Noname manuscript No.
(will be inserted by the editor)

Information Flow Control for Comparative Privacy Analyses

Zubair Ahmad · Stefano Calzavara · Samuele Casarin · Ben Stock

Received: date / Accepted: date

Abstract The prevalence of web tracking and its key characteristics have been extensively investigated by the research community by means of large-scale web measurements. Most such measurements however are limited to the choice of a specific client used for data collection, which is insufficient to characterize the relative privacy guarantees offered by the adoption of different clients to access the Web. Recent work on *comparative* privacy analyses involving multiple clients is still preliminary and relies on relatively simple heuristics to detect web tracking based on the inspection of HTTP requests, cookies and API usage. In this paper, we propose a more sophisticated methodology based on information flow tracking, which is better suited for the complexity of comparing tracking behavior observed in different clients. After clarifying the key challenges of comparative privacy analyses, we apply our methodology to investigate web tracking practices on the top 10k websites from Tranco as observed by different clients, i.e., Firefox and Brave, under different configuration settings. Our analysis estimates information flow reduction to quantify the privacy benefits offered by the filter lists implemented in Firefox and Brave, as well as the effectiveness of their partitioned storage mechanism against cross-site tracking.

Z. Ahmad
Università Ca' Foscari Venezia, Venice, Italy
E-mail: zubair.ahmad@unive.it

S. Calzavara
Università Ca' Foscari Venezia, Venice, Italy
E-mail: stefano.calzavara@unive.it

S. Casarin
Università Ca' Foscari Venezia, Venice, Italy
E-mail: samuele.casarin@unive.it

B. Stock
CISPA Helmholtz Center for Information Security,
Saarbrücken, Germany
E-mail: stock@cispa.de

Keywords Web tracking · Privacy protection · Information flow control · Web measurement

1 Introduction

Web tracking is prominent on the Web and a key component of the business model of many web companies. While web tracking is useful to provide personalized assets like interesting news and engaging advertisement, it also poses significant privacy concerns and for this reason received significant attention by the research community [13, 25]. In the last few years, this attention reached even privacy regulators, giving rise to well-known laws like the General Data Protection Regulation enforced in the European Union [16]. From a more technical perspective, the explosive developments in the ad ecosystem engage the browser vendors and the tracking industry in a competitive race [4, 5, 14]. To meet the increasing user demand for content blocking, and make the browsing environment more private [22], browser vendors are continuously integrating different tracking protection mechanisms in their browsers. Those tracking protection mechanisms intend to target and block social media trackers, cross-site tracking, fingerprinters, crypto miners, and other tracking content.

The effectiveness of such technologies has been investigated by the research community, most notably by means of *web measurements* [13, 15, 28]. Measurement studies perform large-scale analyses over the Web by automatically collecting data from popular websites and detecting interactions with web trackers to reason about privacy threats. Most of the research in the field however is limited to the choice of a specific client used for data collection, e.g., Google Chrome or the popular OpenWPM client used for privacy research. This is useful to analyze web tracking from the point of view of a specific client, but cannot characterize the privacy guarantees offered by the use of different clients. Considering the ever-increasing popularity of privacy-enhancing clients like Brave and the introduction of recent tracking countermeasures like partitioned storage, *comparative* analyses that measure the relative privacy guarantees offered by the use of different clients on existing websites are desired.

Contrary to traditional privacy analyses, comparative privacy analyses received just limited attention by the research community [21, 27, 36]. Zafar and Das compared the privacy guarantees offered by different mobile browsers through a combination of static and dynamic analysis [36]. Pradeep et al. instead conducted a large-scale and multi-directional analysis on a large pool of Android browsers from different app stores [27]. These studies are useful because they might guide privacy-savvy users in the choice of their preferred browser to access the Web, but they rely on relatively simple heuristics to detect web tracking based on the inspection of HTTP requests, cookies, and API usage. We here call for a more sophisticated and accurate approach, due to the observation that comparative analyses are difficult to carry out correctly: for example, recent work by Roth et al. unveiled that specific client characteristics may affect the security headers delivered by web applications, thus affecting the correctness of web measurements [30]. Since web tracking is arguably harder to measure than security headers, we expect similar challenges to also arise in web privacy measurements: correctly comparing the tracking behavior detected in different clients is not straightforward, because any observed differences are not necessarily related to better or worse privacy controls. For example, many websites are dynamic in nature and render differently in different clients, irrespective of privacy enforcement. As another, even more concerning example, just observing the content of the cookie jar is insufficient to draw conclusions about privacy enforcement: the mere presence of tracking cookies in the cookie jar does not suffice to conclude that web tracking is taking place, because privacy enforcement might operate by preventing their communication to known trackers rather than their client-side storage.

Contributions

In this paper, we aim to improve our understanding of the methodological aspects of comparative privacy analyses to support both privacy researchers who are interested in performing similar measurements and web users who are interested in the output of such analyses. In particular, we make the following contributions:

1. We identify the most relevant challenges to overcome to perform a sound measurement study comparing the relative privacy guarantees offered by different clients, i.e., to empirically measure the level of privacy protection they offer to web users on existing websites in the wild (Section 3);
2. We present a methodology to perform comparative privacy analyses that leverages information flow control [31] to deal with the identified challenges. In particular, we propose the use of information flow control to accurately characterize the presence of web tracking and enable a rigorous empirical comparison of different privacy-enhanced clients based on information flows bearing tracking identifiers (Section 4);
3. We report on the results of a measurement study based on our methodology, where we estimate the prevalence of different forms of tracking on a baseline client equipped with information flow control. We then compare the privacy guarantees offered by the use of different privacy-enhanced clients (Firefox and Brave) on the same set of websites, estimating their information flow reduction to measure the corresponding impact on common web tracking practices (Section 5).

2 Background

We here present the key necessary ingredients required to appreciate the rest of the paper. We first introduce details about how the web platform works, we then discuss web tracking and we finally overview defenses against web tracking.

2.1 Basic Web Concepts

The HTTP protocol and its secure variant HTTPS allow browsers to access web applications based on the client-server model. Browsers protect web applications by means of the Same Origin Policy (SOP), which regulates accesses by subjects (scripts) to objects (web resources) at the *origin* boundary, i.e., based on their protocol, host and port. For example, scripts running on a web page served by `www.evil.com` cannot access resources owned by `www.good.com` due to an origin mismatch. Similarly, when `www.good.com` loads content from `www.evil.com` within an iframe (sub-document), that content maintains its origin and is not entitled to access resources owned by `www.good.com`. The story is different though when a script is loaded from `www.evil.com` directly into the main document of a web page served by `www.good.com`: in this case, the script inherits the origin of its embedding web page and is allowed to access its resources. Besides the notion of origin, it is useful here to introduce the concept of *site* (or eTLD+1) of a URL, i.e., its domain name paired with the effective top-level domain: for example, `a.good.com` and `b.good.com` are both sub-domains of the same site `good.com`. The site is the registrable part of the domain name, which identifies the entity who controls the domain.

SOP-protected resources are variegated, but here we are particularly concerned about *client-side storage*, i.e., storage that web browsers make available to web applications for their needs. The most popular forms of client-side storage are *cookies* and *web storage*, which are both based on the key-value paradigm. Specifically, each cookie and web storage item can be thought as a pair (k, v), where k is the identifying key and v is the stored value. Access to specific storage items is performed through their key: in the case of cookies the key is identified by parsing the string returned by the `document.cookie` property, while access to the web storage is more disciplined and can be done by directly requesting the key through the `getItem` method. An important difference between cookies and web storage is that, while the latter can only be accessed through JavaScript, the former are also automatically attached by the browser to specific HTTP requests towards their owners, i.e., web applications can access cookies through HTTP requests even when no JavaScript is running.

Client-side storage can be made *persistent*, i.e., preserved even when the browser is closed. In the case of cookies this is done by specifying an appropriately high value of the `Expires` attribute, while web storage is partitioned between the persistent local storage and the ephemeral session storage (which is purged when the browser is closed). Persistent client-side storage constitutes the foundations of traditional web tracking, as discussed below. We use the term *client-side storage item* (CSSI for short) to uniformly refer to both cookies and local storage items when the distinction is immaterial.

2.2 Web Tracking

The term *web tracking* identifies a wide set of practices whereby a tracker embedded within a web page collects information about the requesting client for analytics and advertisement purposes. Web tracking can be *stateful* or *stateless*: the former uses persistent client-side storage, i.e., cookies and local storage, to store unique identifiers used to pinpoint the client; the latter, instead, uses specific client characteristics, e.g., operating system, language, fonts, to construct a fingerprint of the client without relying on client-side storage. In this work, we focus on stateful tracking alone and we leave the investigation of stateless tracking to future work.

Web tracking is also normally categorized as *cross-site tracking* or *same-site tracking*, based on whether the tracker has the ability to correlate visits performed by the same client across different websites [23]. When stateful tracking is used, cross-site tracking is typically done by means of *third-party storage*, i.e., storage set within an iframe served from a different origin. The intuition here is that, if `site1.com` opens an iframe towards `tracker.com`, SOP requires scripts running in the iframe to set CSSIs which are owned by `tracker.com`. These items are then visible to the tracker when `site2.com` opens an iframe towards `tracker.com`, thus allowing the tracker to detect that the same client accessed both `site1.com` and `site2.com`. Same-site tracking, instead, uses *first-party storage*, i.e., storage set within the same origin of the main document. In particular, if both `site1.com` and `site2.com` load a script from `tracker.com`, then any CSSIs set by the script cannot be accessed on the other website due to SOP, hence the tracker cannot determine that the same client accessed both websites, thus limiting privacy concerns. In line with recent work [25], we further categorize same-site tracking as *pure* or *cross-domain*: pure same-site tracking shares client identifiers with trackers which were directly loaded by the web page, while cross-domain same-site tracking exposes client identifiers to different trackers. Cross-domain same-site tracking has more delicate privacy implications than pure same-site tracking, because client identifiers are shared with third-parties which might even be completely unknown to the web page.

