Table 3: Qualitative manual analysis for 100 websites using COOKIEGUARD, showing % of **No, **Minor**, and **Major** breakages in navigation, SSO, appearance, and functionality.**

7.3 Runtime Performance

We evaluate performance overhead on the top 10K websites by visiting each URL with and without COOKIEGUARD and collecting standard page-load metrics via Selenium. To ensure a fair comparison, we keep only URLs observed in *both* conditions and discard invalid or non-positive measurements. After pairing and cleaning, our analysis covers 8,171 valid websites.

These metrics include the timing of key page load events. Specifically, we measure dom_content_loaded, dom_interactive, and load_event_time using Selenium. dom_content_loaded time indicates when the HTML document has been completely loaded and parsed, without waiting for stylesheets, images, and subframes to finish loading. dom_interactive time indicates the duration until the DOM is fully prepared for user interaction. load_event_time indicates the total time to completely load all resources, such as images and CSS, signifying the full usability of a web page.

Table 3 summarizes the results. A standard benchmark [62] for user-perceived web page performance measures the timeline of key events marking various stages in the browser’s page load and rendering process. The management of cookies by COOKIEGUARD adds an average overhead of 0.3 seconds to website performance. While this may impact website load times negligibly, it is a reasonable trade-off for the significant improvements in security and privacy controls that COOKIEGUARD provides. By managing cookie access more strictly, COOKIEGUARD prevents unauthorized data access and enhances user trust.

Metrics	Normal (Mean, Median)	COOKIEGUARD (Mean, Median)
DOM Content Loaded	1659 ms, 946 ms	1896 ms, 1020 ms
DOM Interactive	1464 ms, 842 ms	1702 ms, 911 ms
Load Event	3197 ms, 2008 ms	3635 ms, 2136 ms

Table 4: Performance analysis for the COOKIEGUARD replacement

Distributional view. Means and medians alone can obscure the heavy-tailed, multiplicative nature of page-load times; a small

fraction of very slow loads disproportionately influence aggregates. Therefore, we visualize *paired* distributions for DOM Content Loaded, DOM Interactive, and Load Event Time on a logarithmic y-axis (Figure 6). The two conditions (“No Extension” and “With COOKIEGUARD”) appear as separate x-axis categories with identical styling; medians are center lines, boxes show interquartile ranges, whiskers extend to $1.5 \times \text{IQR}$, and points denote outliers. The log scale makes factor-level differences legible and reveals long right tails typical of modern pages. Across all three metrics, the *With COOKIEGUARD* boxes are slightly shifted upward, indicating a modest but systematic overhead. The tail is most pronounced for LOAD EVENT TIME, which, by definition, waits for all subresources and any window . load handlers; pages with extensive third-party stacks dominate this tail.

We also report a per-site overhead ratio (With/No) on a log axis (Figure 7) to normalize across websites with very different absolute times. The dashed line at 1.0 marks parity; medians above it indicate a typical slowdown while the spread reflects cross-site variability. Over 8,171 paired sites, median ratios are 1.108 for DOM Content Loaded, 1.111 for DOM Interactive, and 1.122 for Load Event Time. Linear (millisecond) versions of the three boxplots, along with absolute summaries, appear in the Appendix C for direct reading of raw timing differences.

8 Limitations and Future work

COOKIEGUARD’s prototype implementation highlights both the promise and challenges of ownership-based cookie isolation. In this section, we describe its limitations and point to future work that can improve its practicality and expand its applicability.

Manipulation of script source. COOKIEGUARD relies on the script source attribute to distinguish between third-party scripts from different domains in the main frame. However, these third-party scripts can attempt to manipulate the script source to circumvent COOKIEGUARD’s protection in the following ways:

- Embedded as inline scripts: rather than including scripts using script tags, the source of a third-party script can be embedded as inline script. Inline scripts are automatically considered as first-party scripts.
- Hosted as a first-party script: web developers may decide to host the script code on the same domain as the website (or a subdomain of the website).
- CNAME cloaking: CNAME cloaking [40, 63] is a technique that creates a first-party subdomain and assigns a DNS CNAME record pointing to the third-party domain of the script.

As discussed in Section 6, COOKIEGUARD adopts a safe-by-default strategy for inline scripts by treating them as untrusted and denying access to all cookies. An alternative approach would be to assess their behavior at runtime and determine trust based on observed execution patterns. Chen et al. [12] propose a signature-based mechanism that creates behavior signatures for scripts. A practical method would involve conducting a large-scale web crawl to generate signatures for various third-party scripts. If COOKIEGUARD detects that a first-party script’s signature matches that of a third-party script, it can then treat it as a third-party script. A key challenge that warrants further attention is JavaScript obfuscation [42]. For

this signature-based approach to work, the signatures should be resistant to minor changes in a script’s code.

We acknowledge that CNAME cloaking and intentional self-hosting by site owners are difficult challenges for any client-side defense. However, our focus with COOKIEGUARD is primarily on the more common and less deliberate cases where third-party scripts are introduced indirectly, often via tag managers or bundled SDKs, in ways that web developers may not be fully aware of. In these scenarios, our solution provides meaningful protection by preventing unintended cross-script access and exfiltration. When a web developer intentionally colludes with trackers (e.g., through CNAME cloaking) the problem becomes significantly harder, as attribution is intentionally obscured at the network or DNS layer. In such cases, existing DNS-based defenses against CNAME cloaking can also help COOKIEGUARD [10, 46, 75].