2.3 Countermeasures Against Web Tracking

The rapid proliferation of online tracking promoted concerns about user privacy, the invasiveness of ads, and the potential misuse of personal data. To address these concerns, different countermeasures have been initiated in order to offer additional privacy controls. The main ingredients are *filter lists*, i.e., blocklists of known trackers, and *partitioned storage*, which provides improved isolation for third-party storage. In our discussion, we primarily focus on Firefox and Brave, which are the two browsers considered in our experimental evaluation.

2.3.1 Partitioned Storage

For over a decade, browsers allowed advertisers to perform cross-site tracking functions with third-party storage. Because this capability presents a threat to privacy, several popular browsers including Mozilla Firefox and Brave have implemented *partitioned storage* [28,35] to isolate third-party storage so it cannot be used for cross-site tracking. The intuition of partitioned storage is that third-party storage is partitioned based on the embedding document, e.g., if both `site1.com` and `site2.com` open an iframe towards `tracker.com`, scripts running in these iframes may set CSSIs which are associated to separate instances of `tracker.com`. This means that a client identifier set in the iframe at `site1.com` will be different from a client identifier set in iframe at `site2.com`, thus preventing `tracker.com` from understanding that the same client accessed both sites, i.e., thwarting cross-site tracking.

2.3.2 Firefox

Firefox started its journey towards users' privacy by experimenting with third-party cookie blocking in 2013, but did not ship default-on third-party blocking at that time [6, 7]. In October 2018, Firefox introduced Enhanced Tracking Protection (ETP) [26] as an experiment in the Nightly version of Firefox, providing default-on third-party cookie blocking. This move was to provide users with more control over their online data and improve performance. ETP utilized a list from Disconnect [12], a privacy-focused company. This list categorized trackers and enabled Firefox to decide which ones to block. The primary goal was to prevent third-party trackers from harvesting user data across websites, a process that creates detailed user profiles for targeted advertising and other purposes. Firefox now enforces ETP with partitioned storage activated by default, thus blocking cross-site tracking through third-party storage, but its privacy settings allow one to deactivate some privacy controls.

2.3.3 Brave

To address the growing concerns related to online advertisement and web tracking, Brendan Eich and Brian Bondy co-founded Brave [8] in 2016, a privacy-oriented browser based on Chromium. Since its inception, Brave has a built-in ad blocker that not only blocks intrusive and annoying ads, but also enhances users' privacy by preventing different tracking behaviors. Brave took a number of steps to secure the browsing environment by incorporating filter lists within its privacy mechanism *Shields* [5] and activating partitioned storage by default [4]. This approach blocks more trackers than Firefox and similarly prevents cross-site tracking through third-party storage. Brave can also be configured in Aggressive mode, which enforces further protection against first-party trackers, at the risk of additional compatibility issues with existing websites.

3 Challenges of Comparative Privacy Analyses

We here present the main challenges that a sound comparative privacy analysis should deal with. In particular, we discuss how differences in web tracking behavior can be observed across browsers and why it is difficult to attribute them to actual changes in privacy enforcement.

3.1 Motivating Example

To explain why performing sound comparative privacy analyses is challenging, we present a simple motivating example. Assume that a privacy researcher uses Firefox to automatically access the website `https://www.publisher.com`. The web page loads a script from `https://www.tracker.com`, which sets a cookie `trk` with value `Firefox|a677bdee89|1700643590`, where `a677bdee89` is a client identifier and `1700643590` is a timestamp. The client identifier is then read from the cookie jar and communicated to `https://adv.tracker.com` using a GET request. The privacy researcher then uses a different client, e.g., Brave, to access `https://www.publisher.com` again. We discuss two possible scenarios that the researcher might observe in practice and some of the challenges they have to solve to draw meaningful conclusions.

3.1.1 First Scenario

Assume that the researcher does not detect any communication towards `https://adv.tracker.com` from the web page. It might be tempting to conclude that the privacy controls of Brave prevented a possible privacy leak, however this is not necessarily the case. There might be multiple reasons why communication towards `https://adv.tracker.com` did not take place besides privacy enforcement. For example:

1. Loading of `https://www.tracker.com` failed or did not complete due to network errors, hence no request to `https://adv.tracker.com` was sent;
2. Brave was redirected to a different web page when trying to access `https://www.publisher.com`, e.g., the browser accessed a localized version of the website embedding different trackers;
3. The same web page of the original Firefox visit was correctly accessed and rendered, yet the page is dynamic in nature and loads different scripts in different visits, e.g., to provide personalized advertisement.

Some of these scenarios are straightforward to detect and researchers routinely deal with them, e.g., checking the status code and the final URL of the collected data might rule out the first and the second case. The third case however cannot be ruled out in general, hence the absence of a request to `https://adv.tracker.com` cannot conclusively prove that Brave prevented the privacy leak.

One might then choose to take a different route and monitor whether the request is actively blocked by Brave, which would give evidence of privacy enforcement taking place. Unfortunately, it is surprisingly challenging to do that correctly. A possible idea would be monitoring console errors: for example, Brave produces a `net::ERR_BLOCKED_BY_CLIENT` error when a request is blocked for privacy reasons. Monitoring such errors does not suffice, due to *script dependencies*. Indeed, even though communication towards `https://adv.tracker.com` is not actively blocked by Brave, the request to `https://adv.tracker.com` might still not take place because the sender script is not loaded as the result of some other script request being actively blocked, which suffices to prevent the privacy leak in practice.

3.1.2 Second Scenario

Assume then that the researcher detects a communication towards `https://adv.tracker.com` from the web page. One might think that this suffices to conclude that Brave does not prevent a possible privacy leak, however this conclusion might be wrong because the request might not contain a client identifier. For example, it is possible that tracking protection directly or indirectly prevented the cookie `trk` from being set, leading to a request that includes just the rest of the available information, e.g., the user agent and a timestamp which are available through API calls. Hence, one should find a way to match and compare requests sent by different clients, but this is again difficult for multiple reasons: some information in the request is going to be inherently different, e.g., the client identifier and other state-dependent parameters are going to change, while some other component is not necessarily different, but might legitimately differ as a matter of fact, e.g., communication might take place towards a different subdomain like `backup.tracker.com` even when visiting the same website multiple times.

3.2 Main Challenges

Elaborating on the previous example, we identify three main challenges that a sound comparative privacy analysis should solve.

3.2.1 Detecting Client Identifiers

Web tracking is based on the ability of the tracker to uniquely pinpoint a specific client, hence the effective detection of client identifiers in network requests is an important prerequisite to any form of privacy analysis. Unfortunately, due to the lack of access to the server-side logic of web trackers, detection of client identifiers can only be performed by means of heuristics [10, 28]. We do not touch on this problem in this paper, since different heuristics have already been proposed in the past and, in this respect, a comparative privacy analysis does not differ from a traditional privacy analysis. We describe our approach to the detection of client identifiers in Section 4.1.

3.2.2 Detecting Information Flows

Stateful tracking requires the communication of client identifiers from client-side storage to the network. Looking for client identifiers in either client-side storage alone or network communication alone would lead to an over-approximated analysis, because identifiers saved in client-side storage are not necessarily communicated over the network and identifiers sent over the network are not necessarily persisted in client-side storage. These over-approximated approaches are limited in general, but they are particularly concerning in the field of comparative analyses, because privacy-enhancing clients might use different techniques to enforce privacy protection. For example, assume a client forbids communication with a set of blocklisted domains: in that case, a known tracker might set a client identifier in the client-side storage, but this identifier may not be communicated back to the tracker due to blocklisting, hence the mere presence of the identifier might lead to the wrong conclusion that tracking is possible. Prior work in the field normally used simple heuristics to match client-side storage against network requests, e.g., checking whether the same identifier stored in the client also occurs within a network request [3, 13, 15, 29]. This approach relies on syntactic matches and does not track through the JavaScript logic whether the identifier actually flows from the client-side storage to the network, which is a fundamental requirement of stateful tracking. Moreover, this solution is brittle, because any syntactic transformation of the client identifier before network communication might lead to failures in tracking detection, unless specific data transformations like Base64 are hard-coded to account for them. As such, prior heuristics might suffer from both false positives and false negatives, leading to both over-reporting and under-reporting.

To better detect stateful tracking, we advocate the use of *information flow control*, a popular technique spreading extensive research over the last decades [31]. Information flow control tracks how information flows from *sources*, e.g., cookies and local storage, to *sinks*, i.e., APIs used for network communication. A popular approach to information flow control is *taint tracking* [32], which can be roughly summarized as follows: when data is first read from a source, it becomes tainted; the taint is then propagated through the program execution, e.g., concatenating tainted data to untainted data produces tainted data, until it reaches a sink. This way, one can precisely track information flows through the program execution, which is particularly important in the context of same-site tracking because cookies and local storage are accessed and modified by JavaScript code. In our example, taint tracking can identify a flow from the cookie jar to a network sink after that the value of the cookie `trk` is parsed to extract just its intermediate component (the client identifier). Note that taint tracking only identifies *explicit flows*, i.e., information flows based on data dependencies, and ignores *implicit flows* based on the control flow of the program; this is more efficient, less amenable to false positives and proves sufficient in many practical cases [34]. We present our approach to information flow tracking in Section 4.2.

3.2.3 Performing Meaningful Comparisons

A particularly delicate aspect of a comparative privacy analysis is that the conclusions drawn from the observations performed by different clients may be unreliable, as discussed in our motivating example. In particular, one has to implement safeguards against artifacts arising from measurement errors and other side-conditions influencing the empirical experiment, so that any observed difference can be legitimately attributed to different forms of privacy enforcement. Possible threats to validity of a comparative privacy analysis include the choice of different client characteristics upon measurement, temporal drift across website accesses from different clients, the dynamic behavior exhibited by modern websites when they are rendered multiple times, etc. We discuss our comparative approach and multiple safeguards against such threats to validity in Section 4.3.

4 Methodology

At a high level, our methodology quantifies the privacy improvements offered by the use of a privacy-enhancing client as follows:

1. We first use a baseline client without any form of privacy protection to detect the CSSIs which can potentially be used for effective client identification, e.g., because they contain long and random strings that might act as unique identifiers. This is a necessary precondition for any form of stateful tracking and a standard component of privacy measurements [10].
2. We characterize the privacy risks of same-site tracking in terms of information flows from first-party CSSIs storing client identifiers to the network. We similarly identify room for cross-site tracking by looking for information flows originating from third-party CSSIs storing client identifiers and ending in network sinks. Moreover, we look for the presence of client identifiers in third-party cookies, since they are automatically shared with trackers upon network requests and may readily enable cross-site tracking.
3. We measure the privacy improvements enabled by the use of a privacy-enhanced client with respect to our baseline client, in particular by estimating their information flow reduction. This requires some care to attribute observed improvements to privacy enforcement rather than accidental factors or measurement errors.

Our baseline client is Project Foxhound[1], a web browser with dynamic information flow tracking based on Firefox, version 114.0.2 [19]. Project Foxhound uses taint tracking to detect explicit information flows from a predefined list of sources to a predefined list of sinks, thus allowing us to readily implement step 2 of our methodology. We run Project Foxhound with all forms of tracking protection deactivated. In particular, we programmatically disable all features provided by Enhanced Tracking Protection, including the blocking of tracking content and fingerprinters, as well as state partitioning for cookies and local storage.

4.1 Detecting Client Identifiers

By definition, web tracking can only happen when a tracker is able to uniquely pinpoint a given client. To detect client identifiers set in client-side storage, we use the heuristic approach originally proposed for cookies by Chen et al. [10] and we integrate it with a recent improvement recommended to reduce its number of false

[1] `https://github.com/SAP/project-foxhound`

positives [28]. Concretely, a CSSI (k, v) is marked as a *client identifier* if and only if all the following conditions hold:

1. The item does not get deleted when closing the browser, i.e., it is either a cookie with a positive value of the `Expires` attribute or a local storage item;
2. The item persists with the same value when the page is reloaded;
3. The length of the unquoted value v is at least 8 characters;
4. An item (k, v') with the same key k is set when visiting the same web page from a fresh browser instance;
5. The values v and v' are "significantly different", i.e., the item allows one to readily tell apart different browser instances.

The notion of "significantly different" values is also taken from Chen et al. [10]. Specifically, values are pre-processed to remove all the timestamps occurring therein and to recursively strip their longest common sub-sequence, until the longest common sub-sequence includes at most two characters. The similarity score of the residual values is then computed using the Ratcliff-Obershelp algorithm, setting a threshold of 66%. Values whose similarity score is below the threshold are considered significantly different. The second condition above was proposed in [28] to ensure that ephemeral information whose scope is limited to a single page request is not incorrectly marked as useful for tracking.

4.2 Web Tracking Detection

After detecting possible client identifiers in CSSIs, we filter the information flows detected by Project Foxhound to just keep those satisfying the following conditions:

1. The source is either the cookie jar or the local storage, i.e., a source of CSSIs;
2. The sink is any JavaScript method or HTML element that can be used for network communication, e.g., the `fetch` method or the `src` attribute of an image tag;
3. The information read from the source is a client identifier rather than a generic CSSI.

Condition 3 is straightforward to check for accesses to the local storage, since they are performed by referencing to keys, e.g., as in `localStorage.getItem("key")`, hence it is possible to know exactly which CSSI was read from the local storage. Unfortunately, accesses to the cookie jar are not as disciplined as local storage accesses, since the `document.cookie` property returns a string representation of the entire content of the cookie jar and the web application has to parse the output of `document.cookie` to retrieve a specific cookie out of it. Since we are only interested in flows communicating client identifiers, we must attribute the read cookies to a detected flow. We do this "a posteriori" as follows:

1. We parse the output of `document.cookie` to extract all the cookies in the (k, v) format;
2. We extract from taint reports the tainted value v' reaching the network sink, which is associated with a specific part of the HTTP request payload, such as the request's URL or the body of a POST request;
3. If there exist a cookie (k, v) such that the longest common substring between v and v' is longer than 8 characters, we associate the cookie k to the detected flow. To make the attribution more precise, we restrict the longest substring match to the part of v' which is within the *taint bounds* returned by Project Foxhound, which define the part of v' which includes tainted information.

This heuristics might fail, in particular when the cookie value v is transformed such that v' does not share any similarity with v. This is a limitation of our tool chain, but we quantify in our experimental section (Section 5) that the proposed heuristics is effective in practice and succeeds in more than 90% of the cases.

For each website, we collect information flows twice to account for the case where a CSSI is set in the first visit to the website and only communicated over the network in subsequent visits. After the second visit, we take the union of the flows after deduplicating them based on the following information:

1. The source (cookie jar or local storage) and the set of read keys (cookies or local storage items);
2. The sink, e.g., the `fetch` method, and the site of the network target, e.g., `tracker.com`;
3. The URL of the script writing to the sink, e.g., `https://tracker.com/adv.js`, without the query string.

We finally use the labelled information flows to characterize web tracking as follows: if the flow involves first-party storage, we mark it as *same-site tracking*, either pure or cross-domain depending on the script requests sent by the web page; if instead the flow involves third-party storage, we mark it as *cross-site tracking*. Note that this is correct because we run Project Foxhound without any form of privacy protection, i.e., without partitioned storage in place. We use information flows to expose web trackers by observing the target of the network sink, which we abstract at the site granularity. Besides the results of this information flow analysis, we further collect all the third-party cookies storing client identifiers: since cookies are automatically communicated within HTTP requests, their presence alone enables cross-site tracking, even when no information flow is detected by our taint tracker. In particular, the owner of the third-party cookies can be marked as a web tracker in this case.

4.3 Comparative Privacy Analysis

The information computed in the previous section allows one to quantify the privacy protection offered by a privacy-enhancing client with respect to our baseline. In particular, we can check which information flows detected on Project Foxhound do not occur in a privacy-enhanced client C to quantify how the observed web tracking practices are affected by privacy protection. As we explained, performing such a comparative privacy analysis requires some care, because we have to ensure that any observed differences can be attributed to specific privacy protection mechanisms implemented by the privacy-enhanced client rather than to generic reasons. In particular, we discuss below relevant factors that might affect the validity of our findings and we explain how we mitigate them:

– *Client characteristics*: we run all the clients on the same machine, configuring them uniformly. In particular, we execute clients on the same operating system (Ubuntu 22.04.3 LTS) and we configure them to use the same language (English). This way, we mitigate any accidental differences arising from the use of a specific client configuration, which might, e.g., lead websites into performing different redirects depending on client localization [28]. We only preserve analysis results for those websites ending in the same final URL observed in Project Foxhound.
– *Temporal drift*: we run all the clients in parallel, so that each website is visited (approximately) at the same time by each client. This way, we minimize the risk of observing differences related to websites undergoing changes over time rather than some form of privacy protection being enforced by the clients.
– *Rate limiting*: accessing a particular website within a short time span from our unusually high number of clients may trigger rate-limiting mechanisms, temporarily locking our clients out of that website. To mitigate this effect, we schedule our data collection to ensure that no more than four clients can visit a given website simultaneously.
– *Dynamic behavior of websites*: since tracking scripts are often dynamically loaded by websites, different accesses to the same web page might reveal different information flows. Hence, the lack of an information flow might just be attributed to non-determinism. To mitigate the impact of non-determinism, we access each website five times and we only consider an information flow to be blocked by the privacy-enhanced client if none of the visits identifies the information flow based on the proposed techniques. We use the same idea to check whether a client identifier set in third-party cookies is removed by the presence of improved privacy controls.
– *Runtime errors*: generic failures in the rendering of a website might lead to missing information flows. If we are unable to fully render the website five times within a 60 seconds timeout, we discard the website and we do not consider it in our comparative analysis. A failure in just one of the five visits suffices to discard the website, because we want a reasonable number of observations to mitigate the impact of non-determinism.

Since the privacy-enhanced client C that we want to analyze does not necessarily support information flow tracking like our baseline Project Foxhound, we use multiple techniques to assess whether a previous information flow does not take place there:

1. We monitor the content of the relevant sources (cookie jar and local storage): if none of the keys read in the information flow is present in C, we are sure that the information flow does not occur in C;
2. We monitor the network requests: if there is no network request towards the site of the sink in C, we are sure that the information flow does not occur in C;
3. We monitor blocked script inclusions: if the script writing to the sink is not loaded in C, we are sure that the information flow does not occur in C.