Toward Practical Deployment. Currently, COOKIEGUARD’s prototype implementation is available to use as a browser extension rather than a ready-to-deploy browser feature. In practice, browser vendors have historically introduced potentially disruptive privacy protections incrementally, giving developers time to adapt. For example, Safari rolled out Intelligent Tracking Prevention (ITP) in stages—starting in 2017 with limits on cross-site cookies and link-decoration controls [73], then moving to full third-party cookie blocking in March 2020 [74]. During this transition, WebKit employed temporary “grandfathering” mechanisms to preserve existing site data while ITP relearned user interactions, easing the migration for websites [31]. A similar path could be envisioned for COOKIEGUARD: initially offered as an opt-in feature or in private browsing mode, then incrementally moved toward default enforcement with advance notice to developers. Another possibility would be to expose COOKIEGUARD’s policies as user-selectable privacy settings, allowing users to balance functionality and privacy according to their preferences.

Interactive mode and scope limitations. Our automated large-scale crawl includes light user interaction to mimic realistic browsing behavior: the crawler scrolls through pages and clicks on three random links per site. However, we do not perform any authenticated logins during the crawl, and as a result, authentication cookies are not generated. Most session cookies used for authentication are correctly marked with the `HttpOnly` flag, which makes them inaccessible to JavaScript and, thus, out of scope for the class of script-based attacks we study. As described in Section 2.3, browser vendors enforce this restriction specifically to protect sensitive cookies from exfiltration via client-side scripts.

Furthermore, because our crawler does not fill forms or input personal data, we did not observe the leakage of personally identifiable information (PII) during cookie exfiltration.

Exploring script-based exfiltration of authentication tokens or PII in authenticated browsing contexts remains an important direction for future work. We plan to complement our current analysis with user-assisted or session-replay-based authenticated crawls to investigate these higher-risk scenarios.

Limitations of Source Attribution. COOKIEGUARD relies on the JavaScript stacktrace to identify the script origin responsible for cookie access. This approach is effective in most synchronous

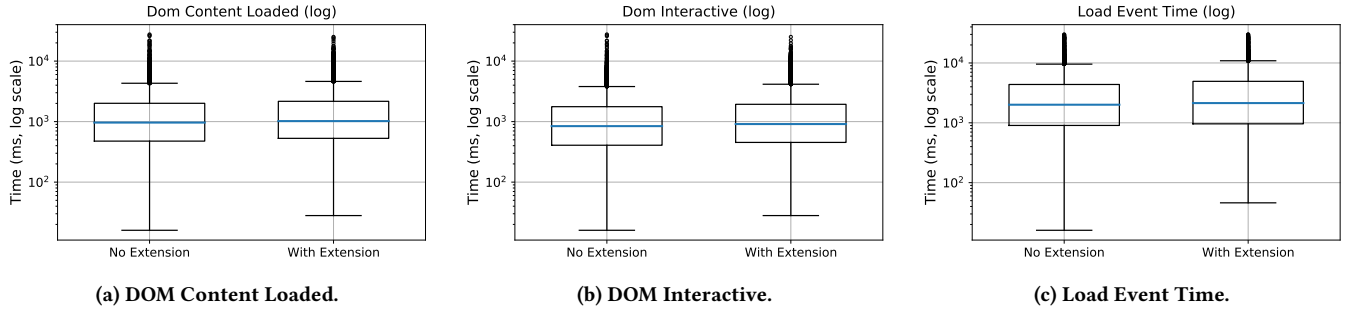


Figure 6: Paired distribution of DOM Content Loaded, DOM Interactive, and Load Event Time (With COOKIEGUARD vs. No COOKIEGUARD) with logarithmic y-axis. Blue lines show medians; boxes show interquartile range (IQR); whiskers indicate $1.5 \times \text{IQR}$; points are outliers.

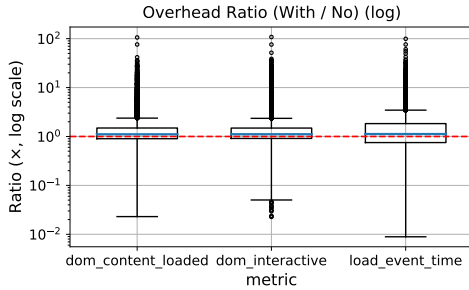


Figure 7: Overhead ratio (With COOKIEGUARD / No COOKIEGUARD) for each metric, log-scale y-axis. The dashed line at 1.0 indicates parity.

and directly-invoked contexts, including event listeners and DOM-triggered handlers. However, it may fall short in certain asynchronous scenarios—such as when cookies are accessed in callbacks following `setTimeout`, `Promise` resolutions, or event queue re-entries—where the originating script may no longer appear in the stacktrace. While recent browser APIs like `Error.prepareStackTrace` and `async stack` traces can improve coverage, some edge cases remain unresolved. We leave a more comprehensive treatment of such indirect flows to future work.