If all these techniques fail, we conservatively assume that the information flow is not blocked by C. We use this approach rather than directly looking for the information flow within network requests, because matching network requests performed by different clients is far from straightforward, e.g., requests might be sent to different sub-domains or have different paths. We thus associate to the client C the following information:

1. If Project Foxhound identifies a same-site tracking flow f which is not blocked by C, we mark f as a same-site tracking flow for C;
2. If Project Foxhound identifies a cross-site tracking flow f which is not blocked by C, we mark f as a cross-site tracking flow for C if and only if C does not implement partitioned storage;
3. If Project Foxhound identifies a client identifier within a third-party cookie c which is present in C, we con-

sider it a cross-site identifier if and only if C does not implement partitioned storage.

Notice that partitioned storage effectively prevents cross-site tracking, because client identifiers set in a third-party position cannot be shared in different websites. We finally use the collected information to characterize web tracking practices on C, thus quantifying web privacy improvements with respect to same-site tracking and cross-site tracking.

4.4 Motivating Example Revisited

Back to our motivating example, assume that a privacy researcher accesses the website `https://www.publisher.com` using Project Foxhound. The web page loads a script from `https://www.tracker.com`, setting a cookie `trk` with value `Firefox|a677bdee89|1700643590`, where `a677bdee89` is a client identifier and `1700643590` is a timestamp. The client identifier is then read from the cookie jar and communicated to `https://adv.tracker.com` using a GET request, leading to an information flow. The privacy researcher now wants to understand whether the same tracking behavior would be blocked by Brave.

Our methodology requires Brave to be configured in the same way as Project Foxhound, so that, e.g., language configuration does not redirect Brave to a different localized version of the website where the flow is not present (by accident). This is also confirmed by inspecting the final URL accessed by Brave and comparing it against the one accessed by Project Foxhound. Access to the website would happen at the same time of Project Foxhound, so that the flow is unlikely to disappear due to website changes. The website is accessed five times to mitigate the effect of non-determinism: only if it renders correctly all the times, we assume to have enough observations to draw meaningful conclusions and we check whether the flow appears also in Brave or not. This is done by monitoring the cookie jar, network requests and script inclusions of Brave: only if the cookie `trk` is absent, communication towards `tracker.com` does not take place or the script is not loaded all the five times, we assume to have enough evidence that Brave performs active blocking.

4.5 Limitations

We highlight and motivate a limitation of our analysis. Our comparative approach starts from a baseline client and quantifies privacy improvements enabled by the use of a privacy-enhanced client. Conversely, it does not look for new information flows which only arise in the privacy-enhanced client, e.g., because a sophisticated tracker adapts its behavior to bypass the privacy protection mechanisms put in place by the client. Although we do not expect these practices to be (yet) prevalent in the wild, it would be interesting to observe them. The reason why we do not generalize our methodology to also cover those cases is that performing dynamic information flow control within the browser is challenging. In particular, information flow control can be implemented with or without browser modifications, e.g., based on JavaScript rewriting [18]. The latter approach has the distinctive advantage of working on any browser, hence it can also be used to detect information flows in arbitrary privacy-enhanced clients, however it suffers from poor performance and is not amenable to large-scale analyses like the one we perform in this paper. Information flow control through browser modifications *à la* Project Foxhound is necessarily tied to a specific client, but is very efficient and only incurs a limited overhead compared to traditional browser accesses. This explains why we use a solution like Project Foxhound in our analysis.

A second limitation of our analysis is that we assume that first-party storage is always used for same-site tracking, which is the standard use case considering how SOP works. However, recent work showed that the blockage of third-party storage by different protection mechanisms in web browsers pushed trackers to abuse first-party storage for cross-site tracking as well [15, 25, 28]. Such abuses are not widespread though: for example, Randall et al. estimated the presence of UID smuggling in just 8% of the analyzed websites [28]. Moreover, cross-site tracking through first-party storage often requires the presence of a static identifier like an email address, which is not going to be present in all web pages. We thus stick to the standard assumption that first-party storage is primarily used for same-site tracking and we leave the investigation of possible abuses to future work.

5 Web Privacy Measurement

We now report the results of our large-scale privacy measurement performed on the top 10k websites from Tranco. We first discuss our experimental setup, we then present results for our baseline client Project Foxhound and we finally investigate the relative privacy guarantees offered by the adoption of different privacy-enhancing clients.

5.1 Experimental Setup

In our web measurement, we assess the privacy guarantees offered by different clients in terms of information flow reduction against our baseline client Project Foxhound, which is run with all forms of privacy protection deactivated (no partitioned storage, no ad-blocking in place). In particular, we consider the following clients:

- Firefox (Standard): an instance of Mozilla Firefox in its default configuration, i.e., cross-site tracking is blocked.
- Firefox (xPS): an instance of Mozilla Firefox with partitioned storage deactivated, i.e., cross-site tracking is allowed.
- Brave (Standard): an instance of Brave in its default configuration, i.e., all cross-site tracking and a significant amount of same-site tracking are blocked.
- Brave (Aggressive): an instance of Brave with aggressive privacy protection, which prevents additional forms of same-site tracking with respect to standard mode.

We motivate our choice of clients as follows. Project Foxhound is representative of a browser without any privacy controls, similar to Google Chrome in its default configuration. Firefox (Standard) is an example of a traditional browser with some additional privacy guarantees, but which does not pursue privacy as aggressively as Brave (Standard), which in turn represents the reference for privacy-savvy users. Firefox (xPS) does not really capture a typical choice of web users, but it is instructive to have because it allows us to measure the importance of partitioned storage alone. Finally, Brave (Aggressive) represents a browser with strict privacy controls, which might deliberately break compatibility with websites to prevent tracking. It is thus interesting to measure whether the improved privacy guarantees offered by Brave (Aggressive) are justified by its potential compatibility cost. Adding more clients to the picture would just be a matter of engineering effort, because our methodology is based on lightweight instrumentation of the tested clients to monitor the cookie jar, the local storage, network requests and script inclusions. Indeed, our methodology only assumes the implementation of information flow control, which is arguably the most complicated component, just in the baseline client.

Our automated crawler ran 22 browser instances in parallel, comprising two instances of Project Foxhound and five instances of both Firefox and Brave under two different privacy settings, as explained above. To simulate a setup which is closer to what real web users would observe, we ran all clients in headful mode, emulating a display with Xvfb.[2] This is particularly important for Brave, because a preliminary experiment on a small set of websites revealed that the headless version of Brave does not enforce privacy protection as strictly as the headful version. Our measurement was performed from a European IP address without interacting with websites in any ways, e.g., our crawler did not click over cookie banners. Note that automatically detecting and clicking over cookie banners is not a straightforward task [20]. Our crawl was stateless, i.e., it always spawned fresh browser instances, which is sufficient for our methodology and has the advantage of making results less dependent on the order of visits to the different websites. We performed our measurement on the Tranco Top 10k list downloaded on November 28, 2023.[3]

5.2 Baseline Client Results

Overall, we were able to successfully access 8,025 out of 10k websites (80%) using our baseline client Project Foxhound. Failures include generic errors due to domains being inaccessible, timeouts and other types of network errors which cannot be attributed to our tool chain. Taint tracking was then successfully performed on 6,817 out of 8,025 accessed websites, leading to a reasonably high success rate of 84%.

5.2.1 Client-Side Storage

Table 1 presents statistics about the collected CSSIs on Project Foxhound, broken down by type. In total, we identified the use of first-party cookies in 5,911 websites (86%) and the use of first-party local storage in 4,579 websites (67%). Client identifiers were detected in 35,767 out of 94,955 first-party CSSIs, i.e., around 38% of the first-party CSSIs have potential for same-site tracking; such identifiers were detected in 5,461 websites (80%). We also detected the use of third-party cookies in 2,319 websites (34%) and the use of third-party local storage in 1,378 websites (20%). In particular, we identified client identifiers in 11,634 out of 23,812 third-party CSSIs, i.e., roughly 48% of the third-party CSSIs might enable cross-site tracking; such identifiers were detected in 2,189 websites (32%).

This picture shows that there are way more websites potentially implementing some form of same-site tracking through first-party CSSIs than websites enabling cross-site tracking through third-party CSSIs nowadays, nevertheless room for cross-site tracking is

[2] `https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml`

[3] `https://tranco-list.eu/list/PZ46J`

Table 1 Client-side storage items set in the baseline client (Project Foxhound)

	Total		**First Party**		**Third Party**	
	#Occurrences	**#Websites**	**#Occurrences**	**#Websites**	**#Occurrences**	**#Websites**
Cookies	75,224	5,983	58,548	5,911	16,676	2,319
- including client identifiers	35,855	5,322	27,105	5,235	8,750	1,751
Local Storage Items	43,543	4,755	36,407	4,579	7,136	1,378
- including client identifiers	11,546	3,449	8,662	3,021	2,884	1,169
Client-Side Storage Items (CSSI)	118,767	6,079	94,955	6,025	23,812	2,445
- including client identifiers	47,401	5,569	35,767	5,461	11,634	2,189

still widespread despite the increasing popularity of partitioned storage among browser vendors. It is instructive that the frequency of client identifiers was higher in third-party CSSIs than in first-party CSSIs, likely suggesting that third-party storage is primarily used for web tracking, while first-party storage is also dedicated to other important tasks like session management.