Functional Motivations for Cross-Site Cookie Access. Our manual evaluation shows that most breakage under COOKIEGUARD is due to Single Sign-On (SSO) systems, where identity providers (e.g., Google, Microsoft, Okta) access or modify first-party cookies to manage login flows. Blocking such access can disrupt session continuity.

Other legitimate use cases—such as analytics, A/B testing, or personalization—may also involve cross-domain access to first-party cookies. However, these are often indistinguishable from privacy-invasive practices like tracking or identifier syncing.

COOKIEGUARD adopts a strict isolation model by default but can be extended to support whitelisting for trusted domains when needed.

Beyond cookies: Cross-domain manipulation of other shared resources. As we discussed in Section 3, scripts included in the main frame have access to a variety of shared resources beyond

cookies such as the HTML DOM. We conducted a pilot analysis to investigate the existence and prevalence of cross-domain DOM modification. We observed that a number of cross-domain scripts run with full privileges modify, insert, or remove DOM elements that do not belong to them on 9.4% of sites. Modification can be applied to the content of HTML elements, e.g., `innerText` or `innerHTML`, the style (CSS) of the HTML element, and the attributes and classes of the HTML element, e.g., `src`. We leave a more comprehensive investigation of this issue to future work and plan to develop a targeted defense mechanism to mitigate this behavior.

9 Related Work

Third-party scripts. Prior research has studied the widespread inclusion of third-party scripts on websites and security risks associated with their use [38, 45, 47]. Lauinger et al. [38] have investigated over 133K websites finding that 37% of them include at least one script with a known vulnerability. Musch et al. [45] modify the JavaScript environment in a such a way that the accidental introduction of a Client-Side XSS vulnerability through a third party is prevented. Nikiforakis et al. [47] analyze the extensive presence of third-party scripts across over 3 million pages from the top 10,000 Alexa sites, reporting that 88.5% of popular sites incorporate at least one third-party script, not necessarily in the main frame. The study also tracks the increasing reliance on third-party scripts over time. In contrast, our research, unlike others that focus on security, explores the privacy implications of third-party scripts in the main frame. Additionally, our study specifically targets the inclusion of third-party scripts in the main frame, finding that 93.3% of websites include at least one such script directly in their main frame. Our analysis categorizes these scripts by their method of inclusion, revealing that only 10% are directly included, while a significant 90% are included indirectly across 10,000 websites. This suggests that the rise in third-party script inclusion may not directly correlate with increased trust by websites.

First-party cookies ghost-written by third-party scripts. Another line of prior work has studied cookie creating and exfiltration by third-party scripts. Sanchez et al. [60] analyzed cookie (both script and HTTP cookies) exfiltration, overwriting, and deleting across 1 million websites. They collected 66.7 million cookies from 74% (738,168) of the websites visited. Their findings reveal

that 11% of these cookies are first-party, 47% are third-party, and 42% are ghost-written. They also find that 13.4% of all cookies are exfiltrated, 0.19% are overwritten, and 0.08% are deleted by scripts or Set-Cookie headers in responses. Additionally, they report that cross-domain cookie exfiltration and collision (including overwriting and deleting) occur on 28.3% and 0.7% of the visited websites, respectively. Our work focuses exclusively on first-party scripts (created by both first-party and third-party scripts on the main frame) due to their critical role in storing users' sensitive information and session cookies. Consequently, we only consider cross-domain actions, including exfiltration and manipulation, by scripts on the main frame. We do not investigate actions caused by Set-Cookie headers in responses, as responses sent from third-party servers cannot access or manipulate first-party cookies. Since the publication of this paper, i.e. 2021, we have noted a significant shift of third-party scripts into the main frame. Notably, 92% of all first-party cookies in our dataset are ghost-written cookies. This shift has had significant consequences. The incidence of cross-domain cookie exfiltration, overwriting, and deletion has increased by factors of 1.3, 19.5, and 6.2, respectively. Additionally, the number of websites engaging in cross-domain cookie exfiltration, overwriting, and deleting has risen by 1.5 and 45.7 times, respectively. It is important to note that these figures represent a conservative estimate, as our study focuses only on first-party and ghost-written cookies manipulated or exfiltrated by scripts in the main frame, excluding other scripts and HTTP responses.

Chen et al. [11] investigated ghost-written first-party cookies set by third-party tracking scripts. Their findings indicate that 97% of the Alexa top 10K websites host at least one ghost-written cookie, with exfiltration occurring on 57.7% of these sites by both same-domain and cross-domain scripts. However, their study does not address other impacts of third-party scripts in the main frame, such as cookie manipulation. Moreover, neither the studies by [60] nor [11] offer a mechanism to protect first-party cookies from cross-domain exfiltration and manipulation. There are extensive number of tools and extensions allowing users to automatically delete cookies from specified domains [6, 15, 70]. Moreover, Munir et al. [44] proposed a machine learning-based mechanism to automatically block ghost-written first-party tracking cookies. However, none of these studies address the more fundamental problem of first-party cookie isolation, i.e., whether first-party cookies can be accessed or modified by different third-party scripts.

Shared global namespace. Another line of prior work has studied JavaScript conflicts in the shared global namespace. Ocariza et al. [48] and Zhang et al. [76] investigated how JavaScript global identifier conflicts might lead to cookie overwriting. Several studies focused on JavaScript issues, predominantly focusing on bad coding practices [32, 37, 48, 49, 53, 55, 76]. The primary objective of these studies is to identify conflicts shared global namespace, especially when they result in functional web issues, as opposed to focusing solely on shared cookie jar on the main frame.

Takeaway. In summary, our work is distinct from prior work in two main ways. First, our study focuses on first-party cookie manipulation/exfiltration by third-party scripts in the *main frame*. Second, we propose a defense mechanism called COOKIEGUARD to isolate cookies from manipulation/exfiltration by cross-domain scripts.

10 Conclusion

In this work, we conducted a large-scale measurement of cross-domain cookie access by third-party scripts in the main frame and introduced COOKIEGUARD, a browser-based system to isolate first-party cookies. COOKIEGUARD advances the state-of-the-art in two major ways. First, different from prior work on third-party JavaScript and cookie measurements, we specifically focus on third-party scripts included – either directly or indirectly – in the main frame such that they can manipulate/exfiltrate cookies in the first-party cookie jar. Second, different from prior work on blocking or partitioning third-party and/or first-party cookies, COOKIEGUARD is the first-of-its-kind first-party cookie partitioning system. COOKIEGUARD complements interventions that are being introduced in web browsers to improve the security and privacy of cookies. Going forward, the underlying problem addressed by COOKIEGUARD goes well beyond cookies. Third-party scripts included in the main frame have access to *all* first-party shared resources (e.g., HTML DOM, global namespace), not just the cookie jar. Future work can develop interventions, similar to COOKIEGUARD for the cookie jar, to introduce isolation in other shared resources.

References

- [1] [n. d.]. Firefox rolls out Total Cookie Protection by default to all users worldwide. <https://blog.mozilla.org/en/products/firefox/firefox-rolls-out-total-cookie-protection-by-default-to-all-users-worldwide/>. Accessed: 2025.
- [2] adblockparser [n. d.]. adblockparser. <https://github.com/scraperhub/adblockparser>. Accessed: 2025.
- [3] Assel Aliyeva and Manuel Egele. 2021. Oversharing is not caring: How cname cloaking can expose your session cookies. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 123–134.
- [4] Nardjes Amieur, Walter Rudametkin, Oana Goga, et al. 2024. Client-side and Server-side Tracking on Meta: Effectiveness and Accuracy. In *24th Privacy Enhancing Technologies Symposium (PETS 2024)*, Vol. 2024. 431–445.
- [5] Anti-Adblock Killer [n. d.]. Anti-Adblock Killer. <https://raw.githubusercontent.com/reek/anti-adblock-killer/master/anti-adblock-killer-filters.txt>. Accessed: 2025.
- [6] AutoDelete [n. d.]. Cookie AutoDelete. <https://raw.githubusercontent.com/detail/cookie-autodelete/fhcjgolkccmbidfdmjlifgaodjagh>. Accessed: 2025.
- [7] AbuBakar Aziz and Christo Wilson. 2024. Johnny Still Can't Opt-out: Assessing the IAB CCPA Compliance Framework. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*. <https://petsymposium.org/popets/2024/popets-2024-0120.php>
- [8] Nataliia Bielova, Arnaud Legout, Natasa Sarafijanovic-Djukic, et al. 2020. Missed by Filter Lists: Detecting Unknown Third-Party Trackers with Invisible Pixels. *Proceedings on Privacy Enhancing Technologies* (2020).
- [9] Blockzilla [n. d.]. Blockzilla. <https://raw.githubusercontent.com/annon79/Blockzilla/master/Blockzilla.txt>. Accessed: 2025.
- [10] Brave. 2020. Fighting CNAME Trickery. <https://brave.com/privacy-updates/6-cname-trickery/>. Brave blog post.
- [11] Quan Chen, Panagiotis Ilia, Michalis Polychronakis, and Alexandros Kapravelos. 2021. Cookie swap party: Abusing first-party cookies for web tracking. In *Proceedings of the Web Conference 2021*. 2117–2129.
- [12] Quan Chen, Peter Snyder, Ben Livshits, and Alexandros Kapravelos. 2021. Detecting filter list evasion with event-loop-turn granularity javascript signatures. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1715–1729.
- [13] chromePartitioning [n. d.]. Cookies Having Independent Partitioned State (CHIPS) origin trial. <https://privacysandbox.google.com/blog/chips-origin-trial>. Accessed: 2025.
- [14] cookieconsent [n. d.]. Cookie Consent Support for AMP Websites. https://community.cookiepro.com/s/article/UUID-94cf9b13-1998-d989-a5e0-d01a73a2e0a7?language=en_US. Accessed: 2025.
- [15] CookieQuickManager [n. d.]. Quick Cookie Manager. <https://addons.mozilla.org/en-US/firefox/addon/cookie-quick-manager/>. Accessed: 2025.
- [16] cookieStore [n. d.]. CookieStore. <https://developer.mozilla.org/en-US/docs/Web/API/CookieStore>. Accessed: 2025.
- [17] CriteosComplaint [n. d.]. Criteo's Complaint Forces Meta to Reopen Access for Ad Tech. <https://videoweb.com/2022/06/16/criteos-complaint-forces-meta-to-reopen-access-for-ad-tech/>. Accessed: 2025.
- [18] CriteosContractfacebook [n. d.]. How to accept Criteo's Business Manager in Facebook. <https://help.criteo.com/kb/guide/en/how-to-accept-criteos-business>