5.2.2 Same-Site Tracking

Table 2 reports the number of information flows detected in a first-party position, which might enable same-site tracking. In total, we identified 52,995 flows from first-party CSSIs to the network on 4,565 websites (66%). Out of these, we were able to detect the exchange of client identifiers in 44,421 flows on 4,305 websites (63%). These cases are detected as instances of same-site tracking. It is instructive to remark that the use of information flow control pays off in our measurement: indeed, observe from Table 1 that we detected client identifiers in first-party CSSIs on 5,461 websites (80%). However, not all these client identifiers are eventually communicated over the network, i.e., the mere presence of client identifiers may over-approximate the prevalence of same-site tracking of around 19%. Indeed, recent work estimated the prevalence of Google Analytics on around 53-62% of the websites based on a simple inspection of HTTP requests [25], while our information flow analysis detected same-site flows towards Google Analytics in just 29% of the websites. Just checking the mere presence of the cookies `_ga` and `_gid` in a first-party context, we observed the presence of Google Analytics on 3,227 websites (47%), which is more in line with prior research [25] and confirms that simple heuristics based on syntactic matching might over-approximate the actual prevalence of web tracking. Later on, we also show that such heuristics may also suffer from false negatives and miss tracking flows in practice.

As to the privacy implications of same-site tracking, we observe that 16,374 out of 44,421 same-site tracking flows (37%) are marked as cross-domain, i.e., communication of client identifiers often involves trackers which were not explicitly loaded in the web page by means of script inclusion. Finally, the table also shows that cookie-based tracking is still more prevalent than storage-based tracking: we detected same-site tracking flows from the cookie jar to the network on 4,136 websites (60%), while same-site tracking flows from the local storage to the network were detected on just 1,366 websites (20%).

In the end, our information flow analysis discovered 1,157 distinct same-site trackers on 4,305 websites. The most popular same-site trackers are reported in Table 3. Despite this abundance of trackers, we identified just 17 trackers embedded in at least 100 websites, i.e., there are just a limited number of trackers dominating the tracking ecosystem. The table also shows for the most popular same-site trackers their category, based on the Disconnect list [12].

5.2.3 Cross-Site Tracking

Table 4 reports the number of information flows detected in a third-party position, which might enable cross-site tracking. In total, we identified 1,030 flows from third-party CSSIs to the network on 287 websites (4%). Out of these, we were able to detect the exchange of client identifiers in 860 flows on 256 websites (4%). These cases are detected as instances of cross-site tracking. These numbers show that cross-site tracking is less prevalent than same-site tracking nowadays, however they suffer from a somewhat opposite problem than what we observed for same-site tracking. As the mere presence of a client identifier in first-party CSSIs does not imply same-site tracking, the absence of information flows from third-party CSSIs to the network does not mean that cross-site tracking is not possible, in particular because third-party cookies are automatically attached to network requests. In other words, the simple presence of client identifiers within third-party cookies trivially enables cross-site tracking even though no information flow was detected by our taint tracker. Observe from Table 1 that this practice was detected on 1,751 websites (25%). If we add these websites to the set of those performing cross-site tracking based on the detected information flows while removing duplicates, we conclude that cross-site tracking was identified on 1,766

Table 2 Same-site tracking practices observed on the baseline client (Project Foxhound)

Same-Site Tracking	#Occurrences	#Websites
Detected information flows from first-party cookies to the network	46,990	4,371
- Information flows with at least one cookie associated	43,696	4,284
- Information flows communicating client identifiers	41,451	4,136
Detected information flows from first-party local storage to the network	6,005	2,211
- Information flows communicating client identifiers	2,970	1,366
Detected information flows from first-party CSSIs to the network	52,995	4,565
- Information flows communicating client identifiers	44,421	4,305
- Detected as pure same-site tracking	28,047	3,465
- Detected as cross-domain same-site tracking	16,374	3,381
- Number of different trackers based on the detected flows	1,157	4,305

Table 3 Most popular same-site trackers detected through information flows on the baseline client (Project Foxhound)

Same-Site Trackers	#Websites	Categories (Disconnect)
`google-analytics.com`	2,628	Analytics, Email
`doubleclick.net`	2,073	Advertising, Email
`google.com`	1,687	Content
`google.de`	1,280	Content
`facebook.com`	831	Social
`bing.com`	382	Advertising, Content
`yandex.com`	294	Content
`nr-data.net`	252	Analytics
`clarity.ms`	252	Analytics
`tiktok.com`	193	Advertising, Social
`criteo.com`	184	Advertising
`chartbeat.net`	159	Analytics

websites (25%). This shows that the use of information flow control only provides limited benefits when measuring cross-site tracking: our information flow analysis detected cross-site tracking in just 15 more websites than a simple analysis based on the inspection of third-party cookies alone, which is common in prior work. The new websites identified by information flow control all set client identifiers in third-party local storage and communicate them over the network.

In the end, our analysis discovered 597 distinct cross-site trackers on 1,766 websites. The most popular cross-site trackers are reported in Table 5, along with their category from Disconnect. In general, it seems that cross-site tracking is declining in popularity nowadays, at least from the vantage point of Project Foxhound. Yet, third-party cookies are still primarily used for cross-site tracking: out of 2,319 websites making use of third-party cookies, there are 1,751 websites using them to store some client identifier according to Table 1

5.2.4 Effectiveness of Cookie Heuristics

In our previous results, we implicitly relied on our heuristic attribution of read cookies keys to the detected information flows from the cookie jar to the network. This is important because it allows us to detect which cookies have been actually communicated over the network, which in turn is required to detect whether client identifiers are exchanged for web tracking purposes. We here measure the effectiveness of our heuristics: out of 46,990 flows from first-party cookies to the network, we were able to attribute at least one cookie to the flow in 43,696 cases (93%). Moreover, out of 909 flows from third-party cookies to the network, we were able to attribute at least one cookie to the flow in 861 cases (95%). This shows that, although our heuristics is sub-optimal in theory, it is practically effective in the very large majority of the cases.

5.2.5 Benefits of Information Flow Control

To better understand the benefits of information flow control, we performed a few additional experiments. The blue line in Figure 1 shows the number of trackers detected by our methodology for different values of minimum required popularity, i.e., the minimum number of websites communicating with the tracker. The orange line in the figure shows how many such trackers are included in Disconnect, hence the gray area between the two lines estimates how many potential new trackers can be discovered through information flow control. The figure shows that Disconnect offers an excellent coverage of the most popular trackers in the wild, in particular it includes all trackers found in at least 39 websites. However, there are many smaller trackers, such as Adobe Marketo Measure (formerly Bizible) and Mutiny, that are not included in the filter list and can be successfully detected through information flow control.

Moreover, we compared the effectiveness of information flow control against simpler syntactic heuristics for tracking flow detection. In this experiment, we consider all the tracking flows detected by our methodology and we count how often the client identifiers set in the client-side storage are communicated verbatim over the network. To do this, we count how many times a client identifier appears as a substring (of length at least 8) within any URL parameter of a network request, or vice-versa. This matching technique is reminiscent of that proposed by Fouad et al. [15], except that we do not hard-code known transformations of client identifiers, e.g., the specific communication pattern of Google Analytics

Table 4 Cross-site tracking practices observed on the baseline client (Project Foxhound)

Cross-Site Tracking	**#Occurrences**	**#Websites**
Detected information flows from third-party cookies to the network	909	240
- Information flows with at least one cookie associated	861	233
- Information flows communicating client identifiers	784	220
Detected information flows from third-party local storage to the network	121	77
- Information flows communicating client identifiers	76	52
Detected information flows from third-party CSSIs to the network	1,030	287
- Information flows communicating client identifiers	860	256
- Number of different trackers based on the detected flows	103	256

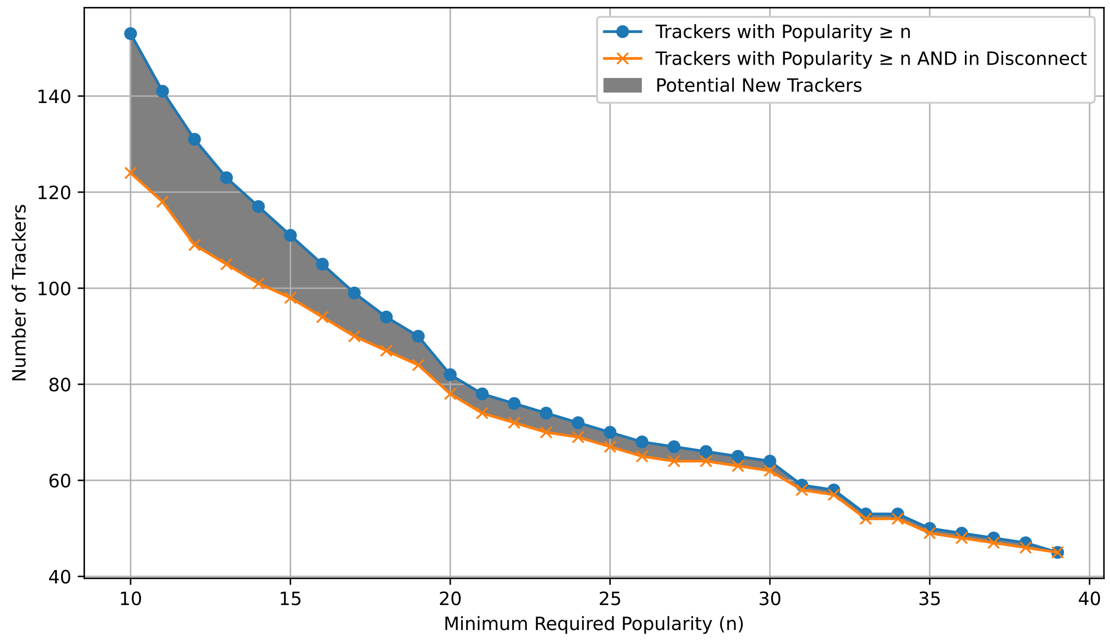


Fig. 1 Comparison between trackers identified using our methodology (blue line with circle markers) and those among them included in Disconnect (orange line with X markers), as the minimum required popularity changes.