- manager-in-facebook-PfIF77hdM0/Steps/775597. Accessed: 2025.
- [19] CSP [n. d.]. Content security policy (CSP). <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>. Accessed: 2025.
 - [20] DDGTrackerRadar 2022. DuckDuckGo Tracker Radar list of entities. <https://github.com/duckduckgo/tracker-radar/tree/main/entities>.
 - [21] Nurullah Demir, Daniel Theis, Tobias Urban, and Norbert Pohlmann. 2022. Towards understanding first-party cookie tracking in the field. *arXiv preprint arXiv:2202.01498* (2022).
 - [22] Yana Dimova, Gunes Acar, Lukasz Olejnik, Wouter Joosen, and Tom Van Goethem. 2021. The CNAME of the Game: Large-scale Analysis of DNS-based Tracking Evasion. *Proceedings on Privacy Enhancing Technologies* 3 (2021), 394–412.
 - [23] EasyList [n. d.]. EasyList. <https://easylist.to/easylist/easylist.txt>. Accessed: 2025.
 - [24] EasyPrivacy [n. d.]. EasyPrivacy. <https://easylist.to/easylist/easylist.txt>. Accessed: 2025.
 - [25] fanboyannoyances [n. d.]. Fanboy Annoyances List. <https://easylist.to/easylist/fanboy-annoyance.txt>. Accessed: 2025.
 - [26] fanboysocial [n. d.]. Fanboy social List. <https://easylist.to/easylist/fanboy-social.txt>. Accessed: 2025.
 - [27] Shehroze Farooqi, Maaz Musa, Zubair Shafiq, and Fareed Zaffar. 2020. Canary-Trap: Detecting Data Misuse by Third-Party Apps on Online Social Networks. *Proceedings on Privacy Enhancing Technologies* 4 (2020), 336–354.
 - [28] Imane Fouad, Cristiana Santos, and Pierre Laperdrix. 2024. The Devil is in the Details: Detection, Measurement and Lawfulness of Server-Side Tracking on the Web. In *24th Privacy Enhancing Technologies Symposium (PETS 2024)*, Vol. 2024.
 - [29] Imane Fouad, Cristiana Santos, Arnaud Legout, and Nataliia Bielova. 2022. My Cookie is a phoenix: detection, measurement, and lawfulness of cookie respawning with browser fingerprinting. In *PETS 2022-22nd Privacy Enhancing Technologies Symposium*.
 - [30] Mohammad Ghasemisharif, Amrutha Ramesh, Stephen Checkoway, Chris Kanich, and Jason Polakis. 2018. O single {Sign-Off}, where art thou? an empirical analysis of single {Sign-On} account hijacking and session management on the web. In *27th USENIX Security Symposium (USENIX Security 18)*. 1475–1492.
 - [31] grandfathering 2020. Remove grandfathering from ResourceLoadStatistics. https://bugs.webkit.org/show_bug.cgi?id=210705.
 - [32] Brian Hackett and Shu-yu Guo. 2012. Fast and precise hybrid type inference for JavaScript. *ACM SIGPLAN Notices* 47, 6 (2012), 239–250.
 - [33] Mingjia Huo, Maxwell Bland, and Kirill Levchenko. 2022. All eyes on me: Inside third party trackers’ exfiltration of phi from healthcare providers’ online systems. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society*. 197–211.
 - [34] IAB Tech Lab. 2024. U.S. Privacy (CCPA) Technical Specifications and FAQ. <https://iabtechlab.com/standards/ccpa/>. Includes guidance on storing the U.S. Privacy string in a cookie.
 - [35] Muhammad Ikram, Rahat Masood, Gareth Tyson, Mohamed Ali Kaafar, Noha Loizon, and Roya Ensafi. 2019. The chain of implicit trust: An analysis of the web third-party resources loading. In *The World Wide Web Conference*. 2851–2857.
 - [36] Muhammad Ikram, Rahat Masood, Gareth Tyson, Mohamed Ali Kaafar, Noha Loizon, and Roya Ensafi. 2020. Measuring and analysing the chain of implicit trust: A study of third-party resources loading. *ACM Transactions on Privacy and Security (TOPS)* 23, 2 (2020), 1–27.
 - [37] Simon Holm Jensen, Anders Möller, and Peter Thiemann. 2009. Type analysis for JavaScript. In *Static Analysis: 16th International Symposium, SAS 2009, Los Angeles, CA, USA, August 9-11, 2009. Proceedings* 16. Springer, 238–255.
 - [38] Tobias Lauinger, Abdelberri Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. 2017. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. *NDSS* (2017).
 - [39] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Koczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS 2019)*. <https://doi.org/10.14722/ndss.2019.23386>
 - [40] Su-Chin Lin, Kai-Hsiang Chou, Yen Chen, Hsu-Chun Hsiao, Darion Cassel, Lujo Bauer, and Limin Jia. 2022. Investigating Advertisers’ Domain-changing Behaviors and Their Impacts on Ad-blocker Filter Lists. In *Proceedings of the ACM Web Conference 2022*. 576–587.
 - [41] metapixel [n. d.]. Meta Pixel. <https://developers.facebook.com/docs/meta-pixel/get-started/>.
 - [42] Marvin Moog, Markus Demmel, Michael Backes, and Aurore Fass. 2021. Statically Detecting JavaScript Obfuscation and Minification Techniques in the Wild. In *Dependable Systems and Networks (DSN)*.
 - [43] MozillaPartitioning [n. d.]. Introducing Total Cookie Protection in Standard Mode. <https://support.mozilla.org/en-US/kb/introducing-total-cookie-protection-standard-mode>. Accessed: 2025.
 - [44] Shaoor Munir, Sandra Siby, Umar Iqbal, Steven Englehardt, Zubair Shafiq, and Carmela Troncoso. 2023. Cookiegraph: Understanding and detecting first-party tracking cookies. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 3490–3504.