Table 5 Most popular cross-site trackers detected through information flows on the baseline client (Project Foxhound)

Cross-Site Trackers	**#Websites**	**Categories (Disconnect)**
`doubleclick.net`	599	Advertising, Email
`criteo.com`	172	Advertising
`adsrvr.org`	100	Advertising
`twitter.com`	93	Advertising, Social
`snapchat.com`	87	Social
`googlesyndication.com`	69	Advertising
`optimizely.com`	66	Advertising
`creativecdn.com`	60	Advertising
`gemius.pl`	56	Advertising
`google-analytics.com`	54	Analytics, Email
`google.com`	47	Content
`vimeo.com`	45	Content

and the encryption scheme deployed by DoubleClick. Hence, this experiment estimated a coverage of 83%, i.e., 17% of the detected flows involve complex string transformations that cannot be detected by simple syntactic matches. This number is non-negligible and shows that simple string matching approaches may suffer from false negatives in practice. These could be recovered by means of more sophisticated techniques for syntactic matching or through a lightweight information-flow analysis, but the latter guarantees the effective dependency of a network request on a client identifier even in presence of non-trivial transformations.

5.3 Comparative Privacy Analysis

We are finally ready to measure how the use of different privacy-enhanced clients would protect web users based on our dataset.

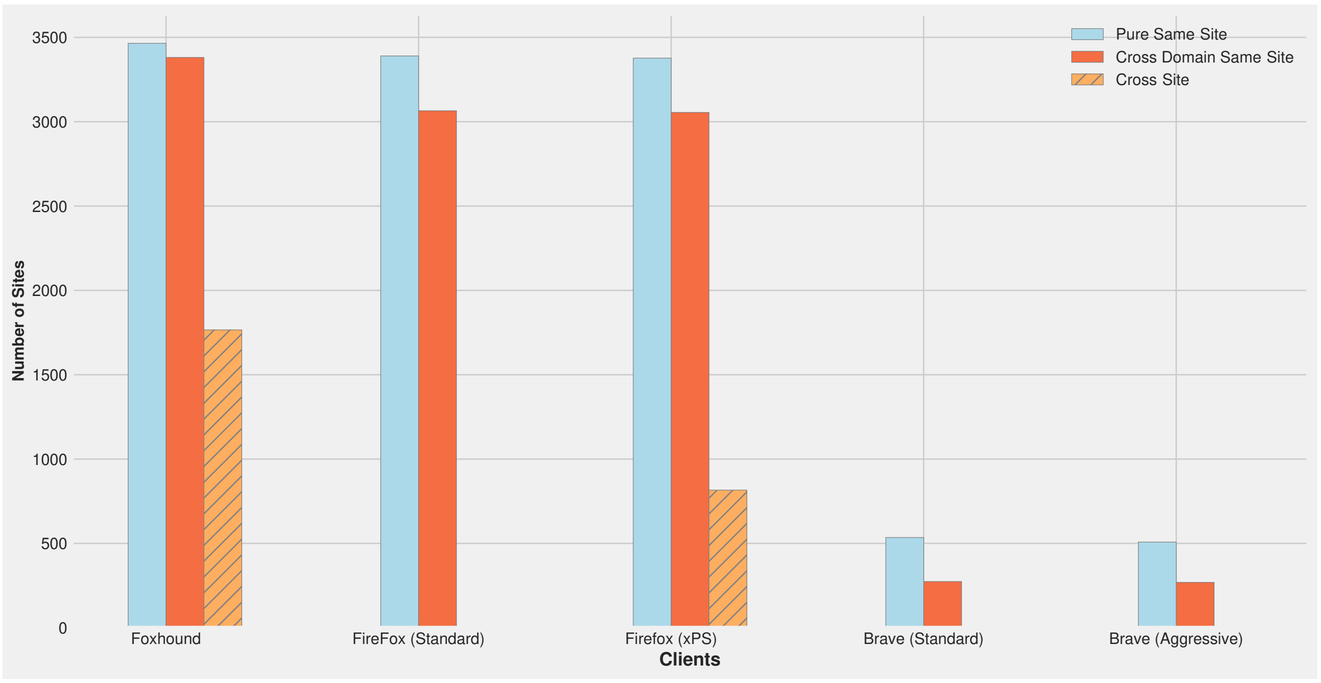


Fig. 2 Number of sites with at least one tracker across different clients

5.3.1 Tracking Reduction

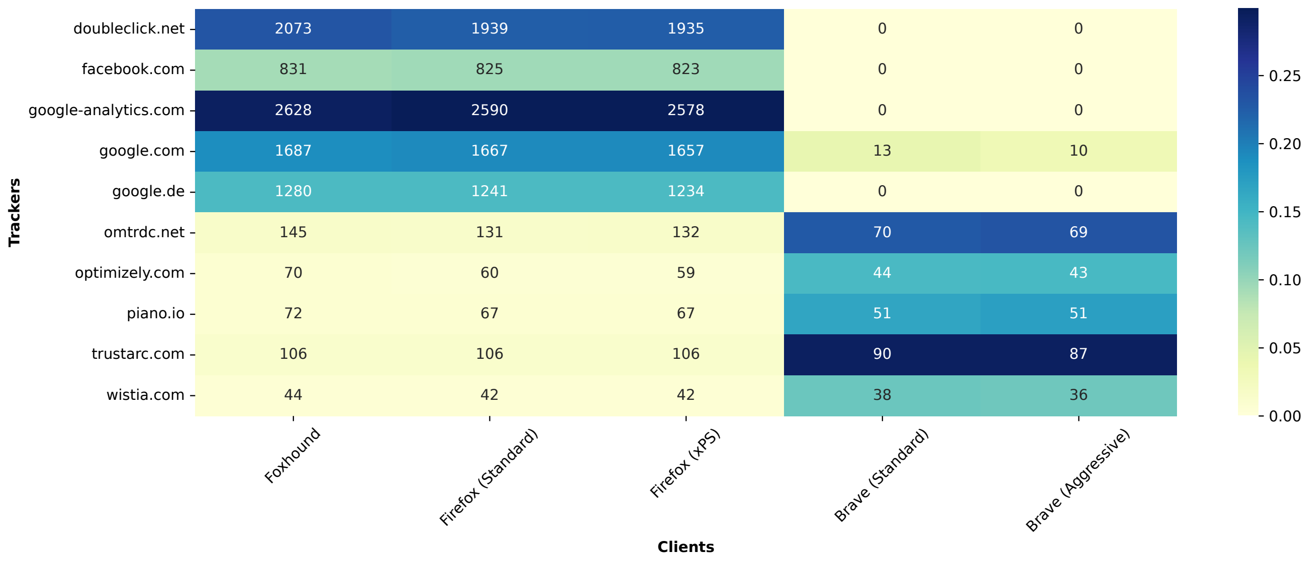
Figure 2 shows the number of websites performing same-site tracking (both pure and cross-domain) and cross-site tracking observed on different clients. The figure supports several interesting observations. First, the privacy controls of Firefox have virtually no impact on pure same-site tracking, which is the least invasive form of web tracking. This is in line with Firefox's philosophy of integrating moderate privacy controls that do not harm the user experience. Cross-domain same-site tracking instead shows a significant reduction: we identified 3,381 websites showing cross-domain same-site practices on Project Foxhound, while we found just 3,065 and 3,055 such websites on Firefox and Firefox (xPS), i.e., we estimate a reduction of around 10% in cross-domains same-site tracking when ETP is activated. We also note that the adoption of ETP significantly mitigates cross-site tracking even when partitioned storage is deactivated: we detected cross-site trackers on 816 websites on Firefox (xPS) as opposed to the 1,766 websites detected on Project Foxhound, which is a reduction of 54%. We thus conclude that the use of partitioned storage removes cross-site tracking from the remaining 674 websites, i.e., around 10% of the websites in our dataset.

As for Brave, its privacy controls turned out to be extremely effective in practice. Pure same-site tracking was detected on 535 websites and cross-domain same-site tracking was detected on 274 websites. The reduction with respect to Project Foxhound is around 85% and 92% respectively. Remarkably, we do not observe any significant difference with respect to web tracking between the standard settings of Brave and its aggressive mode. The reduction in tracking behavior is tiny, around 6% for pure same-site tracking and around 2% for cross-domain same-site tracking. This suggests that the compatibility risks introduced by the use of aggressive mode do not seem outweighed by the privacy benefits, at least in terms of stateful tracking.