 - [45] Marius Musch, Marius Steffens, Sebastian Roth, Ben Stock, and Martin Johns. 2019. Scriptprotect: mitigating unsafe third-party javascript practices. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 391–402.
 - [46] NextDNS. 2025. CNAME Cloaking Blocklist. <https://github.com/nextdns/cname-cloaking-blocklist>. GitHub repository.
 - [47] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2012. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 736–747.
 - [48] Frolin S Ocariza, Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. 2016. A study of causes and consequences of client-side JavaScript bugs. *IEEE Transactions on Software Engineering* 43, 2 (2016), 128–144.
 - [49] Frolin S Ocariza Jr, Karthik Pattabiraman, and Ali Mesbah. 2012. AutoFLox: An automatic fault localizer for client-side JavaScript. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE, 31–40.
 - [50] Lukasz Olejnik and Claude Castelluccia. 2016. To bid or not to bid? Measuring the value of privacy in RTB. *accessed on* (2016), 05–21.
 - [51] osano [n. d.]. Osano. <https://www.osano.com/solutions/privacy-program-management-software>. Accessed: 2025.
 - [52] Michalis Pachilakis, Panagiotis Papadopoulos, Evangelos P Markatos, and Nicolas Kourtellis. 2019. No more chasing waterfalls: a measurement study of the header bidding ad-ecosystem. In *Proceedings of the Internet Measurement Conference*. 280–293.
 - [53] Jibesh Patra, Pooja N Dixit, and Michael Pradel. 2018. Conflictjs: finding and understanding conflicts between javascript libraries. In *Proceedings of the 40th International Conference on Software Engineering*. 741–751.
 - [54] peterlowes [n. d.]. Peter Lowe’s list. <http://pgl.yoyo.org/adserver>. Accessed: 2025.
 - [55] Michael Pradel, Parker Schuh, and Koushik Sen. 2015. TypeDevil: Dynamic type inconsistency analysis for JavaScript. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1. *IEEE*, 3145324 (2015).
 - [56] Tongwei Ren, Alexander Wittman, Lorenzo De Carli, and Drew Davidson. 2021. An analysis of first-party cookie exfiltration due to cname redirections. In *Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb)*.
 - [57] Jukka Ruohonen, Joonas Salovaara, and Ville Leppänen. 2018. On the Integrity of Cross-Origin JavaScripts. In *ICT Systems Security and Privacy Protection: 33rd IFIP TC 11 International Conference, SEC 2018, Held at the 24th IFIP World Computer Congress, WCC 2018, Poznan, Poland, September 18-20, 2018, Proceedings* 33. Springer, 385–398.
 - [58] safariPartitioning [n. d.]. Partitioned Third-Party Storage. <https://webkit.org/tracking-prevention/#partitioned-third-party-storage>. Accessed: 2025.
 - [59] sameoriginpolicy [n. d.]. Same-origin policy. <https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin-policy>. Accessed: 2025.
 - [60] Iskander Sanchez-Rola, Matteo Dell’Amico, Davide Balzarotti, Pierre-Antoine Vervier, and Leyla Bilge. 2021. Journey to the Center of the Cookie Ecosystem: Unraveling Actors’ Roles and Relationships. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1990–2004.
 - [61] Shopifykeepalive [n. d.]. Shopify Cookie Policy. <https://www.shopify.com/legal/cookies>. Accessed: 2025.
 - [62] Michael Smith, Pete Snyder, Benjamin Livshits, and Deian Stefan. 2021. Sugarcoat: Programmatically generating privacy-preserving, web-compatible resource replacements for content blocking. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2844–2857.
 - [63] Peter Snyder, Antoine Vastel, and Ben Livshits. 2020. Who filters the filters: Understanding the growth, usefulness and efficiency of crowdsourced ad blocking. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2 (2020), 1–24.
 - [64] squid [n. d.]. Squid blacklist. <http://www.squidblacklist.org/downloads/sbl-adblock.acl>. Accessed: 2025.
 - [65] Grant Storey, Dillon Reisman, Jonathan Mayer, and Arvind Narayanan. 2017. The future of ad blocking: An analytical framework and new techniques. *arXiv preprint arXiv:1705.08568* (2017).
 - [66] thirdpartyblockinchrome [n. d.]. Update on the plan for phase-out of third-party cookies on Chrome. <https://privacyandbox.com/news/update-on-the-plan-for-phase-out-of-third-party-cookies-on-chrome/>. Accessed: 2025.
 - [67] thirdpartyblockinedge [n. d.]. Introducing tracking prevention, now available in Microsoft Edge preview builds. <https://blogs.windows.com/msedgedev/2019/06/27/tracking-prevention-microsoft-edge-preview/>. Accessed: 2025.
 - [68] thirdpartyblockinsafari [n. d.]. Full Third-Party Cookie Blocking and More. <https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more>. Accessed: 2025.
 - [69] tiktokpixel [n. d.]. About TikTok Pixel. <https://ads.tiktok.com/help/article/using-cookies-with-tiktok-pixel>. Accessed: 2025.
 - [70] uMatrix [n. d.]. uMatrix. <https://chromewebstore.google.com/detail/umatrix/ogcfmafjalgifgnmanfminiepoiejdcf>. Accessed: 2025.

- [71] Yash Vekaria, Yohan Beugin, Shaoor Munir, Gunes Acar, Nataliia Bielova, Steven Englehardt, Umar Iqbal, Alexandros Kapravelos, Pierre Laperdrix, Nick Nikiforakis, et al. 2025. SoK: Advances and Open Problems in Web Tracking. *arXiv preprint arXiv:2506.14057* (2025).
- [72] warning [n. d.]. Warning removal list. <https://easylist-downloads.adblockplus.org/antiadblockfilters.txt>. Accessed: 2025.
- [73] John Wilander. [n. d.]. Intelligent Tracking Prevention. <https://webkit.org/blog/7675/intelligent-tracking-prevention/>. Accessed: 2025.
- [74] John Wilander. 2019. Intelligent Tracking Prevention 2.3. <https://webkit.org/blog/9521/intelligent-tracking-prevention-2-3/>.
- [75] John Wilander. 2020. CNAME Cloaking and Bounce Tracking Defense. <https://webkit.org/blog/11338/cname-cloaking-and-bounce-tracking-defense/>. WebKit blog post.
- [76] Mingxue Zhang and Wei Meng. 2020. Detecting and understanding JavaScript global identifier conflicts on the web. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 38–49.

A Appendix

A.1 Frequently overwritten or deleted cookies

Table 5 presents the frequently manipulated cookie pairs by entities across 20,000 websites.

B Top cross-domain manipulators

Figure 8 presents top Top 20 script-hosting domains engaged in cross-domain overwriting and deleting.

C Absolute Distributions (Linear Scale)

To complement the log-scale views in Section 7.3, we plot the same paired distributions on a *linear* y-axis (milliseconds) in Figure 9. The linear scale highlights absolute timing while making clear how a small number of very slow pages can extend the whiskers/outliers.

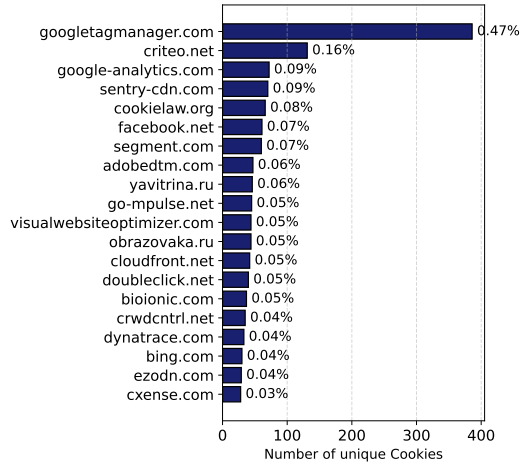
We also include the per-site *overhead ratio* (With/No) on a *linear* y-axis in Figure 10. Whereas the log-scale ratio in Figure 7 makes multiplicative tails visually comparable, the linear view emphasizes deviations near parity (1.0): small slowdowns or speedups are easier to read, while very large ratios are visually compressed.

D Ethics

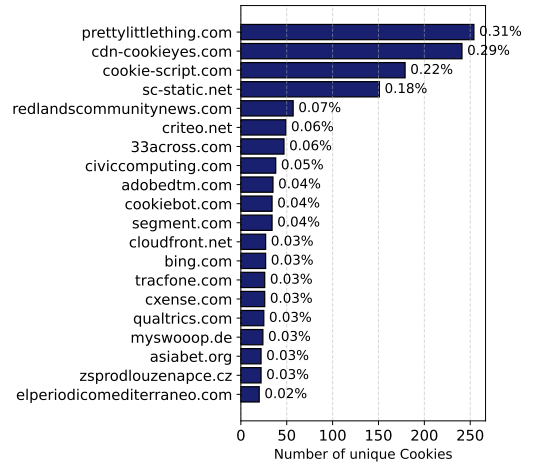
This study adheres to established ethical guidelines for research involving data collection and analysis. Our methodology is based solely on the examination of publicly available data retrieved from websites. Specifically, we analyze cookies stored on our own machines and outbound network requests initiated during automated browsing. At no point did our research involve the collection, interception, or processing of any real user’s Personally Identifiable Information (PII). We exclusively analyzed publicly accessible scripts and configuration data as served by websites. Therefore, we believe our work does not raise any ethical concerns.