5.3.2 Tracker Ecosystems

Figure 3 shows a heatmap of the most popular trackers observed on different clients. To populate the rows, we first identify the five most popular trackers in each client and we compute their union after removing duplicates. The number in cell (x, y) reports how many times client x detected tracker y on some website; the color of the cell is determined by the ratio between its content and the sum of the content of cells in the same column, which estimates the relative popularity of y observed on x. This explains why similar numbers may be associated with different colors on different clients, hence the heatmap is better read column by column to understand the relative popularity of each tracker on the different clients.

As the heatmap shows, Brave is effective at blocking the top five trackers virtually in their entirety, yet it is less successful in blocking the bottom five trackers. Because of this, the relative frequency of the bottom five trackers appears significantly higher compared to our



Trackers	Foxhound	Firefox (Standard)	Firefox (xPS)	Brave (Standard)	Brave (Aggressive)
doubleclick.net	2073	1939	1935	0	0
facebook.com	831	825	823	0	0
google-analytics.com	2628	2590	2578	0	0
google.com	1687	1667	1657	13	10
google.de	1280	1241	1234	0	0
omtrdc.net	145	131	132	70	69
optimizely.com	70	60	59	44	43
piano.io	72	67	67	51	51
trustarc.com	106	106	106	90	87
wistia.com	44	42	42	38	36

Fig. 3 Presence of same-site tracking domains across different clients

baseline client and Firefox. For example, Google Analytics and DoubleClick show high popularity in the baseline client and Firefox, but are completely blocked when using Brave. Instead, less popular trackers like TrustArc and Adobe (`omtrdc.net`) turn out to be the most frequently encountered trackers in Brave. It is worth noticing that Brave does not entirely block these trackers, as it does for Google Analytics and DoubleClick, but still the presence of such trackers is significantly attenuated in Brave. There might be two possible reasons for this. Either Brave is blocking just specific sub-domains and paths of `trustarc.com` and `omtrdc.net`, or these trackers are unrestricted, yet they are not loaded because Brave blocked some other script in charge of loading tracking content from these domains.

6 Related Work

Several studies have been conducted to analyze and investigate the tracking behavior in different web browsers and the effectiveness of the tracking protection mechanisms in them. We categorize them in three classes: traditional privacy analyses based on simple heuristics, more sophisticated privacy analyses based on information flow control and comparative privacy analyses.

6.1 Traditional Privacy Analyses

Traditional web privacy measurements are based on simple heuristics to detect client identifiers in client-side storage and check for their occurrences within HTTP requests, sometimes requiring that their targets are included in a known list of web trackers [3, 13, 29]. These approaches are helpful and popular, however they are quite coarse-grained because they do not track actual data flows, but just rely on simple syntactic matches. As such, they suffer from more false positives and false negatives than dynamic information flow control, which is specifically designed to track data flows through program logic. Moreover, filter lists are well known to suffer from limitations [15]. Also, most of the prior work in the field focused on the perspective of a single client. Englehardt et al. used OpenWPM to record observations (response metadata, cookies, behavior of scripts, etc.), and performed large scale analysis to understand the prevalence of stateful and stateless tracking [13]. Acar et al. proposed novel techniques by extracting pseudonymous identifiers from traditional storage vectors, such as cookies, as well as other vectors such as Flash storage. They track such identifiers as they spread to multiple domains to study three advanced web tracking mechanisms — canvas fingerprinting, evercookies, and the use of “cookie syncing” in conjunction with evercookies [1].

6.2 Information Flow Control for Privacy Analysis

Information flow control was not extensively applied to the web privacy setting, most probably because information flow tracking on JavaScript is particularly challenging to perform. Nevertheless, Chen and Kapravelos presented a taint tracking engine called Mystique and used it to track information leakage from browser extensions [11]. Mystique was applied to a total of 181,683 browser extensions, detecting 3,686 extensions leaking private information. In later work, Mystique was also used to investigate the use of first-party cookies for web tracking [10]. In particular, the authors estimated that around 57% of the sites in the Alexa top 10k include at least one cookie containing a unique user identifier which is exchanged with multiple third parties. Their analysis is based on a single client (Chromium) and does not consider tracking through the local storage.

In more recent work, Ahmad et al. proposed the use of taint tracking to detect privacy threats posed by web storage items [2]. They used information flow control to detect communication of client identifiers to the local storage to network sinks, however their analysis is only based on the Chrome browser. Our analysis instead is comparative and allows one to capture the relative privacy guarantees offered by different browsers. Moreover, our analysis also covers cookies and is not just limited to the web storage.

Sjosten et al. proposed EssentialFP, a principled approach to the dynamic detection of browser fingerprinting [33]. EssentialFP is based on dynamic analysis and in particular on an extension of JSFlow [17]. To capture the essence of fingerprinting, EssentialFP relies on an extensive list of browser-specific sources and looks for information flows ending in known network sinks. The efficacy of EssentialFP was illustrated through an empirical study based on two classes of web pages: fingerprinting pages (authentication, bot detection and more) and non-fingerprinting pages (analytics, polyfills, advertisement). Our work instead uses information flow control to detect stateful tracking through client-side storage.

6.3 Comparative Privacy Analyses

Zafar and Das performed a large-scale comparative analysis by collecting data from 10k sites on different mobile browsers using real mobile devices. The authors highlighted the tracking threat fingerprinting by analyzing the run-time execution traces and source codes and found that different contents are being rendered on different browsers [36]. By considering well-known filter lists as ground truth, Brave performed better by blocking the highest number of tracking content, Focus performed better against social trackers, and Duck Duck Go restricts third-party trackers that perform email-based tracking. Although the idea of performing a comparative analysis is similar to ours, there are important differences with respect to our work. First, they use simple heuristics to analyze cookies and HTTP requests, without checking actual information flows, which leads to coarse-grained observations. Second, they do not discuss how to mitigate the challenges of comparative analyses that we discussed in this work, e.g., arising from the dynamic nature of websites and non-determinism, which further affects the accuracy of the analysis. Finally, they focus on the mobile browser ecosystem, while we consider traditional desktop browsers.

Merzdovnik et al. performed a large-scale two-part measurement study to analyze the effectiveness of browser extensions in blocking third-party stateful and stateless trackers [24]. The authors discovered that rule-based extensions perform better than learning-based ones and none of the extensions are effective in blocking all the stateful and stateless trackers. A large-scale comparative web tracking measurement study of mobile and desktop non-emulated browsers using the tool *OmniCrawl* has been conducted by Cassel et al [9]. Pradeep et al. studied the privacy-harming behavior [27] by conducting a large-scale and multi-directional analysis on a large pool of Android browsers from different app stores. In order to analyze the current privacy-enhancing setups and privacy-harming settings, the authors developed a novel tool that combines static and dynamic analysis methods. The authors found privacy-harming behaviors in various apps, including browsers that claim to have enhanced privacy protection mechanisms.

Finally, a technical analysis [21] has been conducted by Knockel et al. in three Chinese browsers: UC Browser, QQ Browser, and Baidu Browser to investigate the security and privacy vulnerabilities. The authors found that code execution vulnerabilities and leaking sensitive data occur consistently in these browsers, leading to privacy breaches among users in China. UC Browser is quite concerning as it transmits sensitive user data insecurely to the intelligence community to perform surveillance.

7 Conclusion

In this work, we discussed relevant challenges to overcome when performing comparative privacy analyses estimating the amount of web privacy protection offered by different clients. In particular, we explained how the dynamic nature of the Web makes it difficult to effectively attribute observed differences in web tracking behavior to different privacy controls, concluding that simple heuristics based on the inspection of client-side storage and HTTP requests likely lead to unreliable conclusions. We thus advocated the use of information flow control as a precise tool to measure such differences in terms of actual communications of client identifiers from client-side storage to network sinks. In combination with other safeguards against possible measurement pitfalls, information flow control pays off and allows one to uniformly detect different forms of (stateful) tracking, thus enabling a principled comparison of the privacy guarantees offered by different clients.

As future work, we would like to improve our comparative privacy analysis by moving away from the specific choice of a baseline client as the privacy reference. Although we do not see major conceptual challenges in extending our methodology, this would require a significant engineering effort to efficiently support taint tracking across multiple browsers. On the other hand, this would allow us to detect sophisticated adaptive behavior where some information flows are introduced in a privacy-enhanced client to bypass its active privacy controls. Moreover, we plan to extend other methodology to also account for stateless tracking, i.e., fingerprinting, which can also be modeled and detected by using information flow control [33]. Integration with the new Topics API would also be an interesting avenue for future work. Finally, we would like to investigate the prevalence of recent abuses of first-party storage to implement cross-site tracking, e.g., through UID smuggling [28].

Data Availability

All data and code supporting the findings of this study are available at the following URLs: `https://doi.org/10.5281/zenodo.12570248`, `https://github.com/eleumasc/Comparative-Privacy-Analysis`.

Declarations

Funding. This research was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

Competing interests. The authors have no relevant financial or non-financial interests to disclose.

Ethical approval. The authors declare full compliance with ethical standards. This article does not contain any studies involving humans or animals performed by any of the authors.