Manipulation Type	Cookie Name	Creator Domain	Number of Manipulator Entities	Top 3 Manipulator Entities
Overwriting	_fbp	facebook.net	132	Functional Software, Google, Segment.io
	OptanonConsent	cookiecrlaw.org	132	Google, New Relic, WarnerMedia
	_ga	googletagmanager.com	86	Gatehouse Media, Intergi Entertainment, Segment.io
	_gcl_au	googletagmanager.com	49	playbetter.com, Cybot ApS, bombayshirts.com
	_uetvid	bing.com	48	Segment.io, Tealium, forseasky.com
	_uetid	bing.com	45	Segment.io, Tealium, forseasky.com
	ajs_anonymous_id	segment.com	45	Functional Software, Journal Publishing, conwaydailysun.com
	cto_bundle	criteo.com	39	Future Plc, script.ac, CacheNetworks
	_gid	google-analytics.com	34	Olark, Functional Software, Intergi Entertainment
	utag_main	tiqcdn.com	33	UNIDAD Editorial SA, medica.com, Optimizely
Deleting	_uetvid	bing.com	54	Tealium, Segment.io, cookie-script.com
	_uetid	bing.com	51	Tealium, Segment.io, cookie-script.com
	_ga	googletagmanager.com	27	cdn-cookieyes.com, Civic Computing, cookie-script.com
	_fbp	facebook.net	18	cdn-cookieyes.com, cookie-script.com, Civic Computing
	_gid	google-analytics.com	18	cdn-cookieyes.com, cookie-script.com, elfsborg.com
	_gcl_au	googletagmanager.com	18	cdn-cookieyes.com, cookie-script.com, Civic Computing
	_cookie_test	cxense.com	15	Enreach, optable.co, Canadian
	_ga	google-analytics.com	14	cdn-cookieyes.com, cookie-script.com, elfsborg.com
	ajs_user_id	segment.com	13	Functional Software, solvhealth.com, printify.com
	_screload	snapchat.com	11	Cisco, sparksites.io, acdc.com

Table 5: Frequently overwritten and deleted cookies by other entities

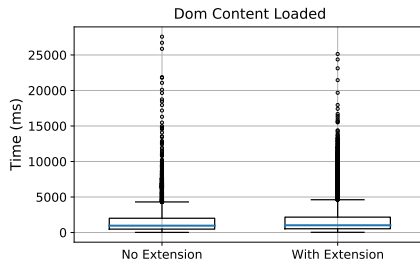


(a) Domains engaged in cross-domain cookie overwriting

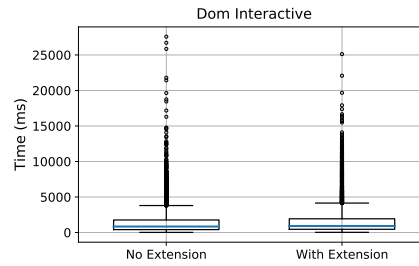


(b) Domains engaged in cross-domain cookie deleting

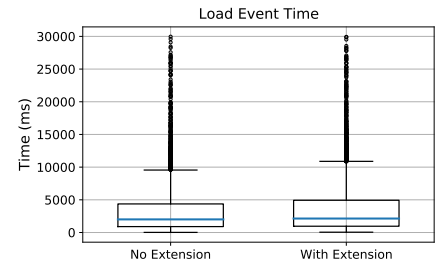
Figure 8: Top 20 script-hosting domains engaged in cross-domain overwriting and deleting unique cookies across 20k websites ranked by ranked by the number of cookies they manipulate.



(a) DOM Content Loaded.



(b) DOM Interactive.



(c) Load Event Time.

Figure 9: Paired boxplots in milliseconds (No Extension vs. With Extension).

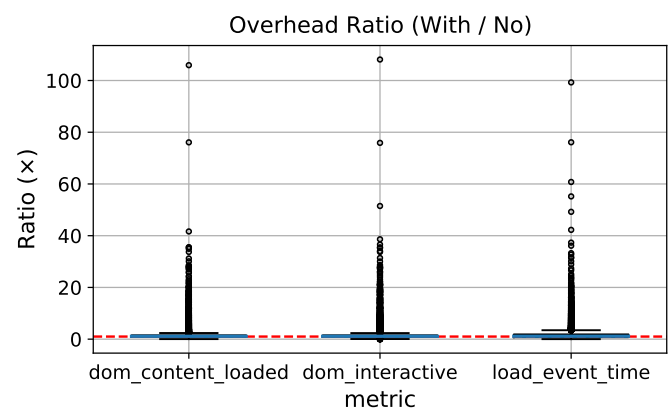


Figure 10: Per-site overhead ratio (With COOKIEGUARD / No COOKIEGUARD), linear y-axis.