References

1. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., Díaz, C.: The web never forgets: Persistent tracking mechanisms in the wild. In: G. Ahn, M. Yung, N. Li (eds.) Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014, pp. 674–689. ACM (2014). DOI 10.1145/2660267.2660347. URL `https://doi.org/10.1145/2660267.2660347`
2. Ahmad, Z., Casarin, S., Calzavara, S.: An empirical analysis of web storage and its applications to web tracking. ACM Trans. Web **18**(1) (2023). DOI 10.1145/3623382. URL `https://doi.org/10.1145/3623382`
3. Bashir, M.A., Wilson, C.: Diffusion of user tracking data in the online advertising ecosystem. Proc. Priv. Enhancing Technol. **2018**(4), 85–103 (2018). DOI 10.1515/POPETS-2018-0033. URL `https://doi.org/10.1515/popets-2018-0033`
4. (2021). URL `https://brave.com/privacy-updates/7-ephemeral-storage/`
5. (2022). URL `https://brave.com/shields/`
6. (2013). URL `https://brendaneich.com/2013/05/c-is-for-cookie/`
7. (2013). URL `https://brendaneich.com/2013/06/the-cookie-clearinghouse/`
8. (2016). URL `https://venturebeat.com/mobile/brave-browser/`
9. Cassel, D., Lin, S., Buraggina, A., Wang, W., Zhang, A., Bauer, L., Hsiao, H., Jia, L., Libert, T.: Omnicrawl: Comprehensive measurement of web tracking with real desktop and mobile browsers. Proc. Priv. Enhancing Technol. **2022**(1), 227–252 (2022). DOI 10.2478/POPETS-2022-0012. URL `https://doi.org/10.2478/popets-2022-0012`
10. Chen, Q., Ilia, P., Polychronakis, M., Kapravelos, A.: Cookie swap party: Abusing first-party cookies for web tracking. In: J. Leskovec, M. Grobelnik, M. Najork, J. Tang, L. Zia (eds.) WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021, pp. 2117–2129. ACM / IW3C2 (2021). DOI 10.1145/3442381.3449837. URL `https://doi.org/10.1145/3442381.3449837`
11. Chen, Q., Kapravelos, A.: Mystique: Uncovering information leakage from browser extensions. In: D. Lie, M. Mannan, M. Backes, X. Wang (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018, pp. 1687–1700. ACM (2018). DOI 10.1145/3243734.3243823. URL `https://doi.org/10.1145/3243734.3243823`
12. (2022). URL `https://disconnect.me/trackerprotection`
13. Englehardt, S., Narayanan, A.: Online tracking: A 1-million-site measurement and analysis. In: E.R. Weippl, S. Katzenbeisser, C. Kruegel, A.C. Myers, S. Halevi (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, pp. 1388–1401. ACM (2016). DOI 10.1145/2976749.2978313. URL `https://doi.org/10.1145/2976749.2978313`
14. (2018). URL `https://support.mozilla.org/en-US/kb/enhanced-tracking-protection-firefox-desktop`
15. Fouad, I., Bielova, N., Legout, A., Sarafijanovic-Djukic, N.: Missed by filter lists: Detecting unknown third-party trackers with invisible pixels. Proceedings on Privacy Enhancing Technologies (PoPETs) **2020** (2020). `https://doi.org/10.2478/popets-2020-0038`
16. (2016). URL `https://eur-lex.europa.eu/eli/reg/2016/679/oj`
17. Hedin, D., Birgisson, A., Bello, L., Sabelfeld, A.: Jsflow: tracking information flow in javascript and its apis. In: Y. Cho, S.Y. Shin, S. Kim, C. Hung, J. Hong (eds.) Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014, pp. 1663–1671. ACM (2014). DOI 10.1145/2554850.2554909. URL `https://doi.org/10.1145/2554850.2554909`

18. Karim, R., Tip, F., Sochurková, A., Sen, K.: Platform-independent dynamic taint analysis for javascript. IEEE Trans. Software Eng. **46**(12), 1364–1379 (2020). DOI 10.1109/TSE.2018.2878020. URL `https://doi.org/10.1109/TSE.2018.2878020`
19. Klein, D., Barber, T., Bensalim, S., Stock, B., Johns, M.: Hand sanitizers in the wild: A large-scale study of custom javascript sanitizer functions. In: 7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022, pp. 236–250. IEEE (2022). DOI 10.1109/EUROSP53844.2022.00023. URL `https://doi.org/10.1109/EuroSP53844.2022.00023`
20. Klein, D., Musch, M., Barber, T., Kopmann, M., Johns, M.: Accept all exploits: Exploring the security impact of cookie banners. In: Annual Computer Security Applications Conference, ACSAC 2022, Austin, TX, USA, December 5-9, 2022, pp. 911–922. ACM (2022). DOI 10.1145/3564625.3564647. URL `https://doi.org/10.1145/3564625.3564647`
21. Knockel, J., Senft, A., Deibert, R.J.: Privacy and security issues in BAT web browsers. In: A. Houmansadr, P. Mittal (eds.) 6th USENIX Workshop on Free and Open Communications on the Internet, FOCI '16, Austin, TX, USA, August 8, 2016. USENIX Association (2016). URL `https://www.usenix.org/conference/foci16/workshop-program/presentation/knockel`
22. Lerner, A., Simpson, A.K., Kohno, T., Roesner, F.: Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In: T. Holz, S. Savage (eds.) 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. USENIX Association (2016). URL `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lerner`
23. Mayer, J.R., Mitchell, J.C.: Third-party web tracking: Policy and technology. In: IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA, pp. 413–427. IEEE Computer Society (2012). DOI 10.1109/SP.2012.47. URL `https://doi.org/10.1109/SP.2012.47`
24. Merzdovnik, G., Huber, M., Buhov, D., Nikiforakis, N., Neuner, S., Schmiedecker, M., Weippl, E.R.: Block me if you can: A large-scale study of tracker-blocking tools. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017, pp. 319–333. IEEE (2017). DOI 10.1109/EuroSP.2017.26. URL `https://doi.org/10.1109/EuroSP.2017.26`
25. Munir, S., Siby, S., Iqbal, U., Englehardt, S., Shafiq, Z., Troncoso, C.: Cookiegraph: Understanding and detecting first-party tracking cookies. In: W. Meng, C.D. Jensen, C. Cremers, E. Kirda (eds.) Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023, pp. 3490–3504. ACM (2023). DOI 10.1145/3576915.3616586. URL `https://doi.org/10.1145/3576915.3616586`
26. (2018). URL `https://blog.mozilla.org/en/products/firefox/latest-firefox-rolls-out-enhanced-tracking-protection/`
27. Pradeep, A., Feal, Á., Gamba, J., Rao, A., Lindorfer, M., Vallina-Rodriguez, N., Choffnes, D.R.: Not your average app: A large-scale privacy analysis of android browsers. Proc. Priv. Enhancing Technol. **2023**(1), 29–46 (2023). DOI 10.56553/POPETS-2023-0003. URL `https://doi.org/10.56553/popets-2023-0003`
28. Randall, A., Snyder, P., Ukani, A., Snoeren, A.C., Voelker, G.M., Savage, S., Schulman, A.: Measuring UID smuggling in the wild. In: C. Barakat, C. Pelsser, T.A. Benson, D.R. Choffnes (eds.) Proceedings of the 22nd ACM Internet Measurement Conference, IMC 2022, Nice, France, October 25-27, 2022, pp. 230–243. ACM (2022). DOI 10.1145/3517745.3561415. URL `https://doi.org/10.1145/3517745.3561415`
29. Roesner, F., Kohno, T., Wetherall, D.: Detecting and defending against third-party tracking on the web. In: S.D. Gribble, D. Katabi (eds.) Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012, pp. 155–168. USENIX Association (2012). URL `https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/roesner`
30. Roth, S., Calzavara, S., Wilhelm, M., Rabitti, A., Stock, B.: The security lottery: Measuring client-side web security inconsistencies. In: K.R.B. Butler, K. Thomas (eds.) 31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022, pp. 2047–2064. USENIX Association (2022). URL `https://www.usenix.org/conference/usenixsecurity22/presentation/roth`
31. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE J. Sel. Areas Commun. **21**(1), 5–19 (2003). DOI 10.1109/JSAC.2002.806121. URL `https://doi.org/10.1109/JSAC.2002.806121`
32. Schoepe, D., Balliu, M., Pierce, B.C., Sabelfeld, A.: Explicit secrecy: A policy for taint tracking. In: IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016, pp. 15–30. IEEE (2016). DOI 10.1109/EUROSP.2016.14. URL `https://doi.org/10.1109/EuroSP.2016.14`
33. Sjösten, A., Hedin, D., Sabelfeld, A.: Essentialfp: Exposing the essence of browser fingerprinting. In: IEEE European Symposium on Security and Privacy Workshops, EuroS&P 2021, Vienna, Austria, September 6-10, 2021, pp. 32–48. IEEE (2021). DOI 10.1109/EuroSPW54576.2021.00011. URL `https://doi.org/10.1109/EuroSPW54576.2021.00011`
34. Staicu, C., Schoepe, D., Balliu, M., Pradel, M., Sabelfeld, A.: An empirical study of information flows in real-world javascript. In: P. Mardziel, N. Vazou (eds.) Proceedings of the 14th ACM SIGSAC Workshop on Programming Languages and Analysis for Security, CCS 2019, London, United Kingdom, November 11-15, 2019, pp. 45–59. ACM (2019). DOI 10.1145/3338504.3357339. URL `https://doi.org/10.1145/3338504.3357339`
35. (2022). URL `https://blog.mozilla.org/security/2021/02/23/total-cookie-protection/`
36. Zafar, A., Das, A.: Comparative privacy analysis of mobile browsers. In: M. Shehab, M. Fernández, N. Li (eds.) Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy, CODASPY 2023, Charlotte, NC, USA, April 24-26, 2023, pp. 3–14. ACM (2023). DOI 10.1145/3577923.3583638. URL `https://doi.org/10.1145/3577923.3583638`